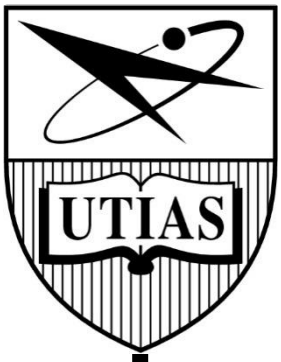


UNIVERSITY OF TORONTO

INSTITUTE FOR AEROSPACE STUDIES

4925 Dufferin Street, Toronto, Ontario, Canada, M3H 5T6

AER201 BOTTLE SORTING MACHINE

prepared by

Team 48 – Tuesday

Michael Kwok (1002699519)
Ling Long (1002516618)
Shichen Lu (1002527839)

prepared for

Prof. M.R. Emami

A technical report submitted for
AER201 – Engineering Design

Acknowledgements

A big thanks to our teaching assistants Wasi Syed and Michael Bazzocchi for their invaluable assistance.

Thanks to Professor Reza Emami for constructing this course and giving the design team the chance to engage in such a challenging and fruitful project, and also for giving us advice and helping us along the way.

And of course, our kindest thanks to our friends and classmates that helped us along the way.

Abstract

This document outlines the design of a continuous, automated bottle sorting machine, and its construction and implementation over a semester. The machine is designed to distinguish between Yop and Eska bottles and determine whether or not each bottle has a cap on. Then, based on the previous properties, the machine will sort each bottle into a specific sorting bin. Each specific subsystem of the machine is detailed in this report, followed by a description of the integration process of each subsystem. Additionally, the design methodology, budget, scheduling, and overall improvements for the design process are discussed.

The Team

Figure 1: Team 48 and the Sonic

We are team 48, consisting of (from left to right in the picture) Michael Kwok, Ling Long, and Shichen Lu.

Abbreviations and Symbols

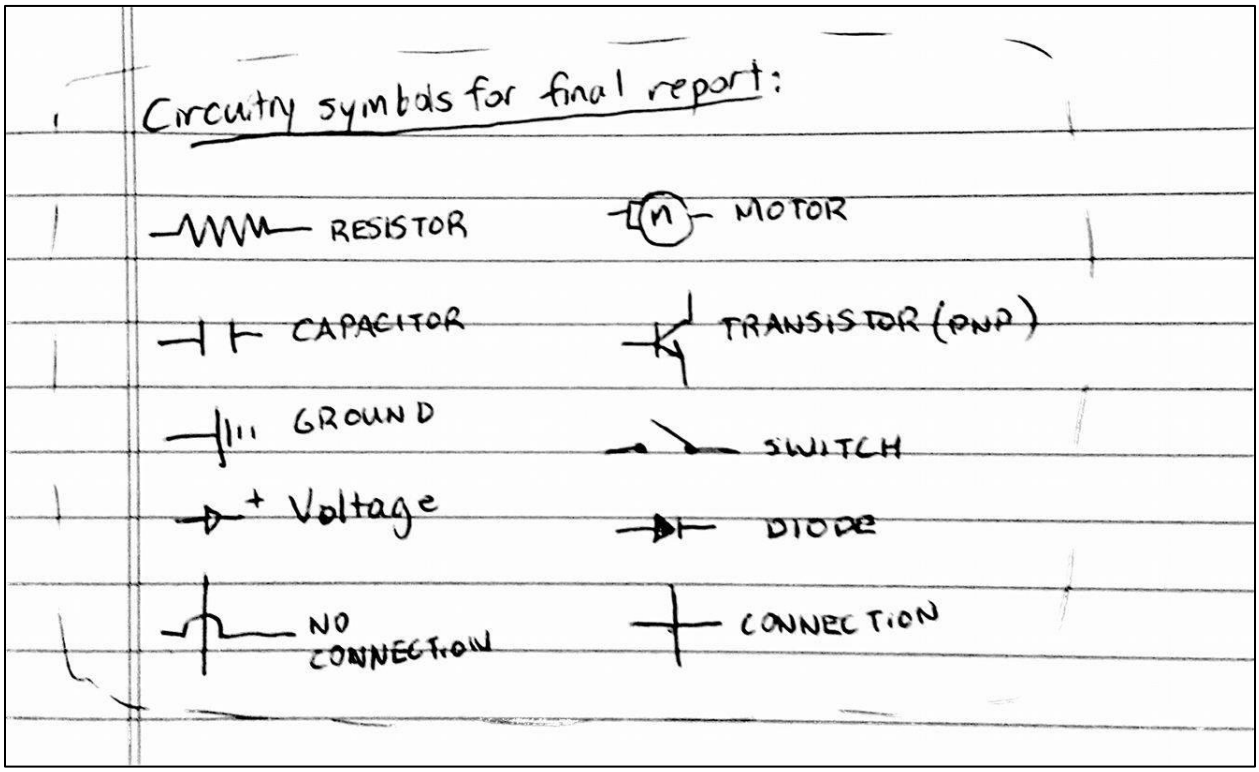


Figure 2: Circuitry Symbols used in Report

Table of Contents

1. Introduction

A recycling company is receiving bottles in a continuous, unsorted stream. These bottles must be sorted in accordance to their plastic composition (ie. made of transparent or opaque plastic) and depending on whether or not there is the presence of a bottle cap so that the correct recycling process can be used on each bottle. Sorting bottles by hand is costly, slow, and prone to human error. Therefore, an efficient and automated sorting machine is desired.

To meet the needs of the processing facility, the machine must be portable, able to process unsorted bottles loaded in 10 at a time in under 3 minutes, able to distinguish between transparent/opaque bottles, and if there is a bottle cap present on the bottle. In addition, the final sorted bottles must be placed in separate, removable, and easily accessible containers.

The proposed design for this task is the Sonic Bottle Sorter. This is a speed-oriented machine designed to automatically sort 10 bottles of mixed variety – either Eska or Yop – depending on the bottle type and the presence of a cap. Main design features include a centrifuge hopper mechanism for loading bottles, a color sensor to continuously scan the bottles and determine which category to sort each bottle into, and a set of rotating servos and guide rails to sort the bottles into respective bins.

This report details the design process and final product for this engineering challenge. Beginning with the background, motivation, and objectives, the report then moves on to detail the engineering processes and challenges behind each of the three major subsystems of this project – microcontroller, circuits, and electromechanical – before converging onto the final, integrated design.

2. Project Concept and Design Parameters

2.1 High Level Objective

Design a portable and automated bottle sorting machine that can sort plastic bottles into four separate categories, depending on the type of bottle and the presence of a bottle cap, in a timely manner.

2.2 Detailed Objectives

The detailed objectives were given as a part of the project requirements. As a group, it was decided that several of the objectives would be taken to

Table 1: Detailed Objectives

Objective	**Metric**	**Constraint**	**Criteria**
Solution is portable	Dimensions (m) Weight (kg)	Dimensions: machine fits within a 0.55m x 0.55m x 0.55m box (excl. power cable) Weight: machine weighs less than 5kg (incl. power cable)	Dimensions: smaller is better Weight: lighter is better
Solution is affordable	Cost ($)	The total cost of the parts required to build the machine must not exceed $230 CAD (excl. labour costs)	Less is better

Solution can sort bottles in a timely manner	Time (min.)	The time from when the machine begins active operation to when the machine returns to a standby state must not exceed 3 minutes per 10 bottle load (excl. load/unload time)	Less is better
Solution allows for timely bottle loading and unloading	Time (min.) Degree of machine manipulation required	Total load time per 10 bottle load must not exceed 1 minute. Total unloading time per 10 bottle load must not exceed 0.5 min. During loading and unloading, no parts of the machine should need to be disassembled.	Less is better
Solution is autonomous	Degree of human interaction required for machine operation	In processing a 10-bottle load, the only external assistance the machine can receive is the loading of the bottles into the machine and the input of a "start" signal.	n/a
Solution does not significantly damage bottes	Degree of additional scratches/deformation on bottles after machine operation	Machine must not significantly damage the bottles (ie. Deformation or scratching) during its operation	Less is better
Solution correctly sorts bottles	Number of bottles sorted to the correct containers	n/a	More is better
Solution must have a UI that meets requirements	The user interface must be: 1. Able to indicate the completion of the sorting process, 2. Able to display the total number of bottles processed, 3. Able to display the number of bottles in each sorted category, 4. Able to display the total time of operation, and 5. Robust and self-explanatory to the user 6. Contain an emergency stop button that immediately ceases all operations of the robot		

2.3 Team Design Values and Considerations

While the high level objectives and detailed objectives encompass the majority of considerations of the projects, the weight of each objective was determined through a consideration of the team design values. Ordered by importance, the three main team design values are as follows

1. Design for Effectiveness
2. Design for Simplicity
3. Design for Speed.

Design for Effectiveness is the most important value, as it dictates if the machine works or not. Design for Simplicity is next, as it was important to us to have a combination of simplicity and effectiveness. We believed it would be a much more effective implementation process if our design was relatively straightforward and concise to construct, rather than having many different complicated parts and processes. Finally, while the speed of sorting of the machine is one of the main detailed objectives of the

design, we as a team also placed additional emphasis on speed over the other quantifiable objectives due to the RFP's emphasis on speed as a limiting factor (ie. machines that run for the full 3 minutes are not preferred)

2.4 Division of the Problem

The project will be divided into three major subsystems to be individually developed and then eventually integrated. This is to allow for a more efficient work schedule, as each subsystem can initially be developed independently of the others. Additionally, this division allows for each member of the team to specialize into one aspect of the design and hopefully provide more insightful design decisions. The subsystems are divided as follows:

1. Microcontroller: Shichen Lu will be in charge of the microcontroller subsystem. This subsystem is responsible for the software component of the project, including programming of the microcontroller, driving of motor logic, and processing of bottle detection.
2. Circuits: Ling Long will be in charge of the circuits subsystem. This subsystem deals with connecting the software components to the hardware components such as motors and sensors and is responsible for construction and implementation of all required circuit components for this machine. Additionally, this subsystem is responsible for designing the power supply system for the overall device.
3. Electromechanical: Michael Kwok will be in charge of the electromechanical subsystem. This subsystem deals with the construction of the overall machine, including the frame, and various mechanical components as needed. Additionally, the electromechanical subsystem is responsible for calculations of overall design parameters to ensure that the machine can physically meet the performance requirements and not break under load.

3. Budget

This section of the proposal summarizes the cost of all the components chosen to be a part of each of the 3 subsystems (see Tables 2 through 4).

Table 2: Budgeting for Circuitry Components

Name of Component	Number of Component Needed	Total Price (tax not included)
TCS 34725 Colour Sensor	1	$7.95 https://www.adafruit.com/product/1334
Misc. circuit components (resistors, wires, etc.)	n/a	<$15 (upper bound estimation)
5V 3A AC-DC wall Adapter	1	$15.99 https://www.pololu.com/product/1461
Protoboards for soldering	2	2*$1.99= $4.00 https://www.creatroninc.com/product/half-size-protoyping-board-breadboard-type/
Step Up voltage Regulator	1	$12.99 https://www.creatroninc.com/product/12v-5a-step-up-regulator/

Rocker Switch	1	$2.30 https://www.creatroninc.com/product/dpst-rocker-switch-large/?search_query=rocker+switch&results=11
Circuits Subtotal	$58.23	

Table 3: Budgeting for Electromechanical Components

Name of Component	**Number of Component Needed**	**Total Price (tax not included)**
2" X 1" X 8' Wood	2	$1.12 * 2= $2.24 https://www.homedepot.ca/en/home/p.1x2x8-framing-lumber.1000173739.html
14" X 5' Aluminum Sheet	1	$13.99 http://www.homehardware.ca/en/rec/index.htm/Building-Supplies/Building-Materials/Roofing-Products/Underlayment/Valley-Flashing/14-x-5-Roll-12-Gauge-Aluminum-Flashing/_/N-2pqfZ67l/Ne-67n/Ntk-All_EN/R-I2610847?Ntt=Aluminum+Flashing
1/8'' X 6" Aluminum Threaded Rod	1/8	$0.47 Jacobs Hardware
Aluminum Foil Duct	1/2	$3.99 Jacobs Hardware
¼' X 2' X 2' Wood Sheet	2	$7.99 * 2 = $14.98 https://www.homehardware.ca/en/rec/index.htm/Building-Supplies/Forest-Products/Panels/Fir-Dfp/G1s/2-x-2-x-1-4-Good-One-Side-Fir-Plywood/_/N-2pqfZ67l/Ne-67n/Ntk-All_EN/R-I2811434?Ntt=Plywood+1%2F4
40mm Lazy Susan	1	$3.99 Jacobs Hardware
1/4" X 8' Wooden Dowel	1	$2.00 Jacobs Hardware
L-Shaped Mounting Bracket	5	$0.70 X 5 = $3.50 Jacobs Hardware
Shenzhen 12V DC 50 RPM Motor	1	$15.99 https://www.creatroninc.com/product/120-1-25d-metal-gear-motor-12v-50rpm/?search_query=12V+Motor&results=35
Servo Motor(SG-90)	2	$2.79*2=$5.58 https://www.amazon.ca/DIGOU-10xPcs-Helicopter-Airplane-Controls/dp/B01G5NR5AI/ref=sr_1_3?s=toys&ie=UTF8&qid=1492031191&sr=1-3&keywords=SG90

L-Shaped Mounting Bracket	1	$5.50 https://www.creatroninc.com/product/25d-gear-motor-l-shaped-bracket/?search_query=Mounting+Bracket&results=26
Yellow DC motor + Wheel Set	1	$3.49 + $4.50 = $7.99 https://www.creatroninc.com/product/dc-gear-motor-wheel-set/?search_query=Wheel&results=
4mm to 5mm Flex Shaft Coupler	1	$6.60 https://www.creatroninc.com/product/4mm-to-5mm-flex-shaft-coupler/?search_query=Shaft+Coupler&results=25
Collection Bins	4	$1.5 * 2 = $3 http://www.dollarama.com/
Screws, Washers, Nuts	Many	< $10
Epoxy	1	$4.29 https://www.creatroninc.com/product/epoxy-adhesive/?search_query=Epoxy&results=6
Wood Glue	1	$5.89 Jacobs Hardware
Electromechanical Subtotal		$ 109.90

Table 4: Budgeting for Microcontroller Components

Name of Component	**Number of Component Needed**	**Total Price (tax not included)**
Devbugger Board	1	$45 Design Kit
PIC	1	$5 Design Kit
LCD + Keypad	1	$6 Design Kit
RTC	1	$5 Design Kit
Microcontroller Subtotal		$61.00

Total: $229.13

4. Perspective Theory and Survey

4.1 Functional Decomposition:

In order to divide the problem into different sections which allow for optimization, three different mechanisms were determined to be crucial to the functionality of the machine:

1. A mechanism to properly feed/orient the bottles through the machine to be processed by a sensor along with an area for loading/feeding.
2. A mechanism to sense what kind of bottle the bottle is, and whether it has a cap or not.
3. A mechanism to feed the bottle into the correct bins after the sensor has processed whether it has a cap or not.

Each functional component of the machine was explored for the advantages and disadvantages of each respective function. Then, the combined functionality of several options was explored to contemplate the integration process of several possible components.

4.2 Theory of loading the bottles and Feed/Orient the bottle

A mechanism to load the bottles is necessary and important as a design consideration because the structure of the loading mechanism determines the structure of the overall machine. To be an effective loading mechanism, it was determined that not only does the mechanism should be compact, but also be aiding the feeding process. By prototyping a centrifuge mechanism (see figure 10), we could see that this loading mechanism was both compact and output bottles with the correct orientation at a steady pace, suitable for further processing.

It was decided that rather than sorting the bottles all at once, it is necessary to sort the bottles one at a time to accurately process each bottle and guide them to their respective bins. As such, it is determined that the most effective way is through streamlining the entire process, meaning that there will be a continuous queue of bottles waiting to be processed.

4.3 Survey of Mechanisms to Load, Feed/Orient the bottle

4.3.1 Hopper Mechanism

The premise of a hopper mechanism [1] involves a large bin for holding the bottles, with a small opening at the bottom that a single bottle is able to pass through at a time (see figure 1). Bottles are dumped into the bin, and by repeated agitation of the bottles, they will eventually tumble and fall through the opening at the bottom of the bin. The overall mechanism is quite simple. However, it may be difficult to sufficiently agitate the bin such that bottles do not get stuck and fail to fall through the opening. In addition, the rate the bottles will be processed is not consistent, due to the semi-random agitation process used to process the bottles. However, if such a mechanism could be made to work consistently without jamming, it would be very simple, efficient design.

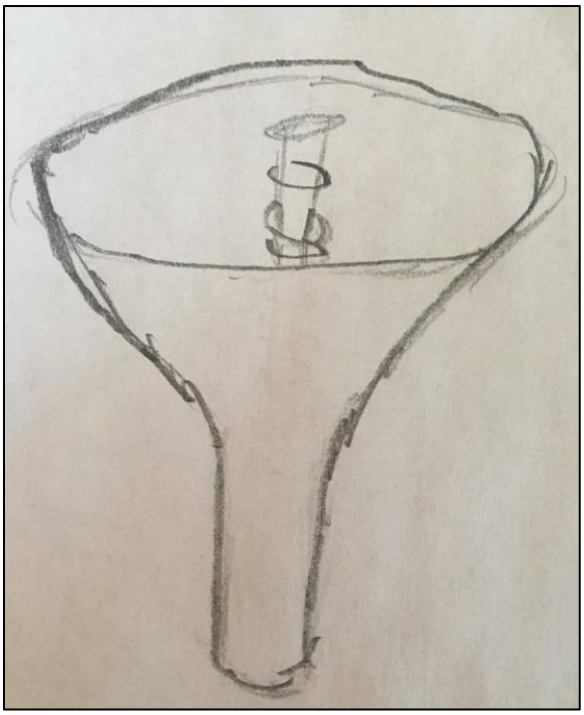


Figure 3: Hopper Mechanism Sketch

4.3.2 Centrifugal Loading Mechanism

The premise of a centrifugal mechanism involves a large spinning plate on which bottles are dumped (see figure 2). Due to the centrifugal force acting on the bottles, they are pushed towards the outer ring of the plate in a lateral orientation. There, a collecting mechanism can collect the bottles such that they are placed in a single stream. Compared to the hopper mechanism, this mechanism is bigger and more complex, but offers a much more consistent processing rate for bottles, mainly because the centrifugal force acts on all bottles simultaneously.

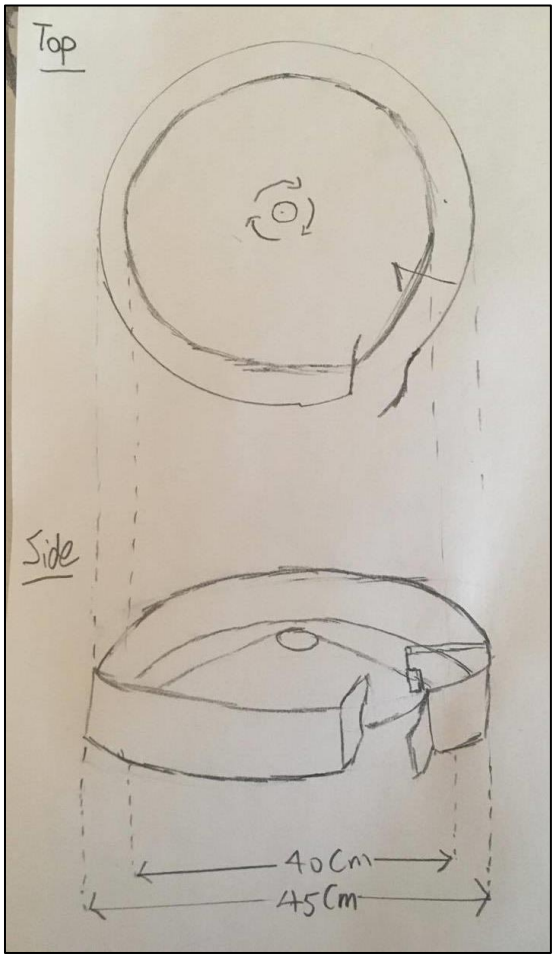


Figure 4: Centrifugal Loading Mechanism Sketch

4.3.3 Pit-Lifting Mechanism

The premise of a pit-lifting hopper is to have an elevator-esque mechanism to continuously lift bottles one by one from a bin filled with bottles (see figure 3). The "bottle elevator" then drops each bottle off at a platform such that they are sorted into a single stream of bottles. This mechanism often has issues with the bottles dropping out of the bottle elevator, and due to the nature of picking up one bottle at a time from the pit, this mechanism is relatively slow. However, unlike the previous two mechanisms, this one can process the bottles into a stream such that the bottles are oriented width-wise and roll as they travel.

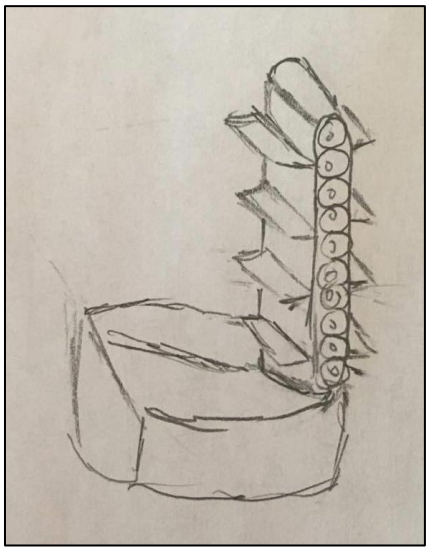


Figure 5: Pit-Lifting Mechanism Sketch

4.3.4 Summary of Bottle Loading Mechanisms Survey

A summary of what was discussed in the previous section is given in Table 5, below.

Table 5: Summary of Bottle Loading Mechanisms Survey

	Centrifugal Loading Mechanism	**Pit-Lifting Mechanism**	**Hopper Mechanism**
Pros of Design	-Known to work well if configured to not jam with existing designs - Takes up less space relative to the other mechanisms	-Very reliable, and has the least probability of jamming	-Simple, easy to build design -Can effectively load the bottles into a compact space
Cons of Design	- Needs to be relatively wide -Prone to jamming depending on the configurations - Needs to be built correctly in order to be effective.	-Large amount of mechanical components- different to implement -Takes up a lot of space due to the need	-Prone to jamming -Difficult to agitate bottles in a compact space without jamming -Difficult to modify and make improvements to existing mechanism due to simple nature of the hopper

4.4 Theory of Mechanism to Sense the Type of Bottle, and the Presence of a Cap

In order to properly accommodate the actuators to move to the right positions, sensors need to be implemented with the mechanical components of the robot. In choosing a sensor, it is not only important to determine a mechanism that can actually detect the bottle, but also whether it can be effectively implemented within the scope of our physical design.

4.5 Survey of Mechanism to Sense the Type of Bottle

This survey Requires a more in-depth consideration than the other mechanisms, because it is at the crux of the project. As such, rather than surveying physical solutions, it is first necessary to survey conceptual ideas as solutions.

The main features distinguishing the Yop and Eska Bottles are listed below (see table 4):

Table 6: Distinguishing Features of Eska and Yop Bottles

Feature	**Yop**	**Eska**
Weight, with Cap	15g	14g
Weight, without Cap	14g	13g
Shape	Cylindrical	Cylindrical
Diameter	5.0cm	5.8cm
Height	14.6cm	15.6cm
Color(bottle)	White	Transparent
Transparency	Opaque	Transparent
Material	High-Density Polyethylene	Polyethylene Terephthalate
Color(label)	Red	Blue

As a conceptual idea, each feature was considering as a possibility in our design. The following metrics were used to decide whether a feature should be used in the design to distinguish between the bottles; these are further detailed in table 5:

1. Cost of equipment needed to distinguish the feature
2. Effectiveness and range of equipment regarding the specifications of our bottles and project
3. Ease of implementation of feature into mechanical design (constraints on size and shape fitting into actual machine design)
4. Simplicity of design (complexity of implementation)
5. Reliability of design (ease of replacement and risk of failure)

Table 7: Conceptualization Table for Bottle Type Detection

Distinguishing Feature/Solution	Cost	Effectiveness	Ease of Implementation	Simplicity	Reliability
Weight/Pressure Sensor	Cheap, ~$10 for load sensor	Relatively ineffective, sensitivity of sensor vs difference in weight low	Not easy, need to make a mechanical method for bottle to be weighed	Relatively simple circuit implementation	Reliable
Shape/Camera & Software Setup	Varying	Relatively ineffective, as	Not easy, especially on software side	Difficult	indeterminate

		the shapes are similar			
Transparency/ Reflectivity sensor	Cheap, <$5 for an IR transmitter/ receiver	Relatively effective, as	Relatively easy	Simple	Difficult to determine
Color/Color Sensor	Medium cost for our purposes (~$10)	Relatively effective considering the significant color difference	Relatively easy	Very simple	Reliable as determined by testing

4.6 Survey of Mechanism to Sense the Presence of a Bottle Cap

In order to detect whether a bottle has a cap or not, the distinguishing features between a bottle with a cap and no cap must be considered:

1. Size (height of bottle and width of cap area)
2. Color in area of cap
3. Weight
4. Shape (having no cap creates an opening in a bottle)

Again, the following metrics were used to decide whether a feature should be used in the design to distinguish between the bottles (see table 6):

1. Cost of equipment needed to distinguish the feature
2. Effectiveness and range of equipment regarding the specifications of our bottles and project
3. Ease of implementation of feature into mechanical design (constraints on size and shape fitting into actual machine design)
4. Simplicity of design (complexity of implementation)
5. Reliability of design (ease of replacement and risk of failure)

Table 8: Conceptualization Table for Cap Detection Mechanism

Solution Mechanism	Cost	Effectiveness	Ease of Implementation	Simplicity	Reliability
Pressure Sensor	Cheap, $10 for load sensors	Relatively ineffective, sensitivity of sensor vs difference in weight low	Not easy, need to make a mechanical method for bottle to be weighed	Relatively simple circuit implementation	Reliable
Shape detection	Varying depending on implementation	Relatively ineffective, as the shapes are similar	Not easy, especially on software side	Not simple	Difficult to determine
Reflectivity sensor	Cheap, <$5 for an IR	Relatively effective, as the	Need to orient the bottle to	Relatively okay	Difficult to determine

	transmitter/ receiver	sensors have a decent range and the caps	be positioned facing towards sensors		
Color Sensor	Low cost (~$10)	Very effective	Relatively easy	Simple	Reliable
Acoustic Sensor	Medium Cost (~$20)	Unknown	Difficult	Not simple	Difficult to determine

Although the theory of the mechanism was important, ultimately we decided upon using a color sensor as a group, because of the method that we decided to read the bottles being a continuous data stream as the bottle passes the color sensor in the middle of the robot.

4.7 Theory of Mechanism to Sort the Bottle

After being sensed and passing through the main body of the machine, the bottles need to be separated into their respective bins. This requires some sort of actuator and mechanical mechanism working together.

4.8 Survey of Mechanism to Sort the Bottle

4.8.1 Paddle Based Sorter

This sorting mechanism has two sets of paddles. The first paddle will send a bottle coming into the sorter into either the right or the left column of the sorter, after which another paddle in each of the columns will again separate incoming bottles into one of two slots (see figures 6 and 7). In total, this sorter will be able to put a bottle into one of four slots. This sorter has a high sorting speed that depends on how fast the paddles can move from one position to the other, however this sorter takes a large amount of physical space, and requires careful timing, as the bottles move through the sorter without stopping. In addition, the circuitry implementation for this sorter is fairly complex due to the amount of paddles and the bi-directional motion they require.

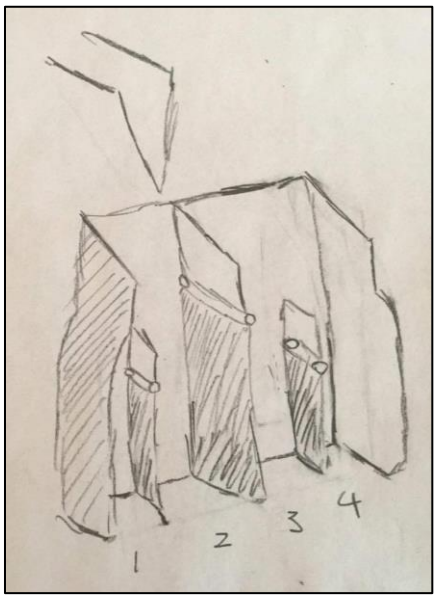


Figure 6: Paddle-Based Sorter

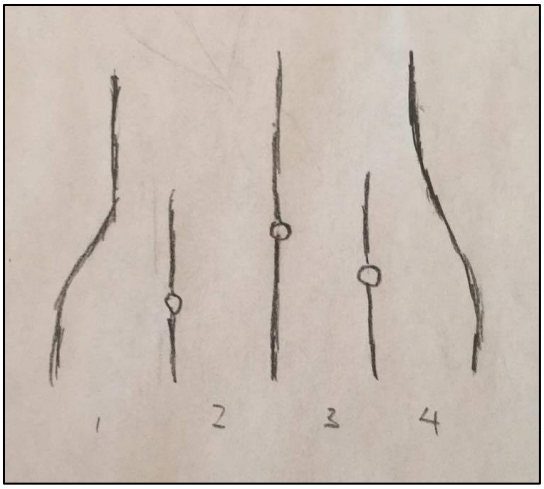


Figure 7: Paddle Setup for Paddle-Based Sorter

4.8.2 Rotating Sorter

This sorting mechanism contains a bottle stopper and a rotating tube [4]. The bottle enters the sorter and is stopped by the stopper. The rotating mechanism can then rotate the tube to point towards a certain sorting box (see figure 8). Afterwards the stopper releases the bottle and the bottle falls through the tube into the sorting box that the tube points to. This sorter can sort items into several different boxes, depending on the arrangement of the sorting boxes around the sorter. The stopping mechanism in this sorter also allows for more leniency in the timing and configuration of the sorter in its operation. However, this sorter has a relatively slow speed that is determined by how fast the rotating mechanism can rotate the tube to point to the correct sorting box.

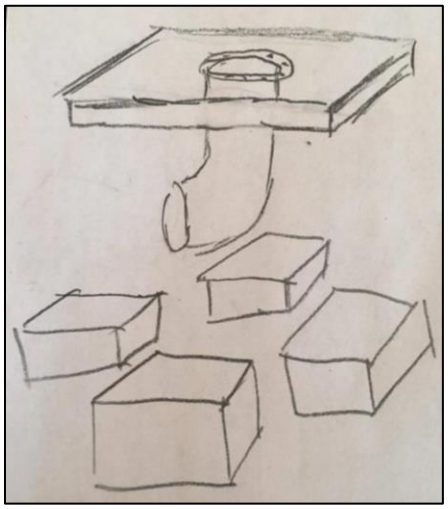


Figure 8: Rotating Sorter

4.8.3 Ramp Based Sorter

This sorting mechanism employs a ramp that directs where a bottle can go. By adjusting the rotation of the ramp using a motor, a bottle flowing into the sorting mechanism can be directed into multiple different sorting bins (see figure 9). Its simplicity allows for a very easy to implement solution. However, compared to the rotating sorter, the range of motion smaller. Additionally, this solution limits the speed

of the bottle processing by the speed of the motor turning, and this is especially slow when the motor needs to change the direction of the ramp from one end to the opposite end.

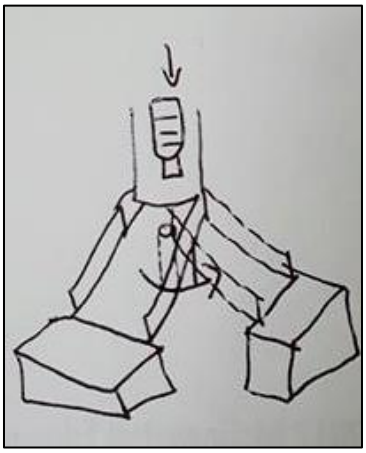


Figure 9: Ramp-Based Sorter

4.8.4 Size Based Mechanical Guide Rail Sorter

This mechanism is the only mechanism that is purely mechanical, but must be integrated with another method in order to sort whether a bottle has a cap or not. This mechanism uses rails that are calibrated to be the right size so that when Yop bottles enter the mechanism normally, they fall through the rails, and when the Eska bottles enter the mechanism, they are large enough to pass the rails.

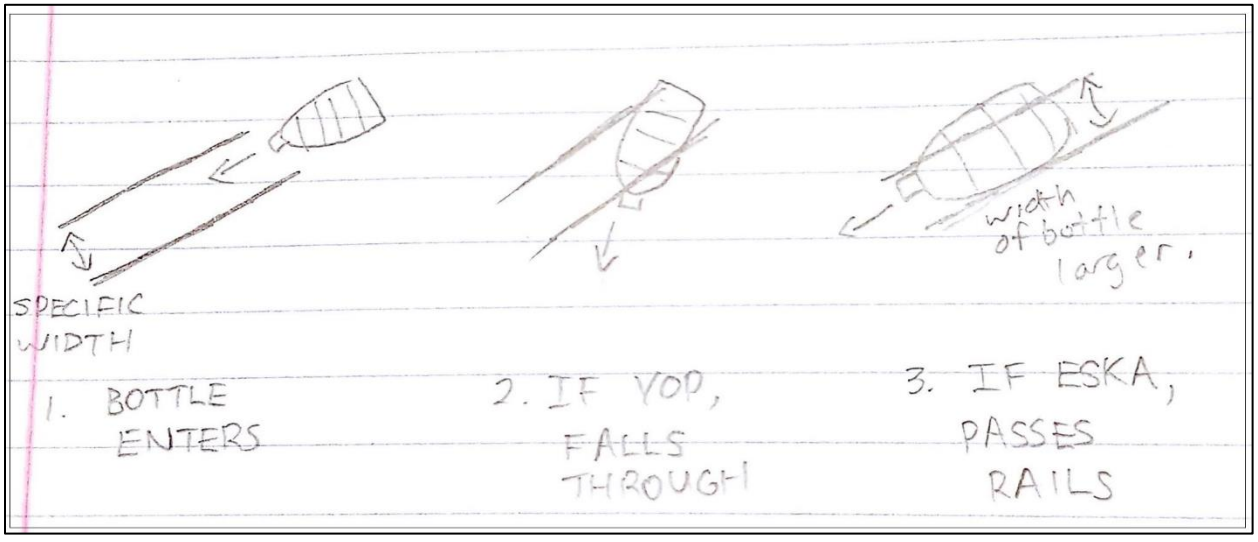


Figure 10: Guide Rail Sorter

A simple mechanism mentioned previously would be sufficient to then be placed under the rails, and sort whether the bottle has a cap or not. The biggest benefit of this design is that it limits the detection to have to only detect whether the bottle has a cap or not, as it is irrelevant if the bottle is yop or eska for sorting purposes. It also takes 0 time to react the the sensor, as it automatically sorts yop or eska without having to receive a signal from the microcontroller.

Table 9: Summary of Sorting Designs

	Ramp Based Sorter	**Rotating Sorter**	**Paddle-Based Sorter**	**Size-based Guide rail sorter**
Pros of Design	-Very simple, only requires one motor -Reliable -Easy to construct	-Simple mechanical design, requires only motor -Flexibility with choice of rotating ramps or positioning of boxes	-	-Requires no electronic components -Requires no signal from microcontroller, eliminates problem to cap detection.
Cons of Design	-Requires a 180 degree rotation for certain cases -Requires room for the ramp to be sufficiently steep for the bottle to flow -Need to stop bottle	-Requires a 180 degree rotation in some cases, may be slow -Requires the bottle to stop before dispensing	-Requires multiple servo motors rather than just one -Servos will need to turn fast, and requires a lot of space to fit more than one set of paddles	-Requires Secondary mechanism for caps - Very sensitive to calibration

4.9 Market Survey

While the goal of the background theory and survey was intended to explore different options for each component of the robot, integration remains difficult to determine based on a simple qualitative judgement of each mechanism. Rather, it is beneficial to survey existing reference designs and observe how their mechanisms were integrated together.

4.9.1 Market Survey: Reference Designs

4.9.1.1 Reference Design: AER201 Vial Sort Robot

Figure 11: AER201 Vial Sort Robot

This reference design sorts plastic PET vials based on transparency and whether they have a cap or not, very similar to the project that was assigned to the design team. The mechanism to load/orient the bottles is a flat centrifuge, which continuously spins to load one bottle at a time into a slotted queue. The bottle flows down a tube to a rectangular container, which has light sensors on either side to detect whether there is a cap or not, and the type of plastic the container is. Once the bottle has been sensed, a set of motors turn a set of ramps in a certain direction to direct the bottle to the correct container. The bottle moves out of the rectangular container after being pushed by a solenoid, and the tube opens to allow the next bottle in the queue to fill the rectangular container.

Table 10: Pros and Cons of Vial Sorting Design

Pros of this design	Cons of this design
-Stopping the bottle in a set position allows sensing to be accurate and consistent -Using a solenoid to push one bottle at a time and queue bottles means there is less risk of bottles getting jammed in the machine -Using a centrifuge design with a slotted opening allows for accurate queueing	-Stopping the bottle to be sensed slows down the overall speed of the sorting -Uses multiple motors to drive an exit bottle mechanism where less could be used -Using a straight ramp to queue the bottles requires a lot of physical space in the machine

4.9.1.2 Reference Design: TOMRA PET Bottle Sorting Plastics Machine

This design employs a conveyor belt where all the bottles are fed into a conveyor belt at a high speed, and when it reaches the conveyor belt the bottles are ejected a certain distance based on the clarity of the plastic. A sensor reads the bottle type before the end of the conveyor belt, and the timing is calibrated so that a notch at the end of the conveyor belt knocks the bottle up to a further bin if the bottle is clear, and simply lets the bottle pass if the bottle is not clear or light blue.

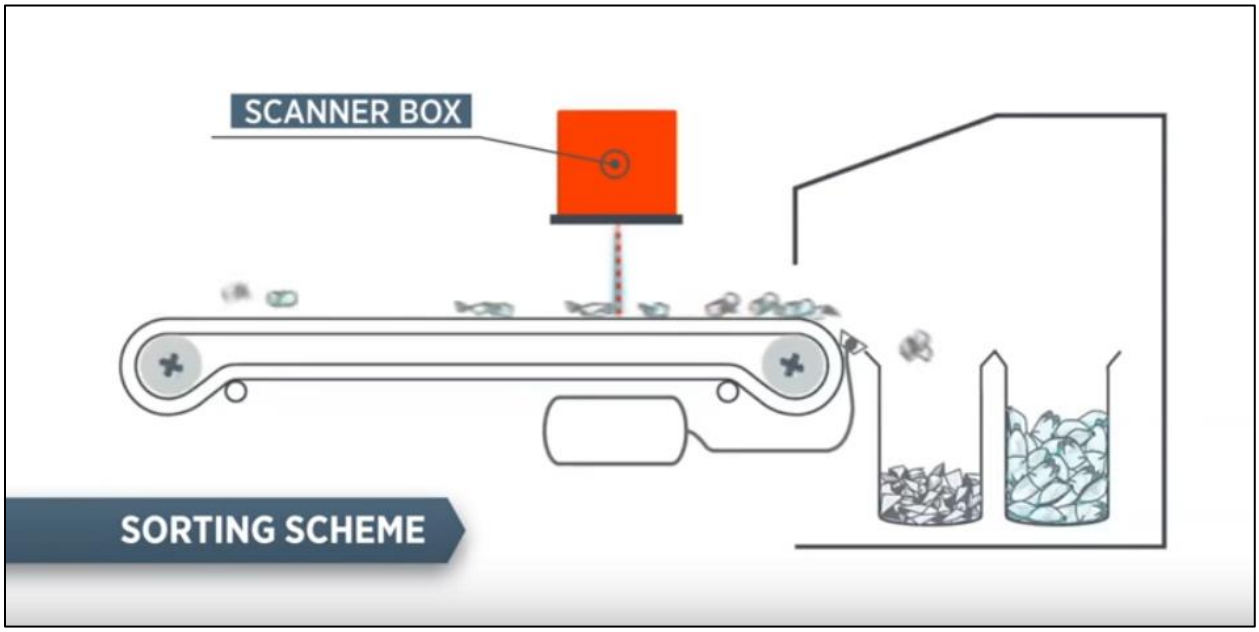


Figure 12: TOMRA Sorting Machine

Table 11: Pros and Cons of TOMRA Sorting Machine

Pros of this design	Cons of this design
-Having a conveyor belt with a set, fast speed allows for extremely fast sorting -Highly accurate due to accurate machine calibration -Requires only one sensor that continuously runs during operation	-Has limited extendibility to sorting more than clear versus non-clear bottles; hard to implement more than the sensor already in place -Difficult to sort more than 2 types of bottles due to exit mechanism -Could be inaccurate and hard to implement due to speed and closeness of exit mechanism. -Is expensive and difficult to implement, requires the construction of a conveyor belt

4.9.1.3 Reference Design: 2010 ASME Design Competition Waste Sorter

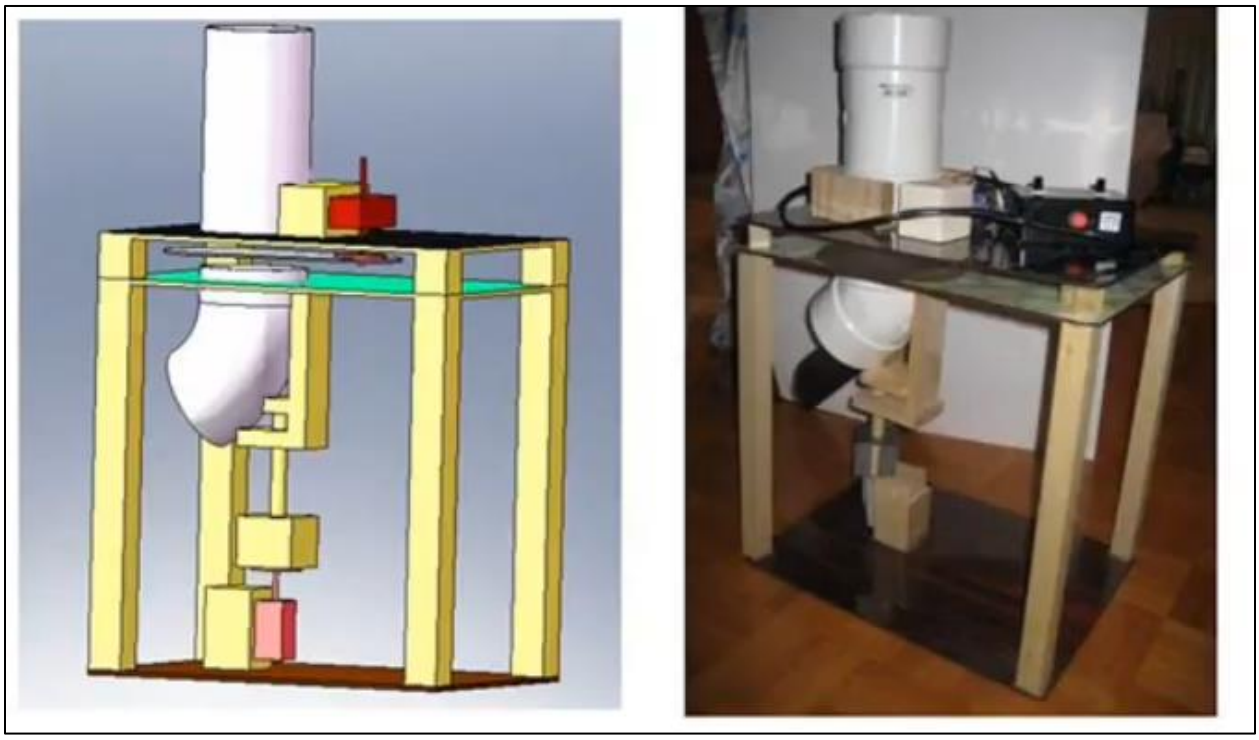


Figure 13: ASME Design Competition Sorting Machine

This reference design involves a curved tube mounted onto a shaft so that it may turn in 4 different directions to unload to bins. Bottles are dropped in one at a time, and are stopped by a gate controlled by a motor near the exit of the tube. A Reflectivity sensor determines what type of bottle it is, and once the bottle has been determined, the gate opens after the tube has been rotated to the correct contained for the bottle. Then, after the gate opens it closes again to allow another ball to be queued up.

Table 12: Pros and Cons of ASME Design Competition Sorting Machine

Pros of this design	Cons of this design
-Stopping the bottle in a position where it can be easily accessed by sensors allows for greater accuracy in detection -Simple mechanical design with few parts, and only requires 2 motors to run all of the moving parts. -Highly functional -Has extendability to other bottles with only a change in code	-Has a limited speed at which the bottles can be sorted due to the gate stopping the bottles going down the tube -Can only sort one bottle at a time, and cannot queue up bottles to be sorted, they must be automatically dropped so the machine is not fully autonomous -The Tube has a limited speed at which the motor can turn, limiting the speed of sorting after te bottle has been sensed.

4.9.2 Market Survey: Recap

Overall, a survey of existing designs reinforced the variety of solutions that exist for each of the components of our functional decomposition. By surveying the industry and past robotics project, we were able to come up with alternative solutions to certain problems that needed to be overcome in the robot. In terms of overall design, it is difficult to say whether having overall reference designs was beneficial to the project, because:

1) The constraints of this project limit our size, weight, and budget to be significantly less than that of a traditional industry robot
2) We have neither the resources nor the means to produce mechanical parts of extremely high quality
3) The constraints of this project are significantly different from any industry plastic sorter which mainly focus on PET
4) Integrating different solutions will have different interactions within the scope of our project than in industry, especially with mechanical components

Certainly, many aspects of these reference designs were taken into consideration when considering certain design choices during functional decomposition.

5. Microcontroller

5.1 Overview

The main microcontroller subsystem is responsible for the logical processing and operation of the electromechanical parts in the robot. In addition, the microcontroller subsystem will include the implementation of a user interface for machine operation, the implementation of a date/time program, and a record of bottle sorting logs generated by the machine.

The microcontroller used for this machine is the PIC18F4620. Through the PIC Devbugger Board, the microcontroller will interface with the motors and sensors on the machine, as well as an LCD and keypad for user I/O.

The MPLAB X IDE was used to allow code for the PIC18F4620 microcontroller to be written in C and then cross-complied into Assembly for programming onto the chip. A complete copy of the program code can be found in Appendix B and can be referenced for more detailed descriptions of each function that is described in section 5.3.

5.2 Assessment of the Problem

The problems that the microcontroller subsystem must tackle are as follows:

1. Reading and processing of sensor inputs
2. Driving motor logic
3. User interface
4. Utility functions (ie. Date/time, operation logs)

5.2.1 Reading and Process of Sensor Inputs

The microcontroller must read and process the input from the TCS34725 color sensor to determine the type of bottle being processed and the presence of a bottle cap. Due to the continuous and quick flow of bottles

past the color sensor and the fact that the bottles reach the sorting apparatus shortly after passing the sensing area, the microcontroller must additionally be able to accomplish the bottle detection task in a short time frame, and in a continuous manner.

5.2.2 Driving Motor Logic

The microcontroller will be responsible for driving the transistor logic that controls the motor operation in both the centrifuge motor and the yellow DC wheel motor. Additionally, the microcontroller will also drive the PWM signal output for the servo motors that control the sorting apparatus. Again, due to the continuous operation of the robot, the logic for all the motors must be able to run continuously alongside the program. This is an especially important point to consider for the servo motors, as they must constantly shift position depending on the type of bottle being sorted, and the PWM signal must constantly maintain the servo position without delaying program operation. As a safety mechanism, the microcontroller should also stop all motor functionality after the machine idles for a certain period of time.

5.2.3 User Interface

The microcontroller subsystem will need to be able to provide basic input/output interactions with a user for operation of the machine. While the machine will operate autonomously, the user must be able to manually control the start and stop of the operation. Additionally, the machine should be able to display a report of the operation details after an operation is finished, including the number of bottles sorted into each category, the total number of bottles sorted, and the total operation time.

5.2.4 Utility Functions

In addition to the features essential to the operation of the machine, the microcontroller will also have supplementary features. This includes a date/time display to display the current date and time, a record of past operation logs in memory, and a display of the total operation time per run of the machine.

5.3 Solution

As previously mentioned, the microcontroller subsystem uses the PIC18F4620 microcontroller, along with a keypad and LCD display to interface with the user. The microcontroller I/O pins are used to drive motor logic and collect information from the sensors. Solutions to each problem will be discussed in more detail below.

5.3.1 Main Program Overview

The main program that is run on the microcontroller is designed to operate based on several states. In each operating state, the program preforms a different function, and displays a different message on the LCD to indicate the state that the program is in. The various states continuously loop back onto themselves until an interrupt function switches the microcontroller state. The main program consists of five main states: *standby*, *operation*, *operation end*, *operation report*, and *operation time*. Additionally, the program has an individual state for each of the supplementary features.

After a power on or reset, the program executes a startup sequence to configure the various program parameters and then boots into the *standby* state, in which it passively waits for user input. Detection of a corresponding keypad press switches the program into the *operation* state, in which the program starts the centrifuge and wheel motors, and begins polling the color sensor for input. During this state, a bottle passing by the color sensor will also trigger the bottle processing algorithm and sort the bottle by driving the servo motors. After the bottle processing is done, or if the operation is manually stopped by the user, the program enters the *operation end* state. In this state, the program will stop all motors and sensors, process the data from the operation, and save data onto memory. From here, the user may then access both the *operation report* and *operation time* states via corresponding keypad buttons to see the bottle processing report and

the time taken for the processing, respectively. Finally, the states for date/time display and operation log display can be accessed by corresponding keypad buttons at any time.

For a more detailed look at the program operation, please refer to the pseudo-code in section 5.3.2 and the program flowchart in Figure 14.

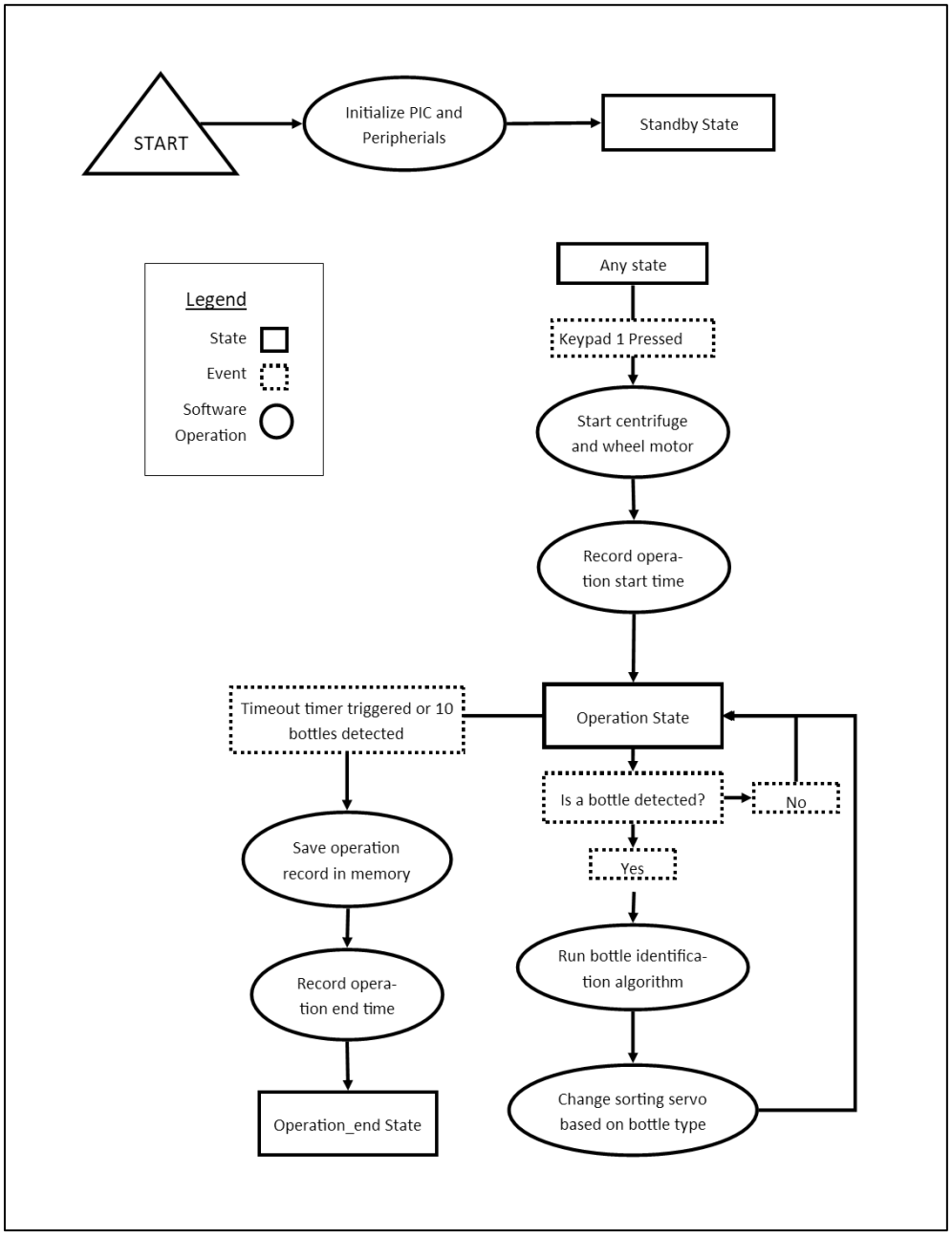


Figure 14: Program Initialization and Operation Flowchart

5.3.2 Pseudocode

```
FUNCTION main
    Initialize ports
    Initialize global variables
    Initialize keypad interrupts
    Initialize I2C
    Initialize LCD
    Initialize Color Sensor
    Initialize Timers           //timeout timer stops operation after idle
                                //two PWM timers control servo signal

    Initialize state variable
    set state to standby
    WHILE true
        read state variable
        run state function
    END WHILE
END FUNCTION

FUNCTION interrupt
    CASE interrupt_ID OF
        keypad_1: set state to operation
                  start centrifuge and wheel motor
                  store start time
                  start timeout timer
        keypad_2: set state to operation_report
                  load most recent operation report logs
        keypad_3: set state to operation_time
                  calculate operating time from start and end times
        keypad_A: set state to date_time
        keypad_4: set state to operation_report
                  load last operation report logs
        keypad_5: set state to operation_report
                  load 2nd last operation report logs
        keypad_6: set state to operation_report
                  load 3rd last operation report logs
        keypad_B: set state to operation_report
                  load 4th last operation report logs
        keypad_7: set state to operation_end
                  stop centrifuge and wheel motor
                  store end time
                  stop timeout timer
                  save operation log in memory
        timeout_timer: go to operation_end state
                       save operation log in memory
        PWM_timer_1: send signal for servo 1
        PWM_timer_2: send signal for servo 2
    END CASE
END FUNCTION

FUNCTION standby
    print "standby"
END FUNCTION

FUNCTION operation
    IF (10 bottles processed)
        go to operation_end state and save operation log in memory
    store last color sensor reading
    read color sensor
    IF (bottle in front of sensor)
        IF (blue color reading > red color reading AND bottle top has not been read)
            set flag_eska_cap
            //Indicates that the bottle is an eska with cap
        IF (red color above threshold OR green color above threshold)
            set flag_yop_nocap
            //Distingushes between a yop with no cap or eska with no cap
```

```
        IF (bottle front is in front of sensor)
            store color reading type (mainly red, mainly blue, or neutral) for bottle front
        ELSE IF (bottle back is in front of sensor)
            store color reading type (mainly red, mainly blue, or neutral) for bottle back
    ELSE IF (bottle has finished passing by sensor)
        increment total bottle count
        reset timeout timer
        IF (bottle front or back read as mainly blue OR flag_eska_cap is set)
            increment eska with cap count
            rotate servo to sort bottle to eska with cap bin
            //Servos are rotated by changing signal generated from PWM_timer_1 and 2
        ELSE IF (bottle front or back read as mainly red)
            increment yop with cap count
            rotate servo to sort bottle to yop with cap bin
        ELSE IF (flag_yop_nocap is set)
            increment yop without cap count
            rotate servo to sort bottle to yop without cap bin
        ELSE
            increment eska without cap count
            rotate servo to sort bottle to eska without cap bin
        clear flag_yop_nocap
END FUNCTION

FUNCTION operation_end
    print "Operation Done!"
END FUNCTION

FUNCTION operation_report
    print loaded operation report logs
    upon repeated call, cycle through report log pages
END FUNCTION

FUNCTION operation_time
    print operation time
END FUNCTION

FUNCTION date_time
    read date and time from Real Time Clock
    print date and time
END FUNCTION
```

**note that pseudocode only details functionality of unique functions in this microcontroller program (ie. I2C, EEPROM functions are omitted)

5.3.3 Pin Assignment

To communicate with sensors, I/O devices, and drive motors, certain pins on the microcontroller are designated for certain tasks. A list of these tasks can be found in the following table (Table 13).

Table 13: Pin Assignments for Microcontroller

Pin Name	*I/0*	*Analog/Digital*	*Description of Assignment*
RA2	O	D	Send Signal for Centrifuge and Wheel Motor
RB1	I	D	Receive Keypad Interrupt
RB4:7	I	D	Receive Keypad Data
RC3:4	I	D	I2C Interface for TCS34725
RD2:7	-	-	Interface with LCD display
RE0:1	O	D	Send PWM Signal for Servos

5.3.3 Reading and Processing of Sensor Inputs

5.3.3.1 Reading the Color Sensor

To read and write to the TCS34725 color sensor, information is relayed through the I2C communication protocol. This is interfaced through the SCL and SDA pins on the microcontroller, which correspond to RC3 and RC4, respectively. The communication channel is initialized once at the start of the program in a single byte write mode to set the data registers that configure the color sensor. Each register is configured by first sending a command code to select a register to perform operations on and then sending a value to the color sensor, which gets written to the selected register. See the following figure (Figure ___) for a detailed look at the communications protocol for the color sensor, and Appendix _____ for the specific code used to implement the I2C protocol.

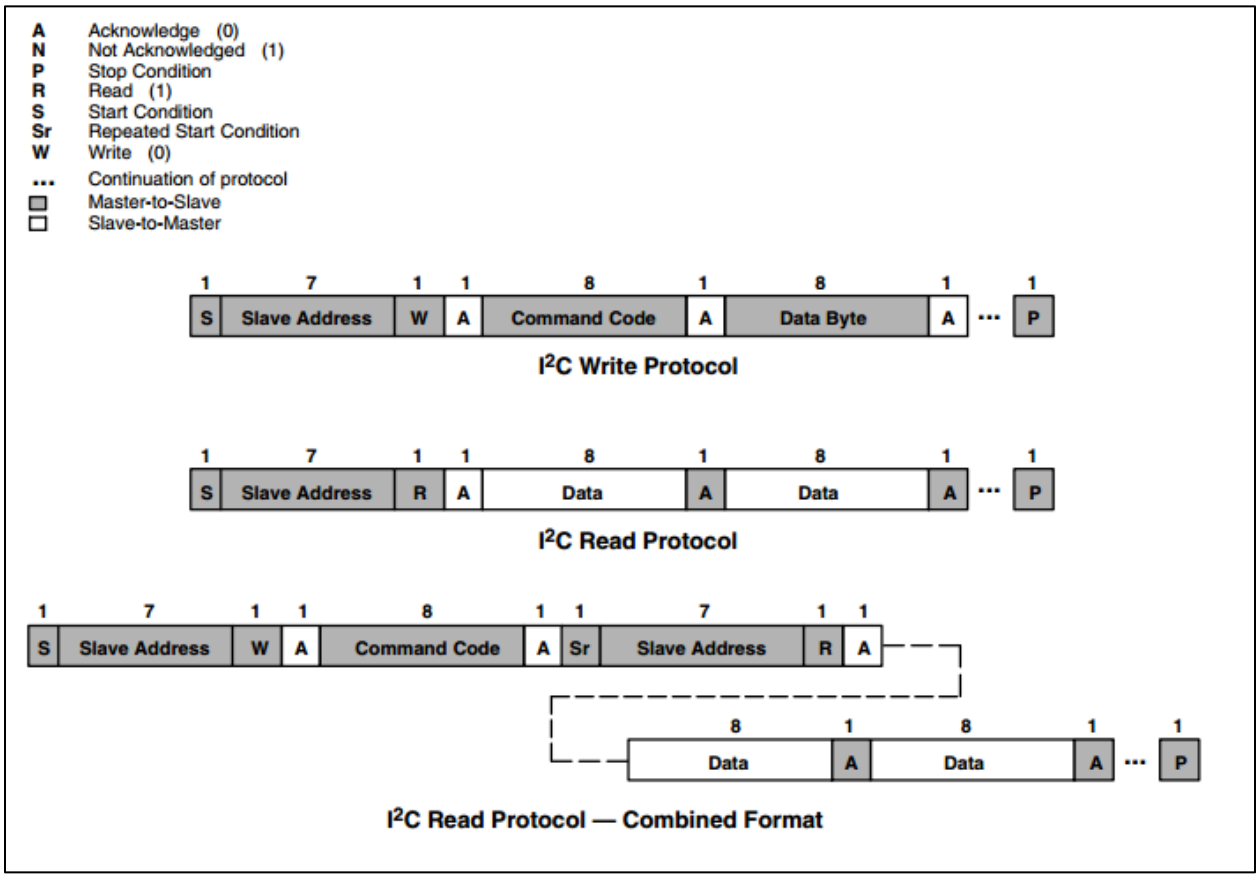


Figure 15: I2C Protocols for TCS34725

In the configuration stage for the color sensor, three main registers are written to:

1. The ENABLE register is written to in order to power on the color sensor and start its RGBC sensing function.
2. The ATIME register is written to in order to set the integration time for one RGBC cycle to 2.4ms (ie. the color sensor will output one color reading every 2.4ms)
3. The AGAIN resister is written to in order to set the analog gain for the color sensor to 16x

The reasoning behind configuration 2 is that the color sensor needs to operate extremely fast in order to generate a suitable reading for bottles that are continuously moving past the sensor. Therefore the fastest

integration time for the color sensor was selected. However, while short integration times increase the rate at which the color sensor generates color readings, it decreases the value of the color readings generated to the point which the color data is unsuitable for processing. To offset this, the 3rd configuration step is needed to increase the gain of the color sensor and up the values of the readings it generates.

The color sensor is read using the I2C Read Protocol shown in Figure 15. By using an auto-increment transaction protocol, we are able to read the color sensor values, which are stored in subsequent registers, using one continuous I2C read protocol. This again speeds up the operation of the color sensor to allow for continuous bottle scanning.

5.3.3.2 Processing Color Sensor Data

After the color sensor data is read, it is processed using an algorithm on the microcontroller to determine the type of bottle that has just passed the color sensor. The algorithm was determined by collecting large amounts of data on color readings as bottles pass by the color sensor and then recognising for certain color reading patterns indicative of certain bottle types. The main properties of the algorithm are as follows:

1. The “clear” reading from the color sensor is indicative of the transparency of the medium in front of the color sensor. Therefore, is used to determine the presence of a bottle and the presence of a bottle cap. A clear reading above the ambient clear reading (ie. when nothing is in front of the color sensor) indicates the presence of a bottle in front of the color sensor. A clear reading of around 30 indicates that either the top or the bottom of the bottle is in front of the color sensor.
2. When either the top or bottom of the bottle is in front of the color sensor, the ratio of the red reading to the blue reading is taken. Because Yop caps are red and Eska caps are blue, a red/blue reading ratio greater than 2 is indicative of a Yop cap being present, while a red/blue reading ration lower than 0.75 is indicative of an Eska cap being present.
3. If the red/blue ratio taken in point #2 is between the two mentioned thresholds, it is determined to be a neutral reading. This indicates that the part of the bottle being read either is the top without a cap, or the bottom of the bottle. This is because on bottles without caps, Yop bottles have a top and bottom which are predominantly white, while Eska bottles have a top and bottom which are predominantly transparent. In both of these cases, the color sensor reading will not be predominantly red or blue; rather, the red and blue readings for white and transparent surfaces are about equal, and the ratio between red and blue readings becomes roughly equal to 1.
4. If the value of either the red or the green readings passes 150 at any time during the scanning of a bottle, a flag for “yop_nocap” is set. This flag distinguishes between the bottle being a Yop or an Eska in the case in which both the top of the bottle and the bottom of the bottle has a neutral red/blue ratio. For Eska bottles, the red and green readings almost never pass 150 due to the transparent nature of the bottle reflecting little light back into the color sensor. In contrast the Yop’s opaque and colorful body will cause a red or green reading above 150. Note that the blue reading being above 150 is omitted from setting this flag, as Eska bottles sometimes have blue labels, which can falsely trigger this flag.
5. Finally, the bottle type is determined. If either the top or the bottom of the bottle is read as predominantly red (ie. red/blue ratio above upper threshold), then the bottle is determined to be a Yop bottle with cap. If either top or the bottom of the bottle is read as predominantly blue (ie. red/blue ratio below lower threshold), then the bottle is determined to be a Eska bottle with cap. In the case in which both the top and the bottom of the bottle is read as neutral (ie. red/blue ratio around 1), then the bottle is determined to be a Yop bottle without cap if the “yop_nocap” flag is set or an Eska bottle without cap otherwise.

Note that due to the open body of the machine, different ambient light levels can drastically affect the different thresholds needed to determine the type of bottle passing by the sensor. In particular, it would affect the ambient clear reading used to detect the presence of a bottle. Therefore, the algorithm's various threshold values need to be calibrated for each different light environment that the machine runs in.

5.3.4 Driving Motor Logic

The Centrifuge and Wheel motors have their state determined by a single transistor. To run or stop these motors, a signal is passed from the microcontroller through port RA2 to said transistor. Additionally, as a safety mechanism, Timer 0 on the microcontroller is set to trigger an interrupt that turns the motors off and ends the operation of the machine after the timer register overflows 3 times (around 30 seconds of operation).

The servo motors must be driven with a PWM signal with 20ms cycle to shift their position and sort the bottles. The duty can vary from 0% to 10% to indicate the position of the servo in a 180◦ arc.

As mentioned, our machine requires a PWM signal to be sent without delaying the main program to allow for continuous processing of bottles. Additionally, the PWM signal must be constantly maintained throughout the program to hold the servo in place, as bottles bumping into the sorting flap that the servo motors are attached to would shift the servo position otherwise.

The solution uses Timers 1 and 3 on the microcontroller to set a port to high or low at controlled intervals to generate a PWM signal. Each timer controls a separate servo, and they use an interrupt function to allow for the correct PWM signal timing to be generated independently of what the main program is doing. Additionally, because the timers are continuously running, the PWM signal is continuously set, and the servo position is constantly maintained.

To shift the position of the servos, we simply modify a variable that controls the values set in the timer registers and change the PWM signal's duty.

5.3.5 User Interface

The user interface is operated through a keypad and a HD44780 LCD. By pressing buttons on the keypad, the user is able to select between the different states in the microcontroller program. The keypad generates an interrupt to RB1 prompt the program to change states, and the four keypad pins connected to RB4:7 can be read to determine which button is being pressed.

The LCD provides feedback for the user. At each state, the LCD displays a certain message to indicate the state that the program is in, as well as relevant information pertaining to the state. The LCD is operated through ports RD2:7 on the microcontroller, and is operated using a set of provided instructions in an LCD program file.

Due to the limit of displaying characters on an LCD, it was determined that user operating instructions would be more easily expressed to a user through a printout next to the keypad. The operating instructions can be found in the following table:

Table 14: Operating Instructions for Machine

Keypad Button	Function
1	Start Operation
2	Display Current Operation Log
3	Display Current Operation Time

A	Date/Time Display
4	Display Last Operation Bottle Count
5	Display 2nd Last Operation Log
6	Display 3rd Last Operation Log
B	Display 4th Last Operation Log
7	Manual Operation Stop

Again, due to the limited space for display on the LCD screen, the functions for displaying operation logs have multiple pages for each log that can be scrolled through by repeatedly pressing the button used to access the logs.

5.3.6 Utility Functions

5.3.6.1 Date/Time Display

The date/time is kept track of on a real-time clock (RTC) on the PIC Devbugger board. The RTC communicates through the I2C protocol and is connected to the SCL and SDA pins of the microcontroller by the design of the PIC Devbugger board. The read and write I2C functions were used from the provided RTC communication file to set the time and read the time from the RTC. To display the date/time on an LCD screen, the data read from the RTC was first passed through a converter to convert the decimal unsigned character output from the RTC to a hexadecimal integer that is suitable for display on the LCD.

5.3.6.2 Operation Time

The operation time is determined by reading the RTC once at the beginning of the operation and once at the end of the operation, and taking the difference in time to be the total time taken for the operation

5.3.6.3 Operation Log Storage

Operation logs are stored in the EEPROM memory of the microcontroller for later retrieval. Each operation log is stored as 5 bytes of memory in 5 consecutive addresses in the EEPROM. These 5 bytes correspond to the total bottle count, the bottle count of Yops with caps, the bottle count of Yops without caps, the bottle count of Eskas with caps, and the bottle count of Eskas without caps. After each operation run, the previous logs stored in the EEPROM have their addresses shifted over by 5 to empty out 5 memory bytes, and the operation log data from the completed run is stored in those 5 emptied bytes. When needed, the operation logs are also accessed by reading the corresponding address in EEPROM memory.

5.4 Suggestions for Improvement of the Subsystem

5.4.1 Improved Bottle Detection Algorithm

The most important improvement to be made would be improving the bottle detection algorithm. Currently, the bottle detection algorithm has room to improve in accuracy (see Table 15).

Table 15: Bottle Detection Algorithm Accuracy

Bottle Type	Trials	Successful Detections	Accuracy (%)
Yop with Cap, Head First	30	30	100
Yop with Cap, Tail First	30	30	100
Yop without Cap, Head First	30	28	93
Yop without Cap, Tail First	30	30	100
Eska with Cap, Head First	30	19	63
Eska with Cap, Tail First	30	25	83

Eska without Cap, Head First	30	26	86
Eska without Cap, Tail First	30	27	90

Two major trends in bottle detection accuracy have been noted. Firstly, the detection algorithm performs worse if the bottle passes by the sensor head first. This is likely due to the orientation of the bottle as it falls down the transporting tube. When visually inspected, it appears that the end of the bottle that last passes the sensor will be closest to the sensor in physical orientation. This corresponds to more accurate readings for the end of the bottle that passes past the sensor last, which explains the increase in accuracy of the detection algorithm if the head of the bottle passes the sensor last, as that is where the algorithm needs to scan to determine the presence of a bottle cap.

The second trend noted in the detection accuracy is the inaccuracy of Eska detection. This is likely due to the transparency of Eska bottles resulting in poor color sensor readings. While the transparent material of the Eska often results in low, uniform RGB readings in the color sensor, it has been observed to occasionally result in exceptionally high red color readings on the color sensor. This often causes a misdetection of the Eska bottle as a Yop bottle. Additionally, due to the larger size of the Eska bottle compared to the Yop bottle, the distance from the bottle cap to the sensor is increased, and it appears to be significantly harder to detect the presence of the cap on an Eska bottle, resulting in many misdetections of Eska bottles with caps as those without caps. This is especially apparent when the Eska bottle travels past the sensor head first, as noted previously.

Finally, a few remarks about overall accuracy can be made. The test in Table ___ was preformed in a dim environment with extensive calibration of the algorithm to suit the light levels of the environment. Although test results were not recorded in a brighter environment, an accuracy drop was noted during demonstration of the bottle sorting machine in the University of Toronto laboratories, which have a significantly brighter ambient light level then the testing location. Additionally, it was noted that the readings for all bottles being detected were extremely variable. This is likely due to the fact that the transport tube that the bottles travel through while being scanned is filled with contours and bumps, resulting in the bottle traveling in a very inconsistent manner through the tube for each trial.

Therefore, possible ways to improve the sensor readings include decreasing the ambient light that reaches the sensor, possibly by sealing off the body of the machine from outside light, and removing the bumps and contours from the transport tube to allow the bottle to travel in a uniform path past the sensor across multiple trials. Additionally, the bottle detecting algorithm can further be improved by collecting more data and running better pattern recognition algorithms to find sets of RGB reading values that better identify bottles.

5.4.2 User Interface

Several improvements to the user interface can be made:

1. Incorporation of a help menu, or other function, to guide the user in operation of the machine and eliminate the need for a separate printout to explain machine operation.
2. Implementation of a new EEPROM accessing method to allow the user to access operation logs from all previous runs stored in memory, instead of most recent 4.
3. Implementation of a standby and operation animation on the LCD to show the user if the microcontroller has frozen during operation

6. Circuits

6.1 Circuits Overview

As the circuits provide an interface between the electromechanical and microcontroller subsystems, their functionality and reliability are crucial. Additionally, the circuits should be implemented in a method as to make the role of the microcontroller as simple as possible. The circuit subsystem must provide power from a standard 120V, AC wall outlet, and convert it so that the power may drive simple components necessary for the robot, as well as the PIC microcontroller. The functionality of the circuits components may be divided into several different sections:

Power management

- Converts power from one standard AC, 110V-60Hz, 3 pin wall outlet into usable power
- Provides appropriate voltages for each circuitry component, and ensures adequate power supply to drive the entire machine with all electrical components running simultaneously

Actuators

- Bridges signals from the microcontroller to the DC motor, and both servo motors.

Color Sensor

- Provides adequate power and voltage to the color sensor, as well as bridges the signal from the color sensor to the microcontroller.

Emergency Stop

- Needs to put the machine's mechanical components to a complete stand-still at the push of a button

6.2 Power Management

6.2.1 Power Management Problem Assessment:

To power the circuits, the power from a single wall outlet must be used. Firstly, the circuit components need to be run off DC power. Because the PIC board is driven using 12V power, it is necessary to step-down the voltage coming from the outlet. Additionally, since the color sensor and several other components of the circuits were chosen to be driven off a 5V supply, it is necessary to have some sort of regulator to supply both 12V and 5V from one singular supply.

6.2.2 Power Solution and Possible improvements

In order to convert the AC wall power to lower voltage DC power, it is possible to use a transformer along with a diode bridge and capacitors. However, it was decided that is safer and cost efficient to simply purchase a AC-DC wall adapter (see Figure 16) that can be directly connected to a wall outlet. The chosen adapter is a 5V, 3A switching power-supply which reliably and safely converts the AC power to DC power, and can be easily connected to wires on a protoboard via a barrel jack.

Figure 16: AC-DC Wall Adapter

The power supply was chosen to be 5V because all of the actuators were chosen to be driven off 5V power, so it would require less circuitry to step-down from 12V to drive the actuators than to step up from a 5V supply to drive the 12V supply of the PIC board. In order to step up the 5V supply to 12V, it was chosen to use a switching 5V to 12V switching step-up regulator (U3V12F5). This regulator was chosen because it can supply ample current to drive the PIC board, and is cheap. As an alternative, it would be possible to use a linear regulator to step up to 12V, or build your own voltage regulator circuit.

6.3 Actuators and Actuator Control

6.3.1 Actuator Problem Assessment

In order to drive the mechanical components of the robot, actuators need to be chosen, power needs to be supplied to each of the respective actuators, and necessary signals from the microcontroller need to be bridged to the actuators. In our scenario, 2 5V DC motors needs to powered and turned on and off at will by the microcontroller. Additionally, 2 servo motors need to be implemented at 5V, with signals coming from the microcontroller controlling their movement.

6.3.2 Actuator Solution and Possible Improvements

Because the power supply was chosen to a 5V switching adapter, the power supply to the actuators may be directly connected to the protoboard of the 5V power supply board, with capacitors connected to prevent voltage spikes. For the 5V DC motor, the choice of the motor itself was determined by the electromechanical member, taking into account different aspects. In order to drive the motor, it was decided that a transistor(TIP142) was to be used, driven by the signal of the microcontroller to be able to change the speed and whether the motor is on.

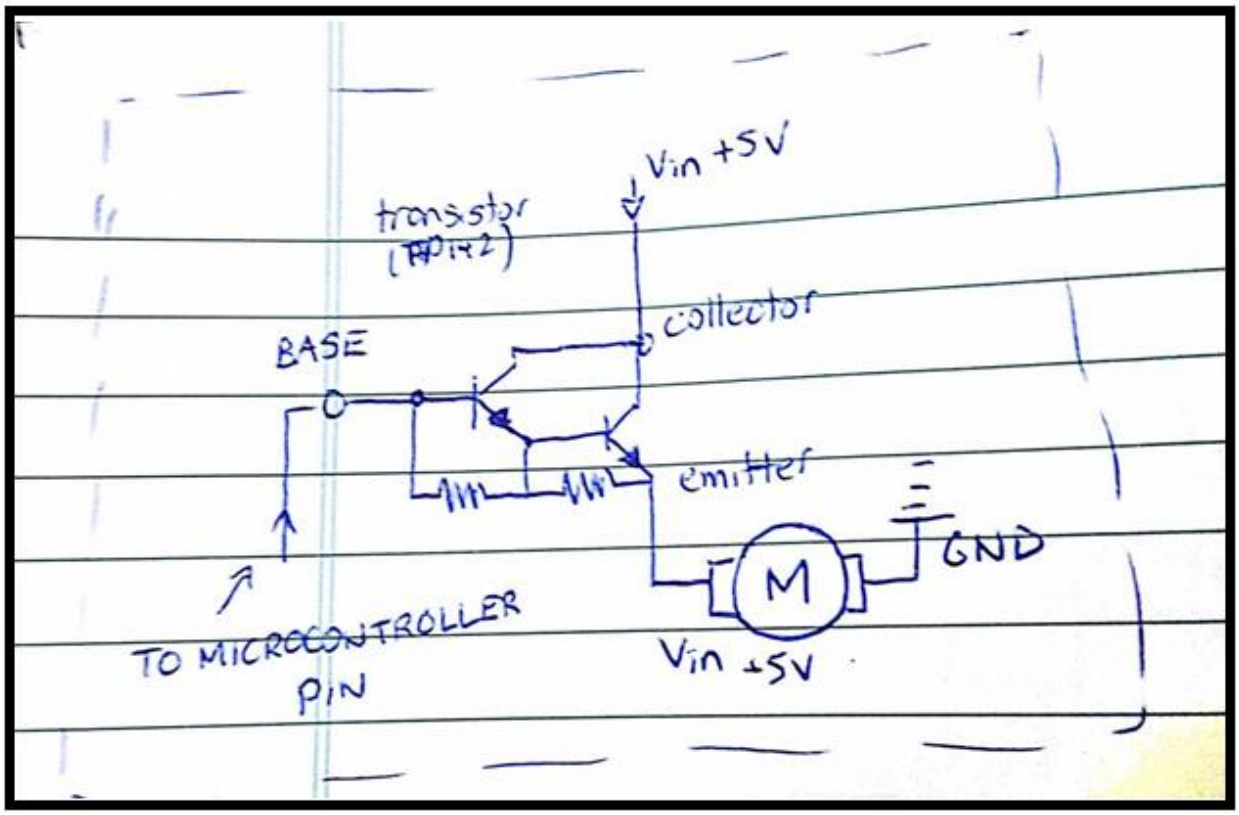


Figure 17: Circuit Diagram for Motor

The TIP142 Transistor (see Figure 17) was chosen because it is reliable, cheap, and was accessible to us in the project kit. Additionally, the base accepts a good voltage(~5v) and current(~40mA) that the microcontroller pin can provide. Capacitors need to be implemented with the DC motor in order to limit noise from voltage spikes.

The servo motors (see Figure 18) are simple 5V mini-servo motors(SG-90) with 3 connections: Gnd, Vin, and Signal. The Gnd and Vin pins were directly connected to the power supply with resistors, and the signal cable was simply connected to the microcontroller pins similar to the following simple schematic.

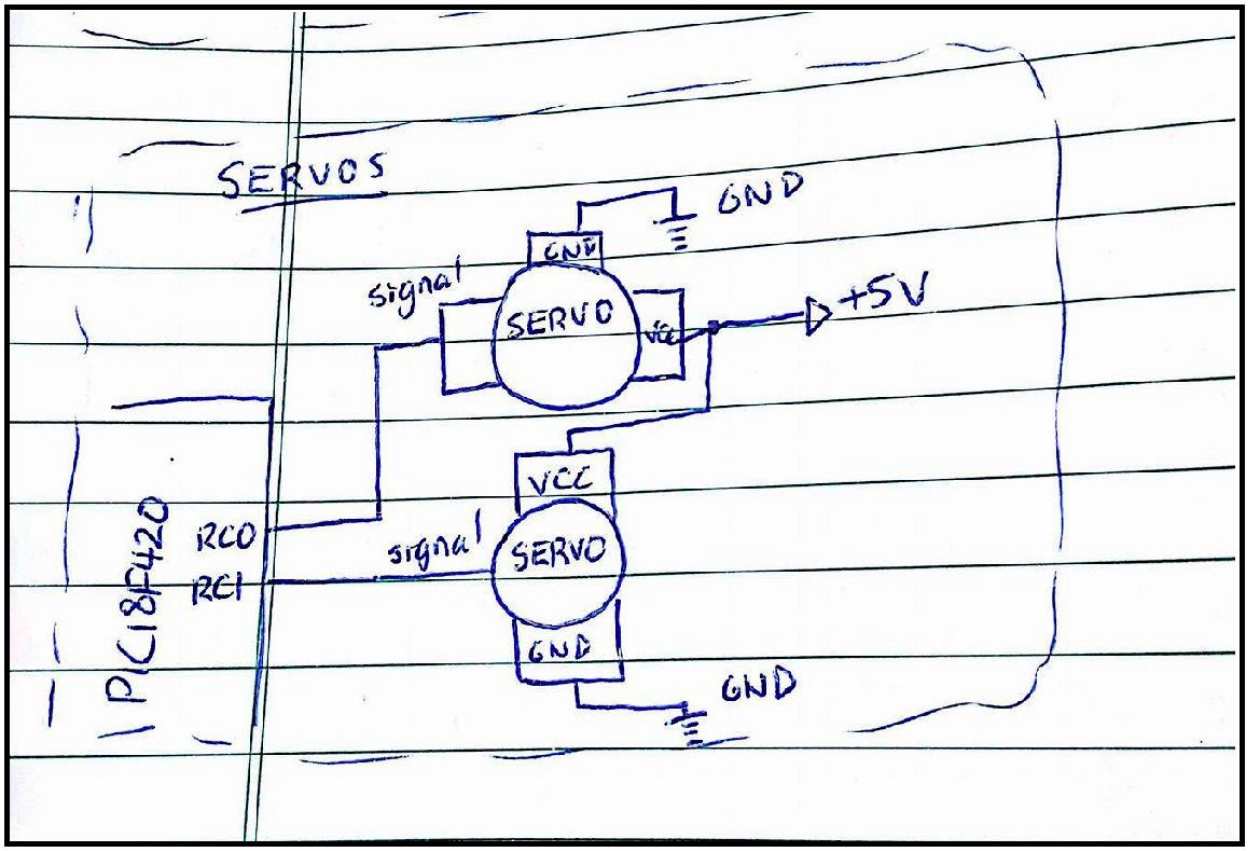


Figure 18: Circuit Diagram for Servo Motors

The servos chosen were chosen by the electromechanical member due to their mechanical specifications. Additionally, no additional circuit was necessary to drive the servos because the PWM necessary to drive the circuits is simple to implement given a microcontroller, so this was left to the microcontroller member.

6.4 Color Sensor

6.4.1 Color Sensor Problem Assessment:

A color sensor needs to be powered and connected to the microcontroller so that the microcontroller can detect the bottles flowing through the machine continuously, and quickly. Thus, the particular selection of color sensor emphasizes accuracy, and frequency of readings. Given that the color sensor is the sole sensor in our circuits, it is crucial that its functionality be perfect.

6.4.1 Color Sensor Solution and Possible Improvements

The TCS34725 (Adafruit) color sensor was chosen as the color sensor implemented in the project. This color sensor is driven off the default 5V supply, and the SDA/SCL pins are bridged to the I2C of the microcontroller. This color sensor was chosen because it is the most accurate color sensor within a reasonable allocated cost range (<$30), and was small enough to fit in the constraints of the machine. Through testing (discussed more thoroughly in integration and microcontroller), it was determined that the readings given were more than sufficient to distinguish between the bottles. One of the best features of the TCS34725 is that it has a 4th reading(clear) different from the normal 3 RGB, which was

beneficial in distinguishing the bottle between Yop and Eska given than one of the bottles is transparent and one isn't.

Alternatives may be to implement some sort of Infrared color sensor, LED/transistor setup, or some other method which may read the differences in the bottle. Regardless,

6.5 Emergency Stop

6.5.1 Emergency Stop Problem Assessment:

The machine needs to completely stop at the press of a button. Due to the nature of our actuators, only the DC motors are continuously moving, and thus only the DC motors needs to be stopped at the push of a button.

6.5.2 Emergency Stop Solution and Possible Improvements

As noted previously in the actuator section, the DC motor was connected to a TIP142 transistor which controls the voltage to the DC motor. If the microcontroller doesn't continuously send a signal to the transistor, the DC motor will not run. Additionally, if the microcontroller doesn't have any power, or the DC motor itself doesn't have any power supply, it won't run. In order to stop the DC motor, simple code was implemented by the microcontroller member to turn off the signal to the transistor driving the DC motor to turn it off at the push of an button. Additionally, in the case of malfunctions, a switch is implemented with the main AC-DC power supply of the circuits in order to completely cut off power to all circuitry in the event of an emergency stop (see Figure 19):

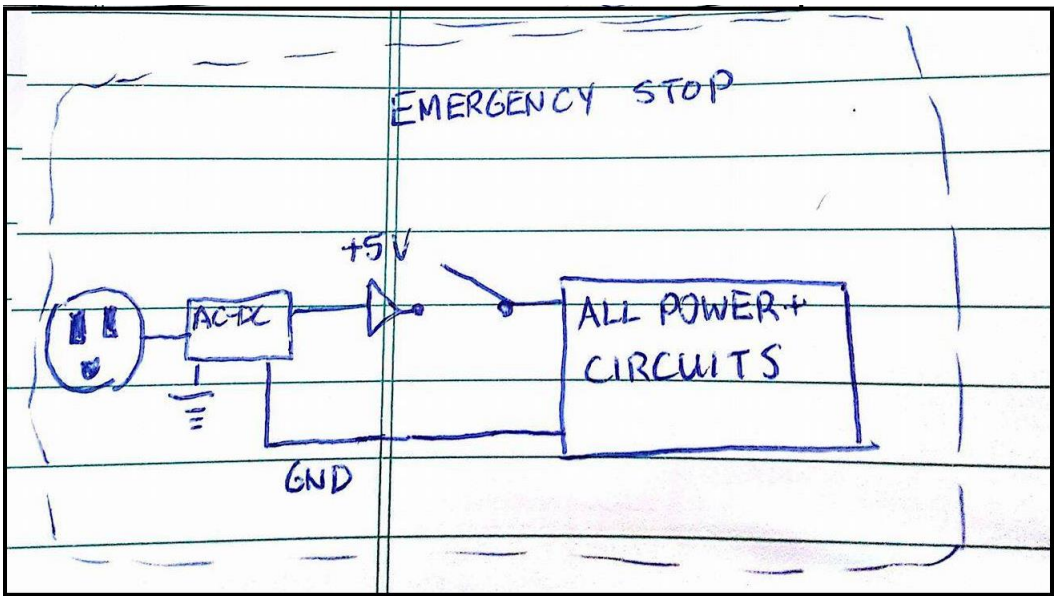


Figure 19: Circuit Diagram for Emergency Stop

This switch is a simple rocker switch which was chosen due to its cheap cost and accessibility. This switch can handle up to 6A of current, which is double the amount from the supply.

6.6 Circuits: Full Schematic Layout with PIC pins

The below circuit diagram (Figure 20) maps out the entire schematic for the PIC and peripherials.

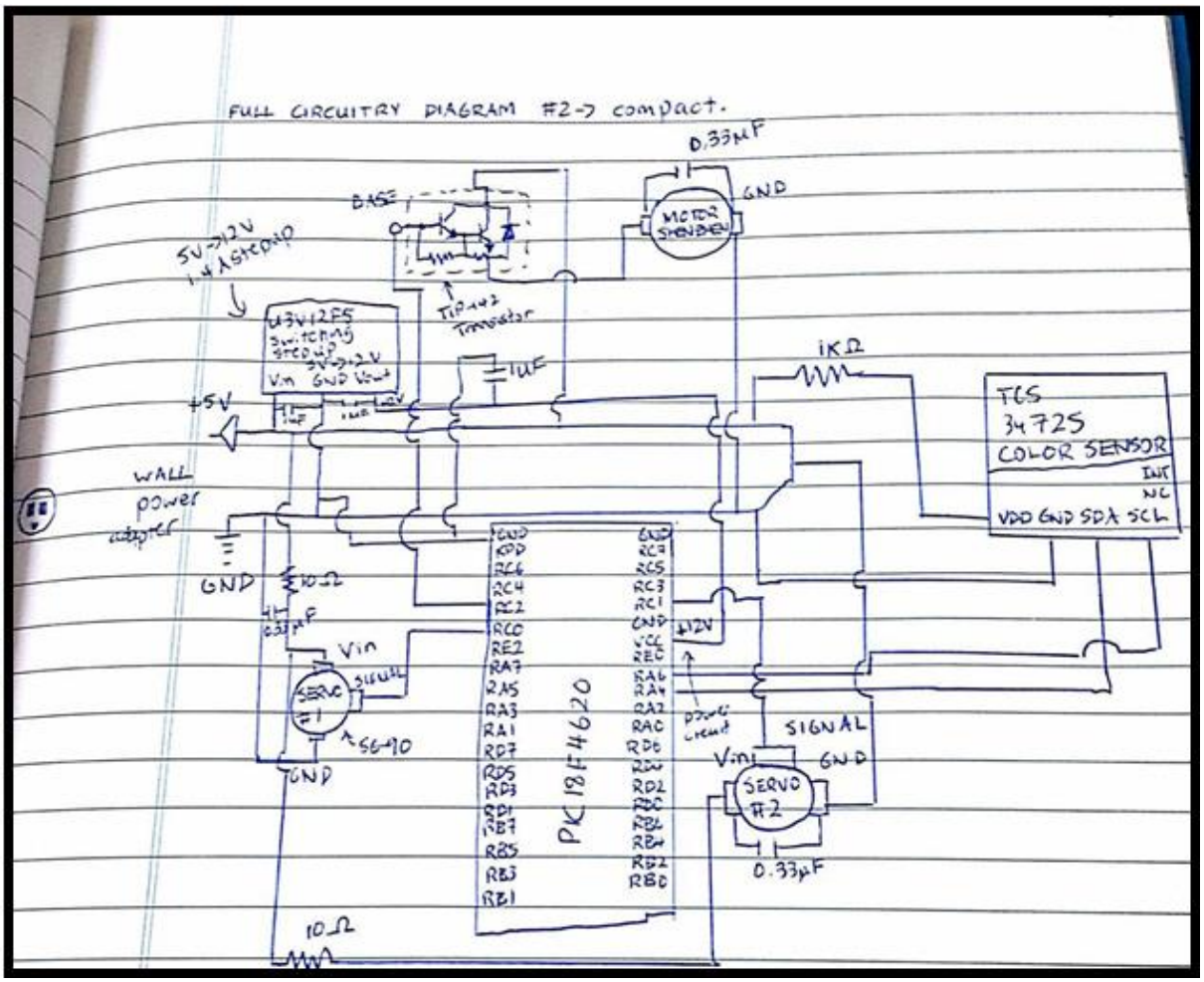


Figure 20: Complete Circuit Diagrams

6.7 Supporting Calculations:

6.7.1 Power Table

The below table (Table 16) lists out the power requirements for the machine.

Table 16: Power Consumption Table

Circuitry Component Type	**Model**	**Amount**	**Current Draw(mA)(max)**	**Voltage (V)**	**Total Power Consumption(W)**
Color Sensor	TCS34725	*1*	*0.235*	*5*	*0.00175*
Shenzhen DC Motor	*Project Kit*	*1*	*800*	*5*	*4*
Servo Motors	SG-90 Mini servo	*2*	*(200*2)=400*	*5*	*2*
PIC board	PIC18F4620	*1*	*1.25*	12	0.0154

Yellow DC motor	*1:48 Gearmotor Virtualbotix*	*1*	*250*	*5*	*1.25*
Totals			*1451.435*	*n/a*	*7.26729*

6.7.2 Efficiency Calculation:

Assume a very low efficiency of 50%, well below the specified minimum nominal efficiency [4] of motors regularly available for commercial purchase. Even with 50% efficiency:

$$Power_{consumed} = \frac{Power_{required}}{Efficiency} = \frac{7.28W}{50\%} = 14.56W.$$

$$Power_{from\ supply} = Voltage_{supply} Current_{Supply} = 5V * 3A = 15W$$

Clearly, we will have enough power from our AC-DC adapter even with a very low efficiency of the power consuming actuators.

6.7.3 Resistor Calculations:

In order to limit the amount of current running to each part of the system, resistors were added to each component of the system. The value of the resistor was especially important for the color sensor, where the intensity of the color sensor light was dependent on the amount of current.

Table 17: Current Calculations Table

Component	Max Current (mA) (I)	Voltage(V)	Resistor Value needed (V/I) (Ω)	Resistor Chosen (Ω)
Color Sensor	330	5	15.15	20
Servo Motor	1	5	5	4.7
Shenzhen DC motor	n/a	5	n/a	n/a
TIP142 Transistor	15	5	0.33	n/a
U3V12F5 Step-up Regulator	3	5	1.66	0
PIC board	400	12	30	30
Yellow Gearhead DC motor	n/a	5	n/a	n/a

The n/a values indicated are due to the high internal resistance of motors, definitely out of the range of our 3A power supply. Thus, no resistor is required. Additionally, no resistor is required for the transistor or regulator because their max current exceeds the max current provided by the power supple (3A).

6.8 Suggestions for Improvement to Circuits subsystem

6.8.1 Increasing Amount of useful Circuitry, Including Sensors

The biggest flaw of the circuitry subsystem is the choice to only use one major sensor in the detection of the bottle. While this was a team decision, it was mainly decided that the mechanism of detection was to be continuous, and that the color sensor read the bottle as it passes by. With this decision made, the color sensor would have to take data quickly, and a complicated algorithm would have to implemented to process the data that the color sensor reads, as the sensor will read a profile of the data of the bottle,

rather than a fixed location of the bottle. This variance is a software issue, but could be remedied by increasing the amount of circuitry in the system, for example implementing an ultrasonic or IR tripwire sensor, which could detect the position of the bottle (cap-first or end-first), as it is travelling down the pipe. The original proposal encompassed such a design.

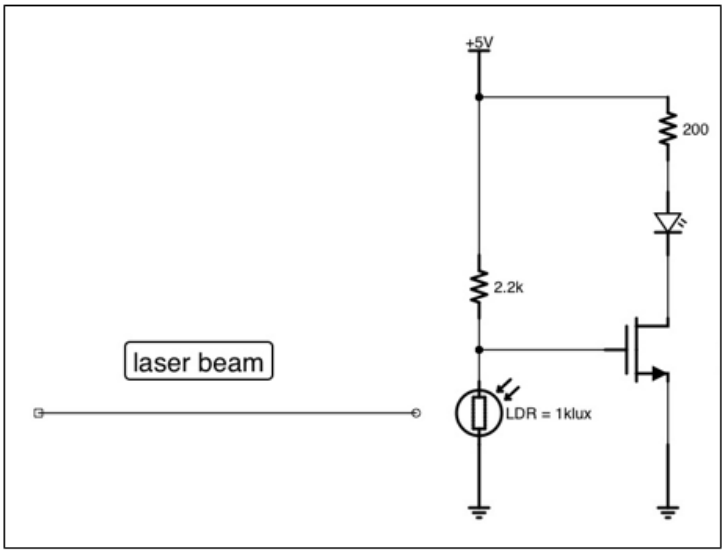


Figure 21: Laser Tripwire Example Circuit Diagram

For example, the above laser tripwire (Figure 21) would help significantly in reducing the amount of code needed when the bottle passes by the color sensor, as the tripwire would be able to trigger an interrupt based on the position of the bottle instead of continuously having the color sensor read data.

6.8.2 Optimizing power usage

While it is most important to ensure that the robot can function at full power consumption of all the circuitry, a lot of power is being wasted, and the machine is not efficient. The power supply provides 3A to the machine, and at maximum load the machine runs at around 2A, given that only one servo needs to turn at a time. Additionally, the regulator that is being used to power the PIC board supplies 1.4 A to an integrated power circuit on the PIC board, which then regulates the power down to a 1.25A max supply. This current is being wasted running to the PIC board, especially since the PIC needs to draw very few power from the board, which it doesn't even come close to consuming, considering that no power is being drawn from the Vcc pin of the PIC board. This could be optimized by changing the on-board power circuit, and using a different regulator for powering the PIC board. We see that power could have been more efficiently used through the regulator by using the onboard power to power simple circuitry, such as the color sensor.

Additionally, our power supply provides 5V and 3A, providing a total of 15W to the machine. While in our supporting calculations, we needed 14.56W to power all the electrical components, this was accounting for an extremely low efficiency (50%), and thus we would have been able to

6.8.3 Optimize the use of pins

While there were sufficient pins on the PIC board to run the amount of circuitry, there were several cases where the microcontroller signals weren't being used effectively. Firstly, the servo motors are run

separately off two pins with two different PWM signals, but for the sake of the design only one signal would be necessary to control the servos to turn to the correct position in all cases. In such a design either removing a servo or using only one pin would be enough. Additionally, there are only 2 color sensor I2C ports, allowing for use of one color sensor within the machine. If more were necessary, it would be possible to optimize the design by using a multiplexer to increase the amount of color sensors on the machine.

6.8.4 Managing Voltage in System

Although the decision to use a 5V motor was done collectively as a team and the motor was chosen by the electromechanical member, the PIC must be run off a 12V power supply. This limits our power supply in general to either obtain an adapter that is 12V, or another voltage that is then regulated to 12V. While our circuitry used a step-up regulator to change 5V to 12V, it would have been beneficial to run everything in the machine off a consistent voltage, instead of having to regulate voltages, as this is prone to voltage spikes and issues with overheating. Additionally, running everything off 12V would have allowed for more power to be provided to actuators, which was an issue that was found when using the DC motor running the centrifuge.

6.8.5 Using a Printed Circuit Board

Rather than using a Normal protoboard and soldering connections with wires, it would have been beneficial to make all the circuitry connections using a printed circuit board. This would have greatly minimized the amount of wires necessary between connections on boards, and would have reduced the amount of interference from wires significantly less, as well as improving the aesthetic of the machine.

7. Electromechanical System

7.1 Overview

The main objective for the electromechanical system is to construct a sturdy outer frame and every other mechanical mechanism that is involved in the design. The outer frame must be smaller than our constraint, which is a cube of 55cm x 55cm x 55cm. As such, every detail of every mechanism must be carefully manufacture to optimize the entire operation. Furthermore, all the actuators involved must be calculated to ensure a streamline operation. During the building and debugging process, many minor parts have been implemented as solutions to alleviate the jamming issues encountered.

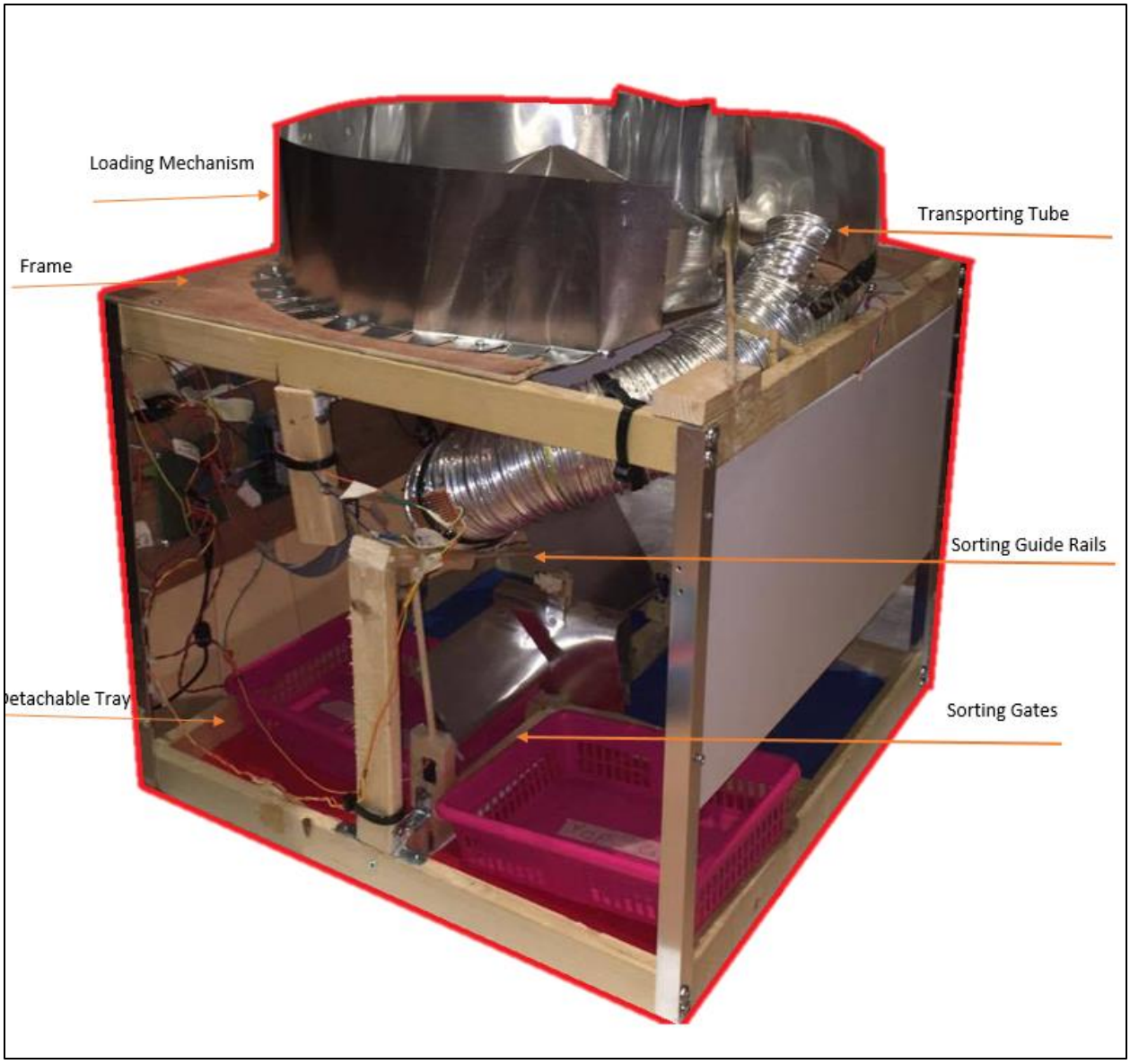


Figure 22: Overall Schematics of Electromechanical Systems of the Machine

7.2 Problem Assessment

The electromechanical subsystem is divided into several sections (Please refer to Figure 22 for different mechanisms):

1. *Frame*
    - Provide structural support for the overall machine
2. Loading Mechanism
    - Enable user to load the bottles randomly in a selected area and align the bottles in a length wise orientation
3. *Transporting Tube*
    - Bridge between the loading mechanism and the sorting guide rails and provide a location for the colour sensor to sense the bottles

4. *Sorting Guide Rails*
    - A mechanical mechanism to differentiate between the Eska and Yop bottle based on their physical dimension (radius of the bottle)
5. *Sorting Gates*
    - Metal Flaps driven by servo motors that guide the falling bottles into its corresponding side after receiving signal from the microcontroller
6. *Detachable Tray*
    - A lightweight basket that is easily accessible and easily removable with the container being able to hold a maximum of 10 bottles.
7. *Mounting*
    - A place that houses all the main circuitry components as well as the PIC board

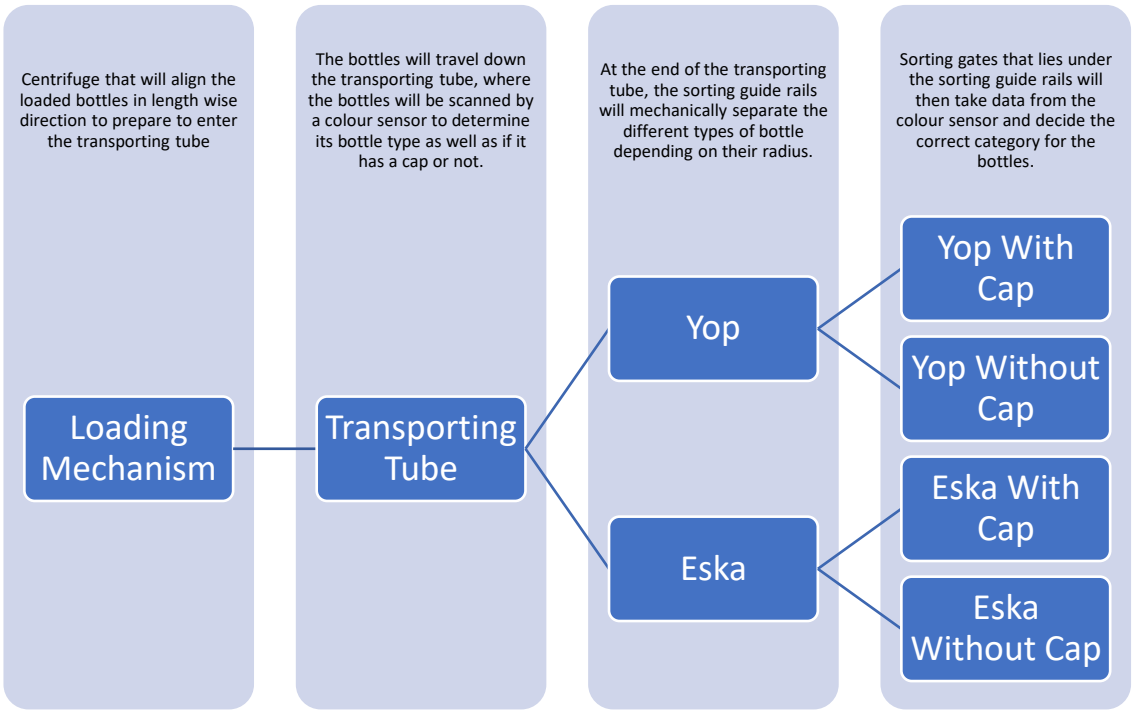


Figure 23: Flowchart of Bottle Travelling through Machine

The above Figure 23 shows a basic flow of a bottle traversing through the machine

7.2.1 Frame

The frame of the machine is the main structure and thus must support the weight of every element involved in the machine. Furthermore, the frame should also provide a base support for detachable tray such that the tray will move along with the frame as well as a top support for the loading mechanism. The overall frame will be under the size constraint of 0.55cm x 0.55cm x 0.55 cm as well as the weight constraint of 5.0 kg.

7.2.2 Loading Mechanism

The loading mechanism should be compact and aid the feeding process of the transporting tube. Not only does the loading mechanism should provide an area of which the bottles can be unloaded onto, but it also must orient the scrambled bottles into a uniform direction. As such, a continuous feeding

mechanism must be implemented to transport the bottles from the initial loading point to the transporting tube. Furthermore, some sort of actuation need to be implemented to provide a constant movement of bottles in the machine.

7.2.3 Transporting Tube

The transporting tube will have to passively move the bottles from the loading mechanism down to the sorting guide rails. It needs to provide a location for the colour sensor to sense the moving bottles at an acceptable accuracy such that the sensor can distinctly determine the type of bottle it is, meaning that the sensing has to be done in an enclosed and dark area to maximize the accuracy. Furthermore, the transporting tube must be within the outer frame such that it curves downward and redirects the bottles 180°, facing the opposite way, while still leaving room for the sorting guide rails and the sorting gates. Under this size constraint, it will be a challenge to obtain the ideal curvature.

7.2.4 Sorting Guide Rails

The sorting guide rails has to be able to mechanically separate the bottles based on their type, Yop or Eska. They have to be parallel to each other and maintain their orientation while the bottles are traversing. Due to the size constraint, these rails have to maintain a balance between maximizing the angle to allow bottles to fall through and minimizing the overall displacement in height to stay within the size constraint. Further, it will be a problem maintaining the guide rails apart at the very edge, as every movement on the rails will minorly displace the rails.

7.2.5 Sorting Gates

The sorting gates have to take in signal from the microcontroller and switch orientation based on the readings from the colour sensor to guide the bottles into their respective bins. The main concern in this mechanism is to create a flap that is optimized in terms of time needed to travel certain degrees to altered the path of the bottles. Furthermore, a middle blocking mechanism have to be implemented to ensure that no Yop bottles can flow through into the Eska bins when the bottles are traveling down the sorting guide rails.

7.2.6 Detachable Tray

The detachable trays will have to be able to hold all 10 bottles in the scenario of edge case, where all the bottles are of a single type. The trays have to also be very easily removable, meaning that they have to be light-weight and compact.

7.2.7 Mounting

The mounting of the PIC board and all the circuitry components have to be securely fastened in place. Furthermore, the circuitry components should be in the vicinity of the PIC board to reduce extra lengthy wires.

7.3 Solutions and Supporting Calculations

This section highlights the design features that addresses the problem stated in problem assessment as well as the material used and any applicable calculations.

7.3.1 Frame

Due to the size and weight constraint, the top and bottom square of the frame will be made out of 2 inch by 1 inch wood planks, supported by vertical L-shaped aluminum angle bar. After construction, the dimensions of the frame turn out to be 55.2cm in width, 55.3cm in length and 45cm in height, as shown in Figure 7.3. The weight of the machine is measured to be 4.9 kg using a balance scale. The top side of

the frame is covered by a quarter inch thick square ply wood board that is 55cm in width. The bottom side of the frame is divided into 4 different quadrants, each corresponding to a bin with its respective bottle type. The quadrants are made in 25.5cm wide squares. Furthermore, the quadrants are enclosed with colour coded polystyrene board. The reason behind choosing the 2' by 1' wood is due to its relatively cheap price point to fit in with our budget as well as its shear strength, tensile strength and compressive strength. The implementation of the L-shaped aluminum angle bar is to ease the attaching, the aluminum piece and the wooden piece can be easily secured together with screws and washers. Polystyrene board is a very light material that can support sufficient weight. In our frame, because the bottom of the frame only need to support the weight of the bins and bottles, the strength of the material can be overlooked, as the stress that the bins and bottles exert on the bottle plates are almost negligible.

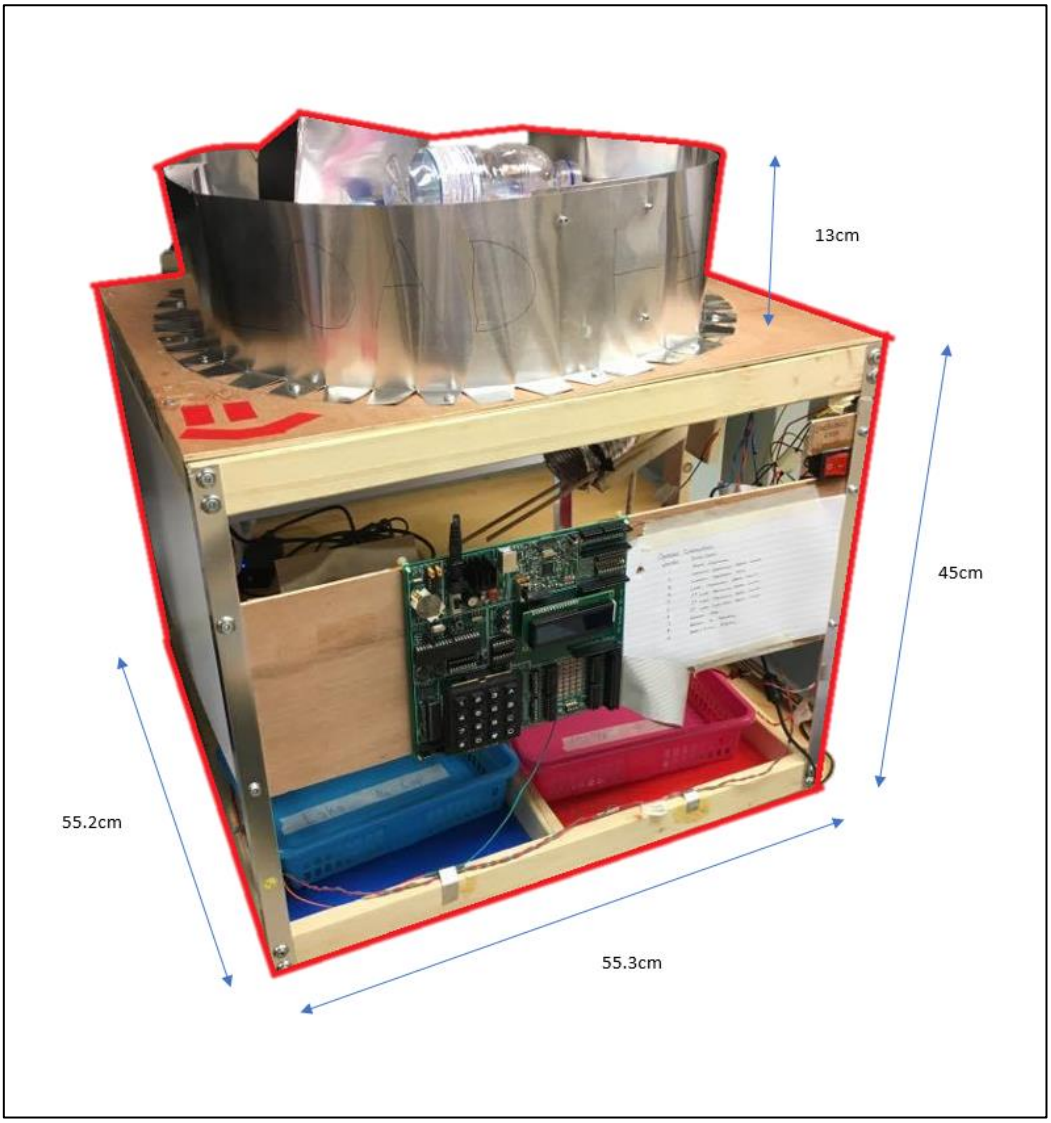


Figure 24: Overall Dimensions of the Outer Frame

Weight Calculations

Top (Loading Mechanism: Centrifuge + resting surface)

$$W_{Resting\ Surface} = (0.55m)^2 \times 0.00635m \times 500\frac{kg}{m^3} = 0.96kg$$

$$W_{Cylinder} = \pi \times 0.1m \times 0.5m \times 0.00205m \times 2700\frac{kg}{m^3} = 0.87kg$$

$$W_{cone} = \pi \times 0.05m \times (0.25m^2 - (0.25m - 0.00205)^2) \times 2700\frac{kg}{m^3} = 0.43kg$$

$$W_{Centrifuge\ bottom} = \pi \times (0.2m)^2 \times 0.00635m \times 500\frac{kg}{m^3} = 0.40kg$$

Frame

$$W_{wood} = 0.0254m \times 0.0508m \times 2.032m \times 500\frac{kg}{m^3} = 1.328\ kg$$

$$W_{aluminum} = 0.04m \times 0.00205m \times 0.45m \times 2700\frac{kg}{m^3} \times 4 = 0.40\ kg$$

Bins

$$W_{Bins} = 0.078\ kg \times 4 = 0.312kg$$

Transporting Tube

$$W_{railings} = \pi \times (\frac{0.003175m}{2})^2 \times 3.65m \times 2700\frac{kg}{m^3} \times 4 = 0.50kg$$

Sorting Gates

$$W_{sorting} = (0.16m \times 0.002052m \times 0.08m \times 2 + 0.16m \times 0.002052m \times 0.13m \times 2) \times 2700\frac{kg}{m^3}$$
$$= 0.37kg$$

Housing for circuitry and Controller

$$W_{housing} = (0.15m)^2 \times 0.002052m \times 3 \times 2700\frac{kg}{m^3} = 0.373kg$$

Total Weight of Robot

$$= 0.96kg + 0.87kg + 0.43kg + 0.40kg + 1.328kg + 0.40kg + 0.312kg + 0.312kg + 0.37kg + 0.373kg = 4.943kg$$

7.3.2 Loading Mechanism

In order to achieve its purpose, the loading mechanism has been divided into its outer aluminum wall, centrifugal cone, and spinning wheel mechanism All of which aims to transport the bottles uniformly into the transporting tube without jamming. Refer to Figure 25 for specified parts.

Figure 25: Loading Mechanism

7.3.2.1 Aluminum Wall

The aluminum outer wall that encloses the entire top loading mechansim has a height of 13 cm and a radius of 23 cm. It is made out of 22 Ga Aluminum that has a thickness of 0.065cm. The wall is built to ensure that during operation, the bottles are contined within the centrifuge and not being displaced outside. At one end, the wall leads to the beginning of the transporting tube; at the other end, the wall leads to a elevating flap with extending height piece to guide the bottle to traverse through the wall once again if it is unable to travel into the transporting tube, as shown in Figure 26. On the side of the wall, there is a incremented V shaped vertical ramp. The purpose of this is to allow the bottles to exit the centrifuge at an angle that will lead to the spinning wheel.

Figure 26: Top of Machine

7.3.2.2 Centrifugal Cone

The centrifual cone is the main component of the entire loading mechanism. It consist of an aluminum cone that is 13 cm in height and 20 cm in radius. It is accompanied by a wooden circular disk that is 25 cm in radis. The entire cone is driven by a 12V 50RPM DC motros that is attached to the wooden disc through a 4mm to 5mm shaft coupler that locks between the motor shaft and a threaded rod. The threaded rod is then secured on both side of the wooden plate using 2 nuts. The motor is mounted on to a L-shaped mount onto a piece of wood. To further reduce the friction force and maximize the power output of the motor, a 4" lazy susan is attached to the bottom of the wooden plate such that the ball bearings on the lazy susan will activate and reduce any possible friction as well as maximizing the power output. Detailed schematics of the centrifuge and motor mounting can be refer to Figures 27 and 28.

Figure 27: Centrifuge Motor

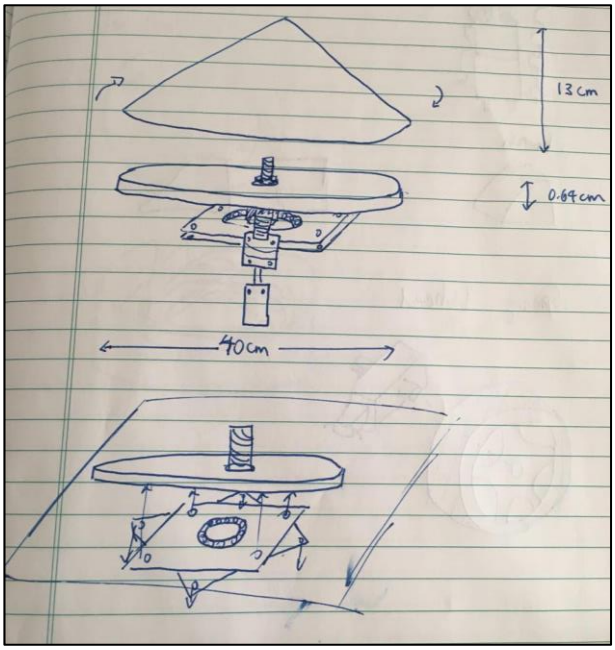


Figure 28: Centrifuge Setup Sketch

Furthermore, on the body of the cone, multiple notches, made out of wood and covered in epoxy and aluminum tape, is present to further agitate the cluster of bottles if there exists any jamming at all, as shown in Figure 29. The wooden ramped slope shown in Figure 30 is implemented specially to remove any jamming occurs at the elevated flap edge of the aluminum wall.

Figure 29: Agitating Piece 1

Figure 30: Agitating Piece 2

7.3.2.3 Spinning Wheel Mechanism

The Spinning wheel inside the loading mechanism acts as a secondary actuation device that aids the movement of bottles along the centrifuge down to the transporting tube. The spinning wheel consist of a yellow DC motor as well as a wheel that fits directly onto the shaft of the motor. The actual motor is then mounted by By portruding the wheel only 0.4 cm above the surface of the top, the wheel does not interfere with the conventional movement of the bottles. Instead, it serves as an additional booster that guides the bottles and gives the bottles additiional push in case of any jamming. The yellow motor is mounted on by screwing the motor into copper pipe strap that is then fasntened on to a piece of wood using screws. Because of the motor is designed to drive the wheel, it is determined that no extra calculations is required. Furthermore, since the wheel is merely gliding pass the bottles, it does not require any significant amount of additional torque. Please refer to Figure 31 for the schematics of the design and Figure 32 for the acutal implementation of the wheel design.

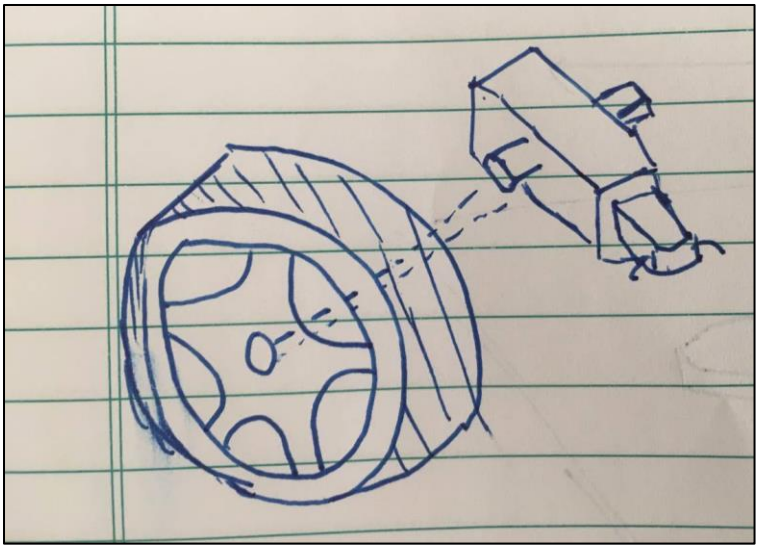


Figure 31: Wheel Motor

Figure 32: Actual Implementation of Wheel Design

Moment of Inertia and Torque Calculations

$$Mass\ of\ Centrifuge + all\ the\ possible\ Load = 0.43kg + 0.015kg \times 10\ bottles = 0.58kg$$

$$F_{centripetal} = F_f$$

$$m\frac{v^2}{r} = \mu mg$$

$$v = \sqrt{r\mu g} = \sqrt{0{,}20m \times 0.4 \times 9.8\frac{m}{s^2}} = 0.88m/s$$

E $\omega = \frac{v}{r} = 4.4\ rad/s$

For the centrifuge moment of inertia,

$$I = \int\int \rho^2 dA = \int\int \rho^2 r d\theta dz = 4.3 \times 10^{-3} kgm^2$$

For Bottles,

$$I = mr^2 = 0.15kg \times (0.2m)^2 = 6 \times 10^{-3} kgm^2$$

$$\sum I = 1.3 \times 10^{-2} kg\ m^2$$

Angular Momentum,

Assuming achieving this angular momentum in 3 seconds,

$$\tau = \frac{dL}{dt} = \frac{0.0572kgm^2/s}{3s} = \frac{0.019kgm^2}{s^2} = 0.186N \cdot m = 1.89kg \cdot cm$$

Thus, the DC motor we choose must have a torque greater than $1.89kg \cdot cm$

As such, the ShenZhen 12V 50 RPM DC motor have a max efficient torque of $269mN \cdot m$.

Through our calculations, the torque required is only $186mN \cdot m$. Thus, the motor can run without problem.

7.3.3 Transporting Tube

The final iteration of the transporting tube consists of aluminum foil duct with a radius of 5cm. This type of tubing is chosen because of its malleability and its weight. Because of the size and weight constraint, it is very difficult to manufacture a precise ramp with aluminum or wood. Because of its ability to extend and twist, the foil duct is perfect for this purpose. The duct is first extended and then twisted in a U-shape to guide the bottles from the centrifuge down to the sorting gates. After the duct is fixed in shape, it is first zip tied to maintain the shape and then it is hardened using epoxy such that the shape of the tube is fixed and that it can no longer be adjusted.

In the middle section of the tube, there is an opening just big enough to house the colour sensor, of which is used to scan the incoming bottles. In addition to its other perks, this aluminum foil duct is enclosed, meaning that the inside of the tube is dark. As such, the colour sensor is then able to function at its ideal configuration, disregarding any potential light source that may affect the readings. In

addition, as the bottles are traveling down the tube, due to the curved nature, bottles will automatically space apart in this continuous operation. That separation allows the colour sensor to pick up distinct readings that can separate from bottles to bottles. Refer to Figure 33 for the location of colour sensor.

On the inner side of the aluminum foil duct, aluminum tape is applied throughout the bottom and the side of the tube to smoothen the bumpy surface of the duct. This way, not only is the bottle traveling with less friction, but also provides the bottle a flatter surface to travel on, of which the colour sensor can more accurately pick up readings. Also it reduces the overall noise level during operation, as there will be no sound of the bottles gliding past the aluminum ridges. Refer to Figure 34 for top view.

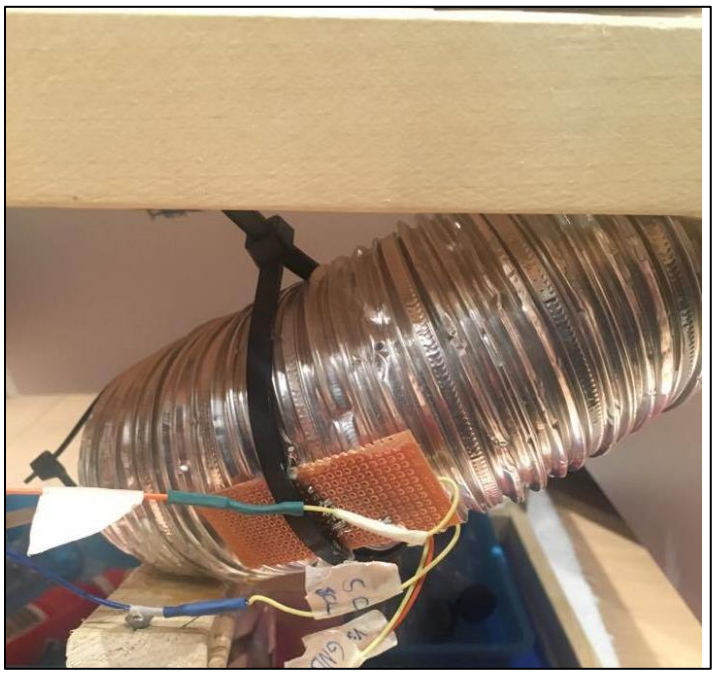


Figure 33: Side View of Transporting Tube

Figure 34: Top View of Transporting Tube

Ideal Angle of Decline Calculations

$$F_{net} = F_g - F_f = ma$$

Since we want a constant velocity downwards, along the guiding rail, therefore the acceleration must be 0.

$$F_g = F_f$$

$$mgsin\theta = \mu mgcos\theta$$

Since the coefficient of friction between the rail and the bottle is 0.15 and that is only 1 point of contact between the bottle and the tube, therefore the equation becomes

$$sin\theta = 0.15cos\theta$$

$$\theta = 8.3°$$

Thus, the angle of decline must be greater than $8.3°$ for the bottles to slide down with an initial velocity. Ideally, the angle is preferred to be as large as possible to ensure that the bumpiness will not affect the bottle's motion but $8.3°$ is the absolute minimum to allow the bottles flow.

7.3.4 Sorting Guide Rails

The sorting guide rails is made out of 2 parallel ¼" wooden dowel with a length of 15 cm. To ensure that they maintain their parallel position, epoxy had been applied to hold the dowel in place. Due to the effect of gravity and friction, we have determined that having the rails slant a little bit under the ideal decline is viable at around $20°$. This is because the smoothened-out aluminum duct is allowing the bottles to travel at a speed fast enough that Yop have the possibility of over shooting, and land in the

Eska regions with its momentum. As such, having a gentler decline slows down the entire process and allow the servo motors that are powering the sorting flaps to adjust into correct position.

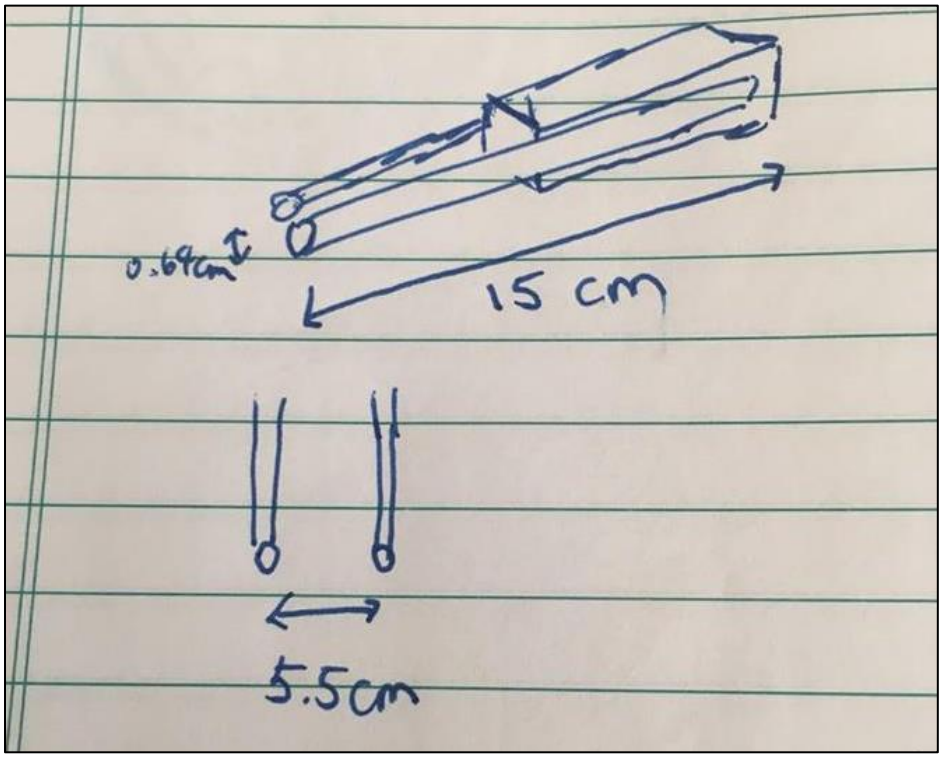


Figure 35: Sorting Flaps Sketch

Ideal Angle of Decline Calculations

$$F_{net} = F_g - F_f = ma$$

Since we want a constant velocity downwards, along the sorting guide rails, therefore the acceleration must be 0.

$$F_g = F_f$$

$$mgsin\theta = \mu mgcos\theta$$

Since the coefficient of friction between the rail and the bottle is 0.2 and that are 2 points of contact between the bottle and the tube, therefore the equation becomes

$$sin\theta = 0.4cos\theta$$

$$\theta = 21.8°$$

Thus, the angle of decline must be greater than 21.8° for the bottles to slide down with an initial velocity. Ideally, the angle is preferred to be as large as possible. However, because the bottles have an initial velocity, a smaller angle at this point will suffice as the sorting rail does not need to be very long and thus angles less than $21.8°$ is acceptable but not ideal.

7.3.5 Sorting Gates

The sorting gates have to take in signal from the microcontroller and switch orientation based on the readings from the colour sensor to guide the bottles into their respective bins. The main concern in this mechanism is to create a flap that is optimized in terms of time needed to travel certain degrees to altered the path of the bottles. Furthermore, a middle blocking mechanism have to be implemented to ensure that no Yop bottles can flow through into the Eska bins when the bottles are traveling down the sorting guide rails.

The sorting gates are made out of 3 components, the SG90 servo motor, the flap, and the wooden dowel attached to the motor shaft. The motor is mounted on a piece of wood using screws and then the wooden dowel is attached to the servo by cutting out the shape of the shaft and then epoxying the tight fitted shape to the shaft. This attachment allows the servo to quickly turn with no visible delay time. Then, the aluminum flaps are epoxied on to the wooden dowel and then folded and attached using rivets. This entire process ensures that the flaps can withstand the hit of a bottle falling.

Additionally, to prevent the momentum of the Yop bottle traversing into the Eska bins, a middle divider mechanism is implemented to block off any potential free falling Yops. The pieces have been epoxied on to ensure that they maintain the upright position after being hit with a bottle.

Figure 36: Actual Sorting Mechanism

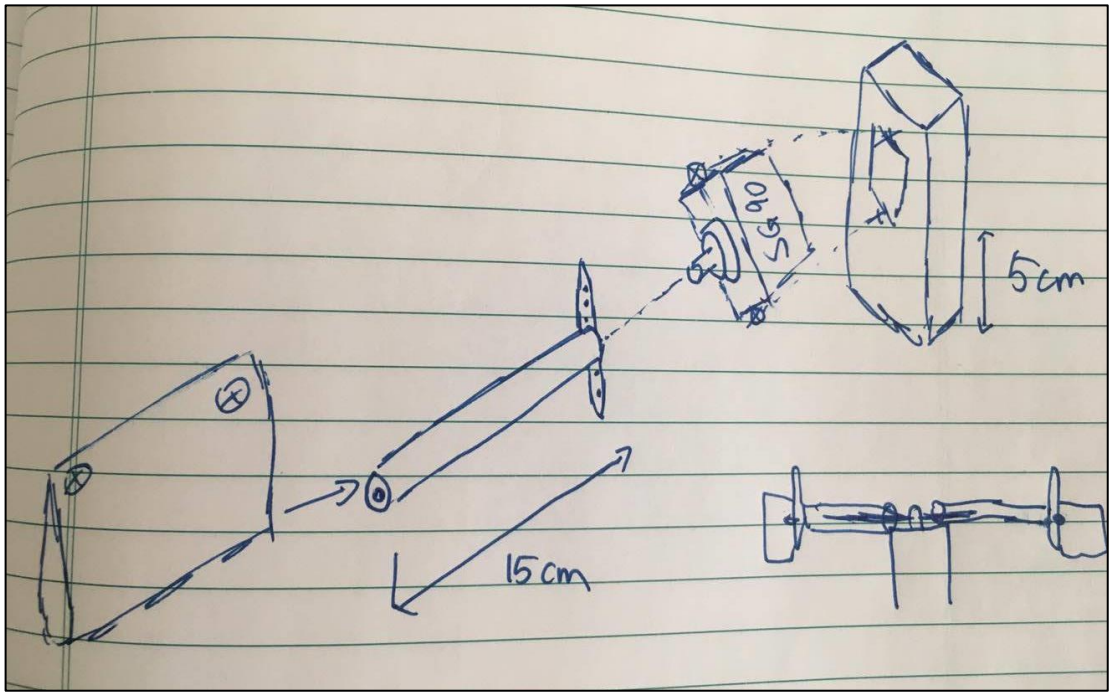


Figure 37: Schematics of Sorting Gates

Moment of Inertia and Torque Calculations

Moment of the motor,

$$I = \frac{m}{12}(4h^2 + w^2) = \frac{0.1kg}{12}(4 * 0.1^2 + 0.05^2) = 3.54 \times 10^{-4} kgm^2$$

Because we want the motor to have the speed to switch quickly, thus, assume that $\omega \; is \; 6$,

$$L = I \times \omega = 3.54 \times 10^{-4} kg\, m^2 \times 6\, rad/s = 2.125 \times 10^{-3} kgm^2/s$$

The motor has to be able to activate straight away so assume time = 0.3s

$$\tau = \frac{dL}{dt} = \frac{2.125 \times \frac{10^{-3} kgm^2}{s}}{0.3s} = 7.08 \times 10^{-3} kgm^2/s^2$$

SG90 has a torque of $1.80kg \cdot cm$ and we only need approximately $0.708kg \cdot cm$ of torque to drive the servo.

7.3.6 Detachable Tray

To address the numerous criteria related to the detachable tray, it has been determined that store bought plastic tray are the most cost and time efficient method to go. Not only are the containers prebuilt and can easily be separated by colour, but also the price point at which they were purchased is much lower than that of a container that is built using aluminum or wood. Each container cost 75 cents in Canadian. These bins are ideal, as they only weigh 7.8g each with a length of 23 cm, width of 18 cm and height of 6 cm. Not only do these fits in with the criteria of portable and compact, they also can be easily distinguishable based on their colours

Figure 38: Sorting Trays

7.3.7 Mounting

To address the concerns regarding mounting, an additional panel of 1/4” ply wood has been installed on the side of the wall, as shown in Figure 7.18. This panel allows a secure installation of all the circuitry components as well as the PIC board

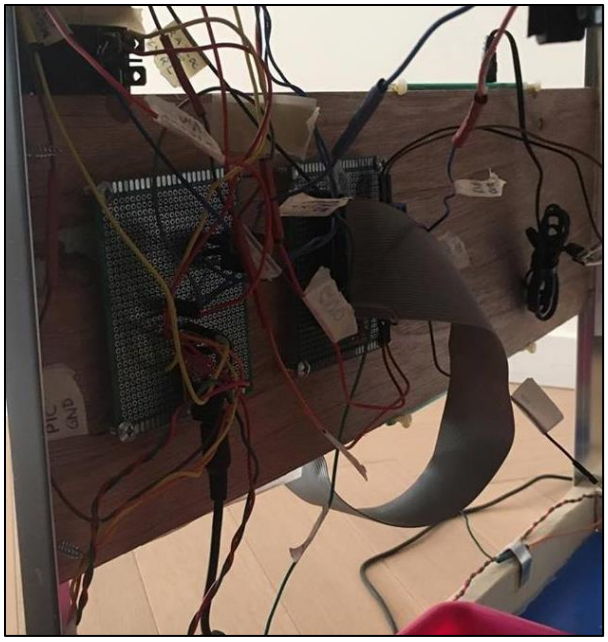


Figure 39: Mounting Board

7.4 Improvements to electromechanical systems

This section highlights the potential improvements that can be made to each design feature as well as the overall electromechanical system.

In general, the quality of the electromechanical systems can be improved not only with experience but also with extensive planning. As such, it is crucial to be able to visualize the design from drawings and have a plan on how to bring design into reality.

7.4.1 Frame

The actual main frame that is constructed is actually a little bit over the design dimension of 55cm x 55cm x 45cm, as it has dimension of 55.3cm x 55.2cm x 45cm. The extra 0.2cm and 0.3 cm is due to the unaccounted thickness of the angular aluminum piece as well as the protruding screw and washer. To ensure the size of the machine is under the design constraint, the two-piece mentioned above should be considered when designing for the frame in the future, as there should be some leeway between the design and reality.

7.4.2 Loading Mechanism

During testing, many runs resulted in bottles jamming in the loading mechanism. One way to solve this without implementing any extra feature is to add an H-Bridge to the DC motor such that the centrifugal cone can rotate both clock and counter-clock wise. Because our machine rotates in a clock wise direction, if bottles are jammed, it will be helpful if the motor can reverse direction to loosen up the jamming and then continue to the usual clock wise direction. As such, a configuration like 4 seconds in the clock wise direction and 0.5 seconds in the reverse might significantly reduce chances of jamming.

To even further reduce any potential jamming, additional actuation mechanism can be implemented to agitate the bottles and prevent them from staying in one spot. For example, one solenoid that perpetually hits the side of the wall can sometimes give the bottle the necessary movement to go through. Or, another

spinning dowel that can be placed at the beginning of the transporting tube to prevent any bottles from stacking on top of each other when entering the tube. In the case of stacking, the spinning dowel will pop the bottle on top back into the centrifuge where it will realign itself and then enter the entrance in the correct orientation.

To better the efficiency of the motor, it is possible reduce the height of the threaded rod to fully engage the ball bearings, as it occurs at times that the ball bearings on Lazy Susan are not engaged, resulting in an uneven movement of the centrifuge. Although this uneven motion reduces jamming, an overall built precision is preferred.

7.4.3 Transporting Tube

Although the transporting aluminum foil duct/tube is a very good design solution to address the objectives, the inner side of the tube can be further smoothened such that the bottle will be falling as if it is a smooth surface. This will require a lot of precise manufacturing. If budget permitted, a custom-made aluminum tube in the shape of curving U will be a lot more desired, as it not only can have all the upsides of the current aluminum foil duct, but also possess a smooth inner wall that aid the streamline process of bottles.

There are minor jamming issues later, as bottles lost their momentum while traveling down the tube. It is then plausible to aid the bottles by introducing more wheels that can be located on the sides as well as the bottom to provide the bottles with the little extra momentum they need.

7.4.4 Sorting Guide Rails

The biggest issue the guide rails encountered was the inconsistency in holding its position. To combat this, a parabolic curvature piece should be attached at the end, joining the two guiding rails. This will allow the ends to be permanently fixed together and thus reduce any movement horizontally that might allow a preemptive Eska Drop. Furthermore, this can also be addressed by installing a vertical support directly under each rail such that the position of the rails is completely fixed in place.

Another improvement that can be made is to completely revamp the rails such that the bottles will stop at the end of the tube, entering the rail and a set of wheels driven by gears and motor will guide the Eska bottle along while the Yop will directly drop below. This method will be much more consistent but will require a lot more materials and difficulties to manufacture.

7.4.5 Sorting Gates

One of the major jamming occurs at the sorting gates, as the Yop bottles cannot smoothly transition from the Guide Rails to the sorting gates, as they get caught between the rails and flap. The easiest way that this can be improved upon is to lower the overall height of the sorting gates, as they are currently 5 cm above the frame. With the extra room, the Yop for sure will have enough room to fall through and not jam the entire machine.

Furthermore, the aluminum divider installed in the middle should be brought slightly closer to the Yop side and a little bit taller. With these minor adjustments, the divider can then more efficiently prevent the Yop bottles from flying through the mechanism while still allowing Eska bottles to flow through.

7.4.6 Detachable Tray

Although the bins can potentially hold the edge case of 10 bottles in a single bin, the bottles have to be carefully arranged to prevent any bottles from flowing out of the machine. As such, a bigger container of the same type should be considered. One with same height but bigger in terms of both length and width to fully utilize the 25.5 cm X 25.5cm on the bottom of each quadrant.

7.4.7 Mounting

Although the current mounting mechanism is sufficient enough for the purpose of mounting all the circuitry and PIC board, the actual housing could be improved and there can be an enclosed mounting area that protect all the circuitry components and the PIC board from any potential outside damage.

8. Integration

8.1 General Conceptualization

Once each member has completed their own respective section of the project, integration begins immediately with the intention of combining each of the three individual subsystems into one cohesive machine. Due to general limitations of the robot, certain tasks need be performed before the completion of the next. The following sections will outline the completion of such tasks, and a summary of the completion and the timeline will be given shortly after.

8.2 Mounting of Actuators onto the Machine

While some of the mounting of the actuators was done by the electromechanical member of the group and was mentioned previously, the mounting of actuators had to be done deliberately in context of the rest of integration, because it was extremely important to optimize the position of actuators to prevent jamming throughout the process of the machine (see section 8.6 and section 7)

8.3 Mounting of Circuits onto machine, including the microcontroller board

This was the next necessary step after mounting the actuators, because after verifying the functionality of the actuators, it is important for the circuits to be mounted onto the board, so that they may:

1. Provide power to the actuators
2. Create autonomy for the whole robot

In the decision to choose were to mount the circuits on the board, the most important decision to be made was where to place the color sensor within our main mechanism of the robot. Because our design emphasized a continuous detection of the color sensor, the readings would be significantly different if the bottle was placed at different positions in the tube. This decision will be discussed in more detail in the section about calibration of the sensors.

In order to decide where to mount the circuits, it was necessary for us to minimize the amount of wiring from the microcontroller to all of the actuators and sensors, as well as the wiring from the power board to each of the circuits. Additionally, it was important that wires would not cross each other significantly to reduce the amount of interference in the circuits.

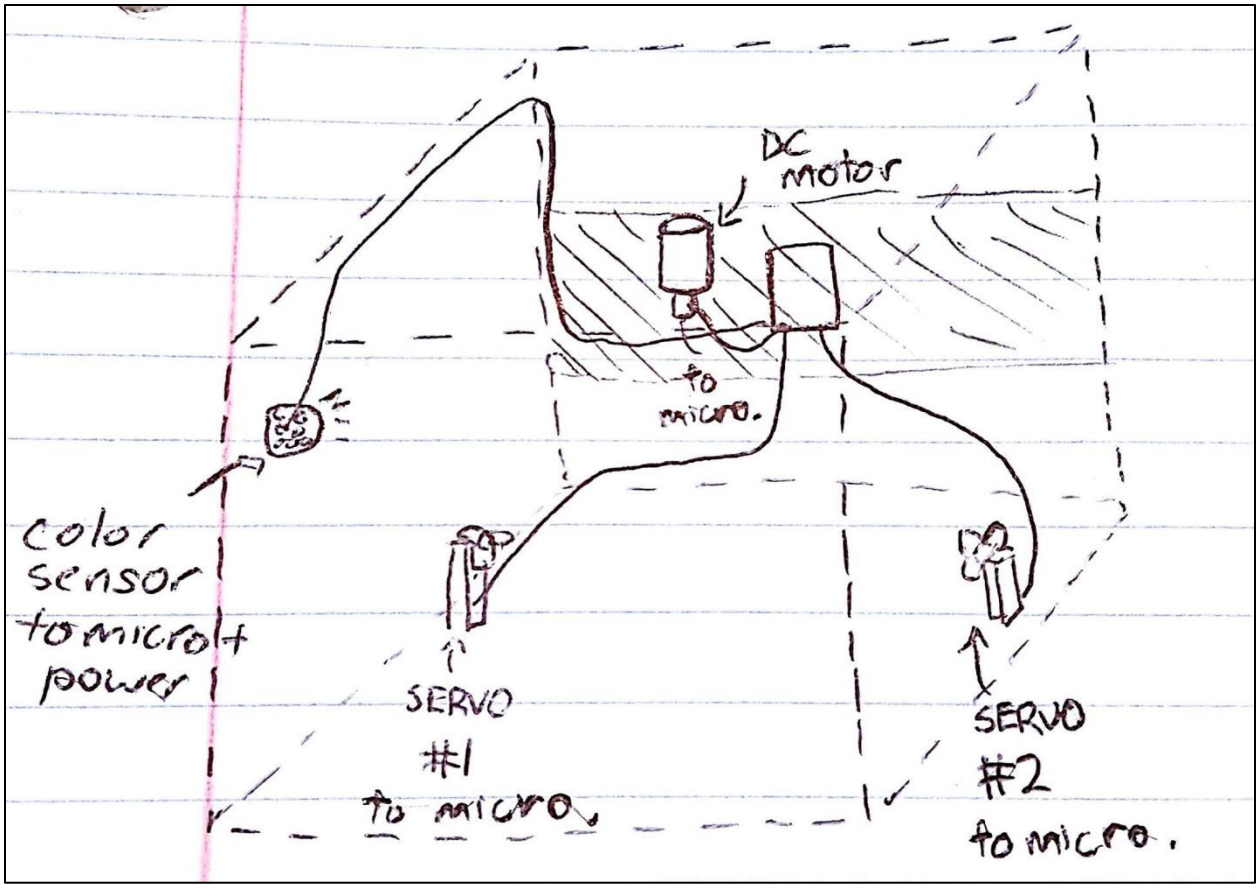


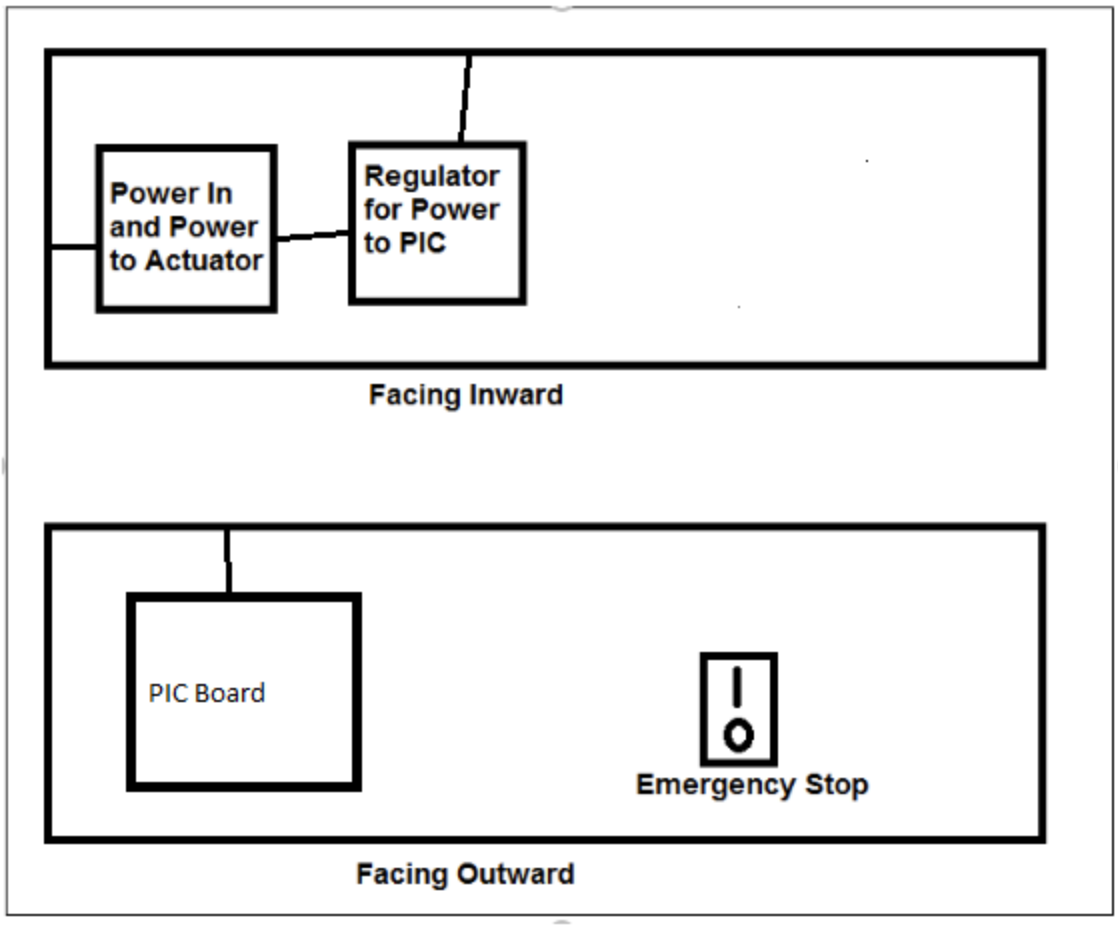
Figure 40: Machine Wiring

It was decided that the board would be mounted on one of the walls in the middle of the positioning of the two servo motors. By considering many different walls: the minimum possible wire was decided to be on the wall indicated in the above diagram out of the possible walls. However, the Top was excluded as a possibility because it would be much more difficult to access the circuits compared to a sideways configuration

Table 18: Position and Approximate Length of Wiring Needed

Position	**Approximate Length of Wiring Needed ***
Current Position	**4.75**
Right Wall	**5.25**
Left Wall	**5.00**
Top	**4.75**

*1 side length of the machine is ~0.55m



As a final decision, the Circuits were mounted a piece of plywood attached to the wall exterior of the frame as indicated in the frame above.

8.4 Interface Between Machine and Microcontroller

As mentioned previously, the decision of where to mount the microcontroller board was done in conjunction with the decision of where to mount the power of the machine, because the power needed to be regulated to the microcontroller board. Each of the actuator required a connection to the power circuit board and the microcontroller board, so it was an obvious choice to place the microcontroller board right beside the power. In order to organize the interfacing, wires were labelled and a ribbon cable was used from the pins of the microcontroller board and soldered onto the regulator board and wires were soldered onto that board to bridge the microcontroller signals to the actuators.

8.5 Calibration of the Sensors in the Context of the Machine

As Mentioned previously, the sensor positioning is extremely important, as it forms the basis of the algorithm and readings necessary for the detection of the bottle. The readings are extremely different depending on the position of the sensor. As per our design decisions, there are two possible positions within the machine where the sensor may be place:

1. Top section, near the Centrifuge
2. Within the Tube, where the bottle passes before reaching the bottom sorted section

Table 19: Advantages and Disadvantages for Each Sensor Placement

Position	**Advantages**	**Disadvantages**
Top of Machine	-Can be in many different positions for varying degrees of accuracy -Allows for flexibility in design	-Extremely Exposed to environmental light conditions -Bottles may not be in the correct position in front of the sensor for scanning
Within Tube of Machine	-Can be positioned in many different positions for varying degrees of accuracy and different algorithms -Relatively Obscure from light relative to the top of the machine	-Large variance in data due to different in flow of bottles through rough tube interior -May contribute to jamming in interior of bottle due to positioning -Aluminum tube may contribute noise to the circuits, or may short a circuit if wires are touching the metal accidentally

Upon the actual design, many modifications had to be made to the top centrifuge, while fewer modifications were made to the tube, which was consistently functional for our purposes. As such, it was decided that a small crevice would be cut out of the tube for the color sensor, and the color sensor will fill up the empty space of aluminum where the tube was, and the sensor would be continuously scanned within the tube and not the top of the machine.

8.5.1 Calibration Factors

Upon the choice of the tube, it became necessary to determine the positioning in the tube for the color sensor based on several factors:

1. Closeness to front and end of tube; relative exposure to external light changing readings in different environments like the lab and home

Because the color sensor varies significantly in different light conditions, it is necessary to make sure the color sensor is as enclosed as possible to reduce this variance. In the very middle of the tube there is the least light interference, so the sensor should be positioned near the middle of the tube.

2. Shape of bottle at position in tube; speed at which the bottle passes the position

Because the tube curves 180 to position the bottle for sorting, the speed is fastest along the straight portions of the tube, and the curved portion of the tube slows the bottle down significantly. In general, the bottle needs to be as slow as possible, to give the color sensor the time to take as many readings as possible.

3. Orientation of tube; probability of jamming due to color sensor being inserted

Because a hole needs to be cut out of the tube to insert the color sensor, the hole may cause the bottle to get caught on the color sensor as it flows down the tube. It is difficult to qualify which positions will cause the most jamming, because it is difficult to accurately qualify the actual flow of the bottle through the tube (there is too much variance). As such, any position that seems to not jam consistently is sufficient for the purposes of this project.

4. Closeness of Color sensor to bottle passing through tube; closer is better for more variance in readings

The closer the color sensor is when the bottle passes, the color sensor readings will change more significantly compared to if it is further away, improving the accuracy of an algorithm to detect the bottle type and cap.

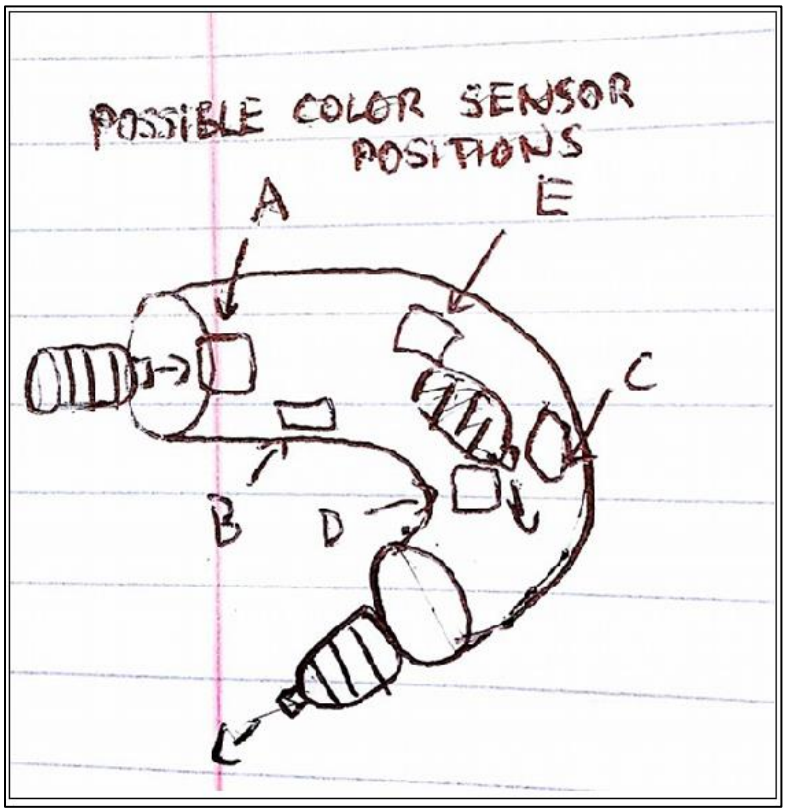


Figure 41: Color Sensor Positions

Table 20: Comparison Matrix for Sensor Positioning in Tube

Position in tube	Description	Sensitivity to Light	Probability of jamming	Speed of Bottle	Closeness to Bottle Passing
A	Beginning of Tube, Side	0	0	0	0
B	Beginning of Tube, Bottom	-	-	-	+

C	Middle of Tube, Side	+	0	+	+
D	Middle of Tube, Bottom	+	-	+	++
E	Middle of tube, Top	+	+	+	-

Based on calibration of the sensor, it was decided that the sensor would be placed in the side of the tube as indicated (position C).

The actual calibration of the sensor was described in full during the microcontroller section. The integration step of this calibration took an extremely long time, as issues with the top part of the machine not working properly made it very difficult to simulate actual run conditions, so integration and testing needed to be done with manually inserting bottles through the tube at a time. Additionally, the method chosen to use a data set and process it set continuously made the job of determining an algorithm very tedious. While a viable amount of accuracy with detection was achieved, especially with the Yop bottles, there is room for improvement, as mentioned in section 5.4.1.

8.5 Calibration of Servo Timing

Because the bottles continuously flow from the transport tube into the sorting mechanism, the servo motor actuation must be carefully timed such that they are quick enough to respond to the most recently sorted bottle, while also maintaining the position necessary for sorting bottles in the sorting mechanism for the time it takes for a bottle to pass through the sorting mechanism. Additionally, the servo timings must be precise enough to sort a string of bottles entering the sorting mechanism one immediately after the other in the case that each bottle requires a different servo orientation.

This timing was achieve through repeated experimentation. Bottles were dropped through the transport tube and the time it took for each bottle to travel through the sorting mechanism was evaluated. Based on the timings, the program code was changed to match the bottle's speed. In particular, the initial program processed the servo angles much too slowly, and required a combination of code optimization and reductions in program delays to allow for the servo to actuate fast enough to respond to a Yop bottle passing through the transport tube.

Also, to tackle the precision problem of the servo switching positions in between a string of bottles entering the sorting mechanism one after the other, the angles that the servo motor would rest at for each sorting category were adjusted. Initially, the angles were too large and the servo switched between angles too slowly to allow for sufficient bottle sorting. Through testing, we were able to determine a closer angle that allowed for correct sorting of each bottle while also improving the servo turning time between the angles enough so that a stream of bottles could be sorted.

8.6 Debugging of Jamming of Mechanical Mechanisms

This involved debugging the entire internal mechanism of the machine, from the flow of the bottle from the top mechanism to the tube to the guide rails to the servo-controlled metal flaps at the bottom of the machine. With so many different areas where the bottle could jam, it was a difficult to task to optimize the jamming of the machine, and ultimately this is what led to the disqualification of the robot. This is discussed in much further detail in the electromechanical section.

8.7 Overall Integration/System Improvements

There were many different aspects of integration would were not as optimal as expected. Already, there were lots of improvements which were suggested to the machine through integration of separate parts of the machine, as well as individual aspects of the entire machine. There are several key aspects of integration and the system which should be addressed should the project try to be repeated or the machine replicated.

8.7.1 Standardization of Design

The machine was designed with a centrifugal hopper, which was to be constructed as a method to load the bottles into the tube to be scanned. Additionally, the specifications of size and shape of the machine were initially laid out using calculations for the bottle to pass through. When implemented as a physical design, there were many different issues that led to the ultimate failure of the machine. During the debugging process, what happened was a very rough trial-and-error method where we attached pieces to the centrifugal cone in an attempt to correct the jamming. While this did end up serving its purpose in the end, every single part of the design was adjusted and very sensitive to calibration:

1. Centrifugal cone had multiple notches attached roughly with epoxy of varying sizes and shapes
2. Centrifugal cone metal wall had a large triangular notch attached
3. The Central Tube was shaped into a rough curve which is difficult to standardize
4. The inside of the tube was corrected with aluminum tape at several positions
5. The guide rails at the end were secured into a very awkward position with epoxy
6. The servo flaps at the bottom had their angle adjusted many times to mitigate jamming of the bottles as they exited the tube

With so many mechanisms that didn't follow the original design specifications, it is difficult to tell what exactly is causing the problem of jamming in the machine. If the machine were to be redesigned, it would be beneficial to make more prototypes to examine the jamming problem more thoroughly, and design features around those. In that case, large-scale manufacturing would be possible of the machine and its certain parts, rather than manually calibrating and adjusting each part of the machine.

8.7.2 Size and Weight Constraints

The machine was designed with a size constraint of a 55cm cube in mind. With this amount of size, it proved extremely difficult to implement a method to pass the tubes through without jamming. Because we were restricted to a smaller area, the tube that was designed to flow the bottles through the machine was prone to jamming. If the machine were able to designed with a larger volume, it would have been significantly easier to prevent jamming of the bottles, simply by having the ability to construct wider and steeper mechanisms which would aid the flow of the bottle (see Figure 42).

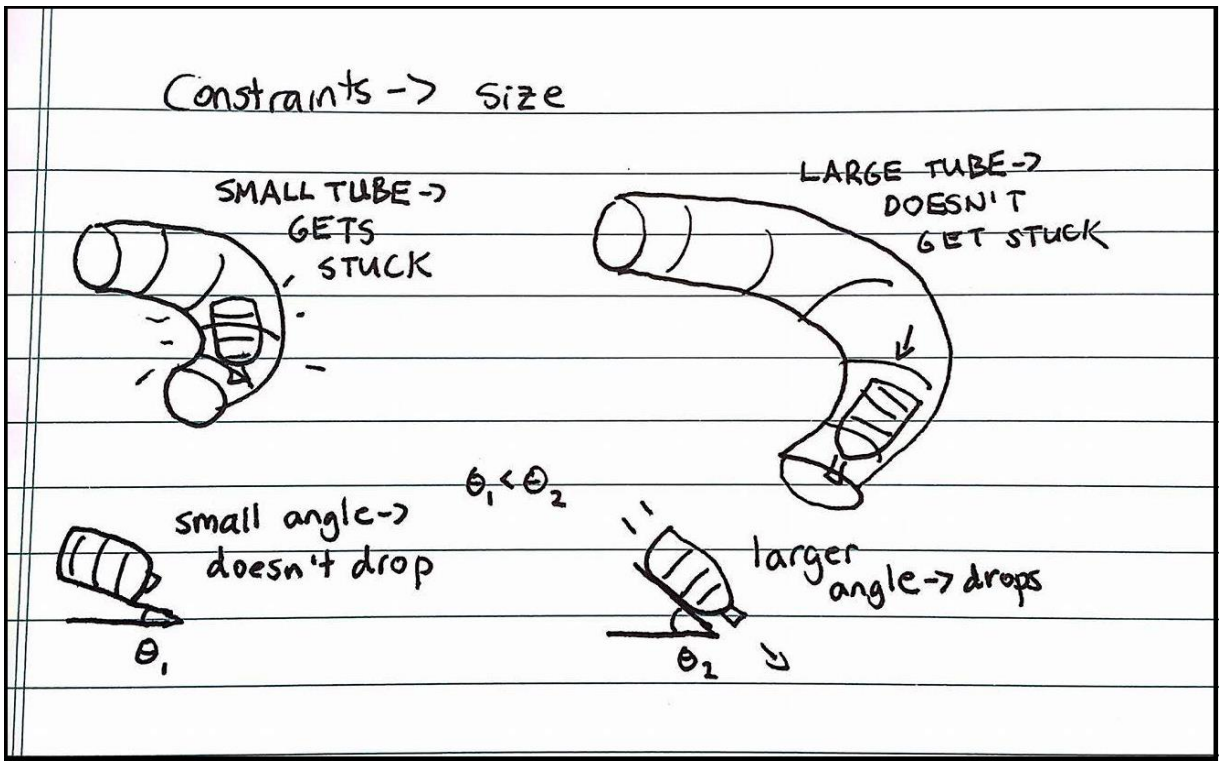


Figure 42: Possible Tube Improvements

8.7.3 Color Sensor Accuracy

The color sensor algorithm was key to our design, and when its accuracy was low, the overall detection of bottles was inaccurate, due to it being the sole sensor in our machine. As mentioned previously, it was very sensitive to changes in environmental light. When moving the machine from an external working environment to the AER201 lab, the color sensor required calibration to the light in the room, and this led to differing levels of accuracy of the algorithm. In order to combat this, it would have been beneficial to ensure that the color sensor would operate in a completely dark environment regardless of the external light, by boxing either the color sensor off from the environment or boxing the entire machine off using light, using external walls (see Figure 43).

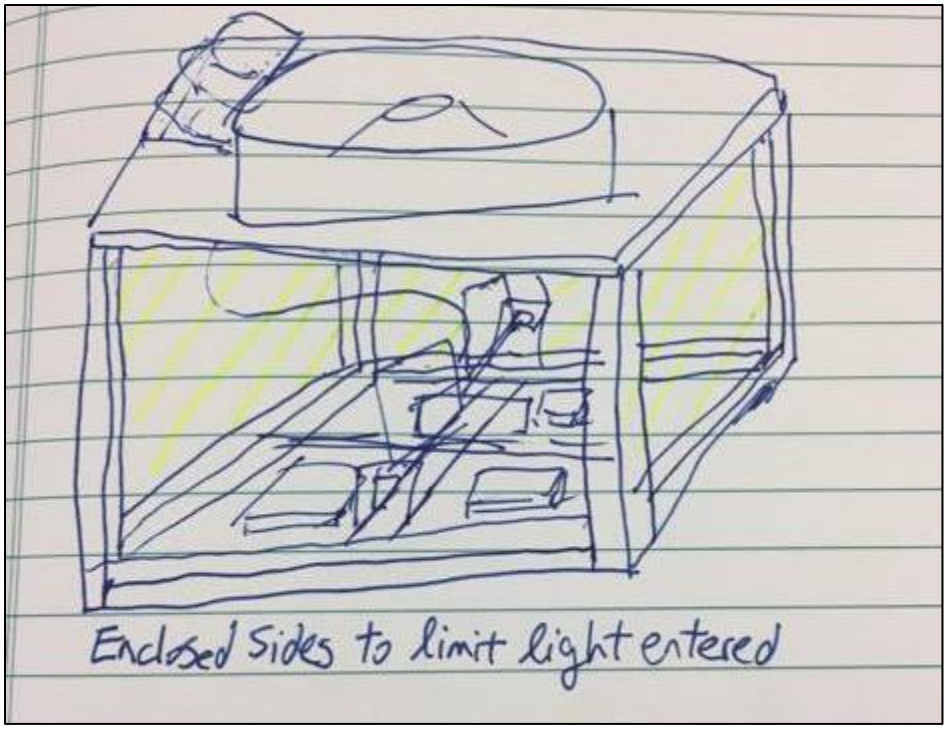


Figure 43: Possible Side Improvements

8.7.4 Overall Bottle detecting concept

The method chosen to detect the bottles was ambitious. In order to comply with our design value of design for speed, it was chosen that the bottle would be continuously scanned in order to detect the bottle type, and the presence of a cap. Firstly, combining both the detection of the cap and the bottle type made the algorithm difficult, and this was only made more difficult by the necessity of using only one sensor collecting continuous data. While in the end the algorithm reached levels of accuracy that were acceptable, it was still not over 95% accurate, which is significantly lower than industry level accuracy of plastic sorters. During testing, thresholds of readings using a fixed bottle position and sensors like IR sensors yielded 100% accuracy of cap detection. Additionally, fixing a bottle and using a color sensor positioned ~1 inch away from the bottle yielded 100% accuracy of bottle type detection as well. While these methods are viable, they require the bottle to be stopped and fixed in a certain position, changing the overall design. This sacrifice increases the accuracy of the algorithm, but significantly changes the mechanical portion of the machine, as well as reduces the overall speed of the process (see Figure 44, Table 21).

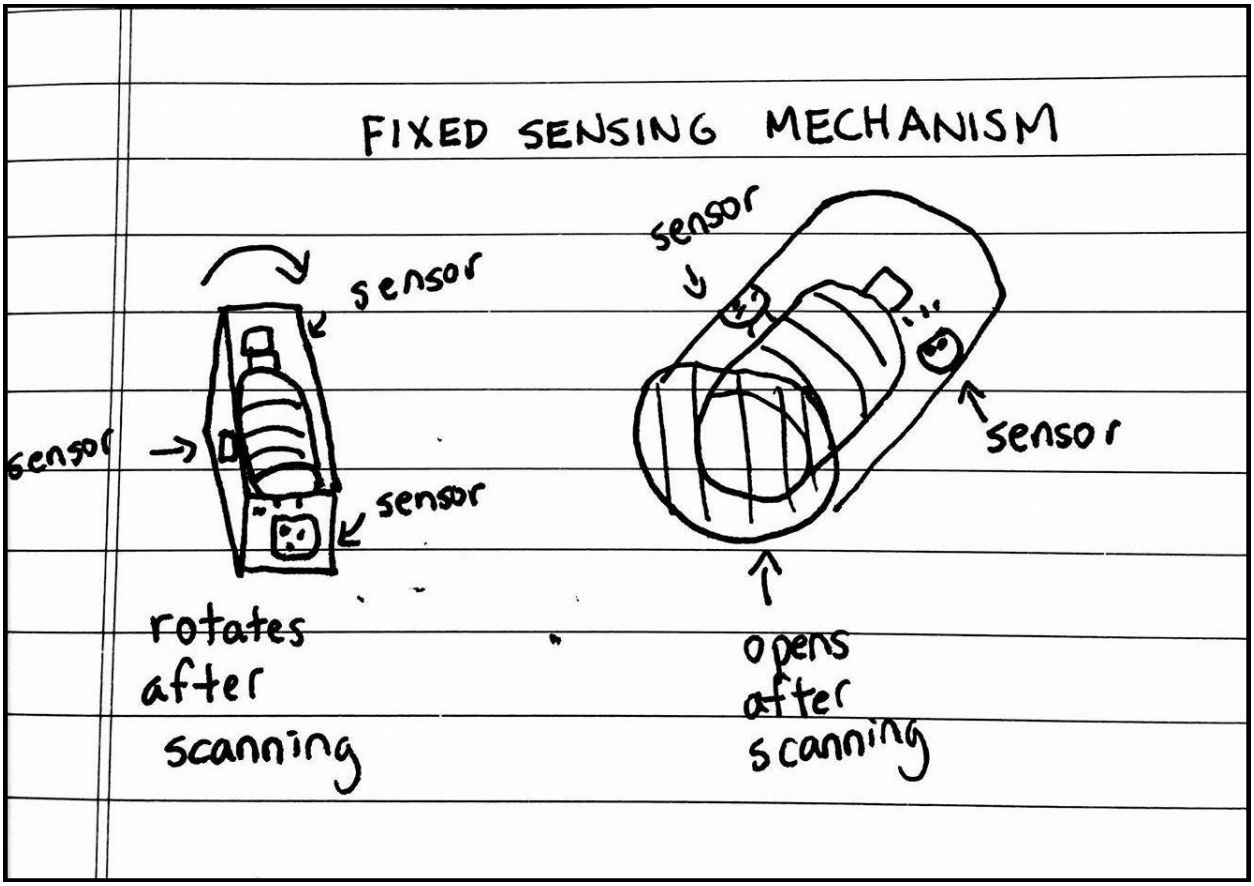


Figure 44: Fixed Sensing Mechanism

Table 21: Pros and Cons of Different Scanning Setups

	One sensor, Continuous Scanning	Multiple Sensors, Fixed Scanning
Pros of Design	-Makes the machine sorting significantly faster -Less circuitry is required -Makes the design have smoother mechanical flow	-Allows for greater accuracy in sorting -Reducing jamming by setting a fixed position, rather than random flow throughout a machine
Cons of Design	-Requires a complicated algorithm to scan and detect the bottle type/cap -Accuracy is low relative to fixing the scanning, and the algorithm is partial to changes in light	-Requires multiple sensors, so requires more circuitry and pins -Slows down the flow of the bottles through the machine significantly - Requires a mechanical mechanism to fix the bottle in place during operation

9 Project Scheduling

9.1 Initially Expected Schedule

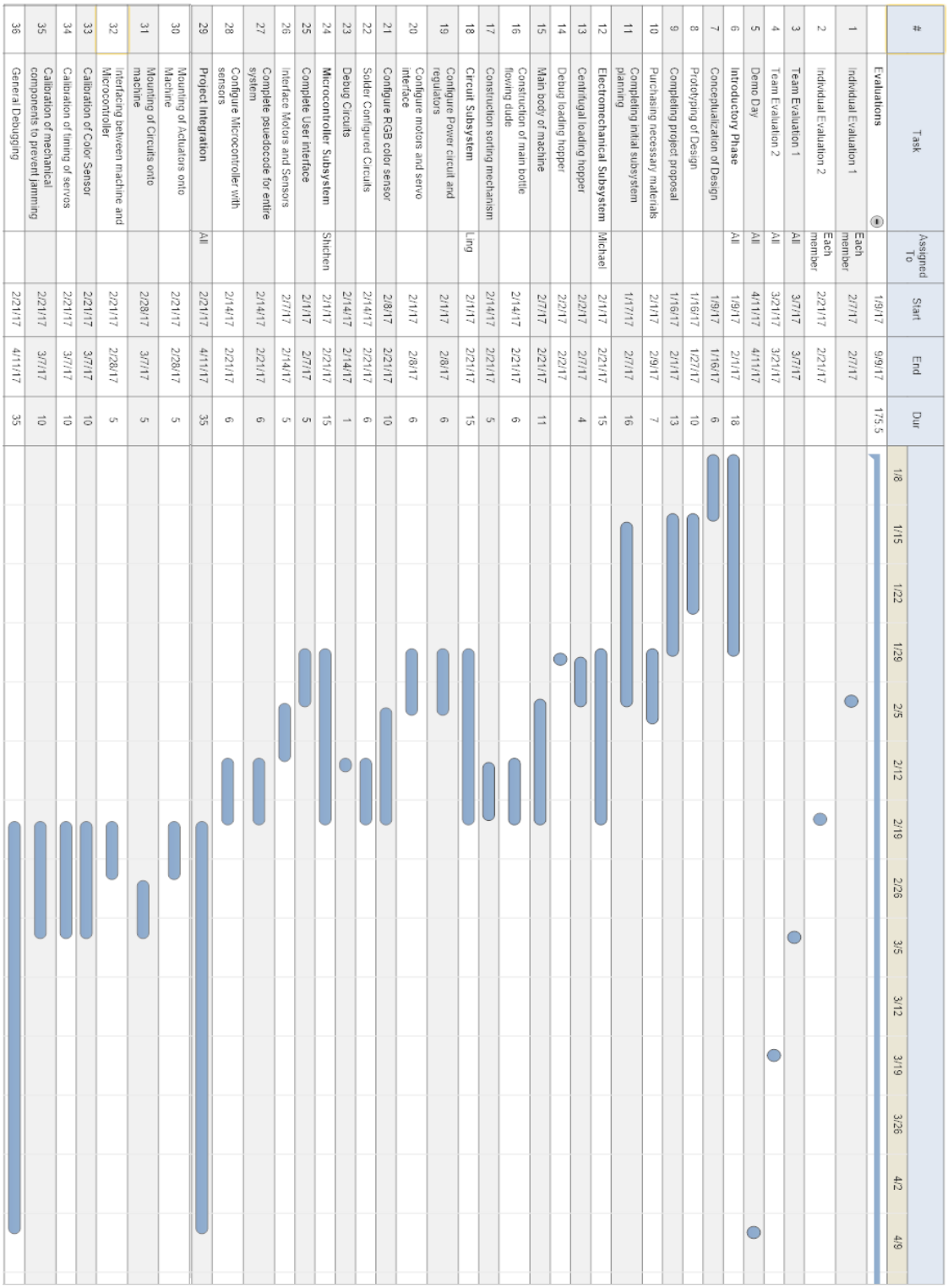


Figure 45: Initially Expected Schedule

9.2 Actual Schedule

#	Task	Assigned To	Start	End	Dur
	Evaluations		1/9/17	9/9/17	175.5
1	Individual Evaluation 1	Each member	2/7/17	2/7/17	
2	Individual Evaluation 2	Each member	2/21/17	2/21/17	
3	Team Evaluation 1	All	3/7/17	3/7/17	
4	Team Evaluation 2	All	3/21/17	3/21/17	
5	Demo Day	All	4/11/17	4/11/17	
6	**Introductory Phase**	All	1/9/17	2/1/17	18
7	Conceptualization of Design		1/9/17	1/16/17	6
8	Prototyping of Design		1/16/17	1/27/17	10
9	Completing project proposal		1/16/17	2/1/17	13
10	Purchasing necessary materials		2/1/17	2/9/17	7
11	Completing initial subsystem planning		1/17/17	2/7/17	16
12	**Electromechanical Subsystem**	Michael	2/1/17	2/21/17	15
13	Centrifugal loading hopper		2/2/17	2/7/17	4
14	Debug loading hopper		2/2/17	4/7/17	46.5
15	Main body of machine		2/7/17	2/21/17	11
16	Construction of main bottle flowing tube		2/14/17	4/7/17	39
17	Construction soiting mechanism		2/25/17	4/14/17	35
18	**Circuit Subsystem**	Ling	2/1/17	2/21/17	15
19	Configure Power circuit and regulators		2/1/17	2/8/17	6
20	Configure motors and servo interface		2/1/17	2/8/17	6
21	Configure RGB color sensor		2/20/17	3/10/17	15
22	Solder Configured Circuits		2/14/17	2/21/17	6
23	Debug Circuits		2/14/17	3/28/17	31
24	**Microcontroller Subsystem**	Shichen	2/1/17	2/21/17	15
25	Complete User interface		2/1/17	2/7/17	5
26	Interface Motors and Sensors		2/7/17	2/14/17	5
27	Complete psuedocode for entire system		2/14/17	2/21/17	6
28	Configure Microcontroller with sensors		3/10/17	4/12/17	24
29	**Project Integration**	All	2/21/17	4/11/17	35
30	Mounting of Actuators onto Machine		2/21/17	3/19/17	18.5
31	Mounting of Circuits onto machine		4/3/17	4/9/17	5
32	Interfacing between machine and Microcontroller		2/21/17	3/20/17	19.5
33	Calibration of Color Sensor		3/21/17	4/9/17	14
34	Calibration of timing of servos		3/21/17	4/10/17	15
35	Calibration of mechanical components to prevent jamming		3/21/17	4/10/17	15
36	General Debugging		2/21/17	4/11/17	35

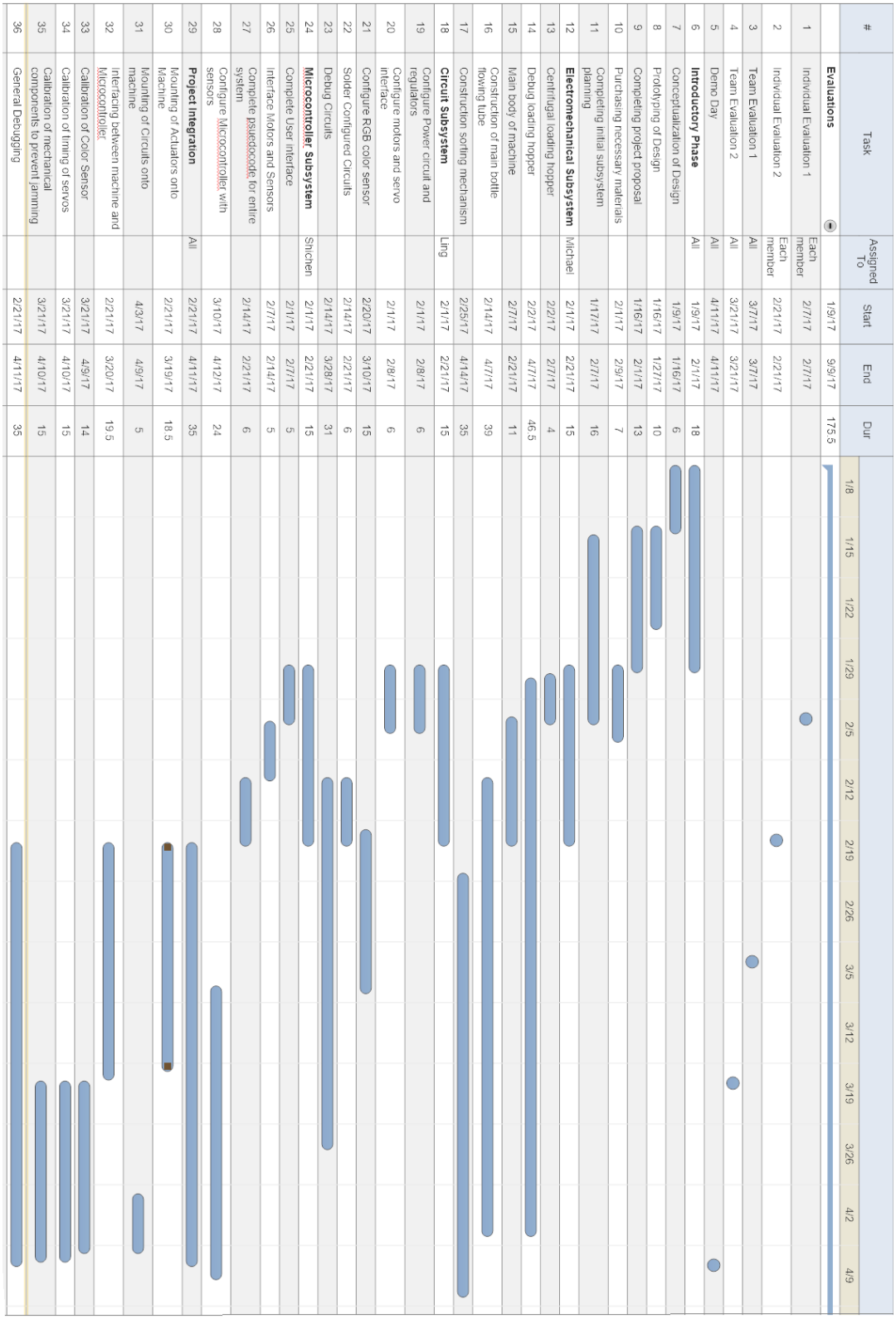


Figure 46: Actual Schedule

9.3 Project Management

9.3.1 Project Management Table

The major differences in project management are summarized in the table below (Table 22)

Table 22: Project Management Actual versus Expected Differences

Task	Expected Dates	Actual Dates	Reason for Difference in Scheduling
Debug Loading Hopper	2/2-2/4	2/2-4/11	It was expected that the subsystem member fix the mechanical components by an early date, but this turned out to be impossible because issues arose during integration which extended debugging until the end of the project.
Construction of main bottle flowing mechanism	2/1-2/7	3/21-4/7	This mechanism was initially constructed, and was scrapped in favor of a new design, so time had to be allocated near the end of the project for the reconstruction.
Construction of Sorting Mechanism	2/2-2/21	3/21-4/7	This mechanism was initially constructed, and was scrapped in favor of a new design, so time had to be allocated near the end of the project for the reconstruction.
Debug Circuits	2/14-2/21	2/14-3/28	New circuit components unintended from the original project meant that new circuits had to be constructed and debugged
Configure RGB color sensor	2/8-2/21	2/20-3/17	Configuring the sensor turned out to be extremely challenging on the software side, as a bug from transferring an algorithm from Arduino to PIC bugged the configuration.
Calibrate Microcontroller with Color Sensor	2/14-2/21	3/10-4/12	Again, the calibration was difficult because of issues not foreseen with the color sensor, as well as reconstruction of the mechanical parts of the machine. This led to last-minute calibrations of the sensor in the lab.
Calibration of Timing of Servos	2/21-3/7	3/21-4/10	Because the servos were reduced from 3 to 2, and the mechanism which they sorted was changed, it was necessary to redo the calibration and it was much more difficult to calibrate than anticipated due to mechanical issues slowing down bottle flow.
Calibration of mechanical components to prevent jamming	2/21-3/7	3/21-4/10	This was the biggest challenge of the entire project, and the most frustrating. While all the components were constructed early, jamming was a huge problem throughout the entire debugging process. After rebuilding the mechanical components and adding an extra DC motor to combat jamming, it still proved extremely difficult to prevent jamming, and as such the debugging of jamming lasted until demo.
General Debugging	2/21-4/11	3/21-4/11	Because our design significantly changed, debugging was significantly pushed back in scheduling.

9.3.2 Project Management issues

One of the biggest ideas with our initial schedule was that we expected that by Week 10, we were to have a mostly fully functioning robot, so that debugging would be completed over a long time, and there would be time for the final report to be completed. In the actual workings, we did have a functional robot by week 10, with an integrated ramp sorting system. However, it turned out that the design had too many fatal flaws with mechanical jamming and sensing, so that there was no way the task could be accomplished with the constraints and specific requirements. As such, at week 10 a complete overhaul of the design was made, with all the mechanical components excluding the centrifuge replaced. In addition, the centrifuge was given an almost complete overhaul.

9.3.2.1 Week 10 Design

In our week 10 design. We realized that the combination of building material, size of bottle, size restriction within the machine, and friction of surfaces made it impossible for the bottles to be sorted with a set of 2 servos. The construction consisted of a set of 3 ramps, that was intended to guide the bottle down the inside of then machine and then have 2 sets of servos which would divide the bottles into 4 different sections.

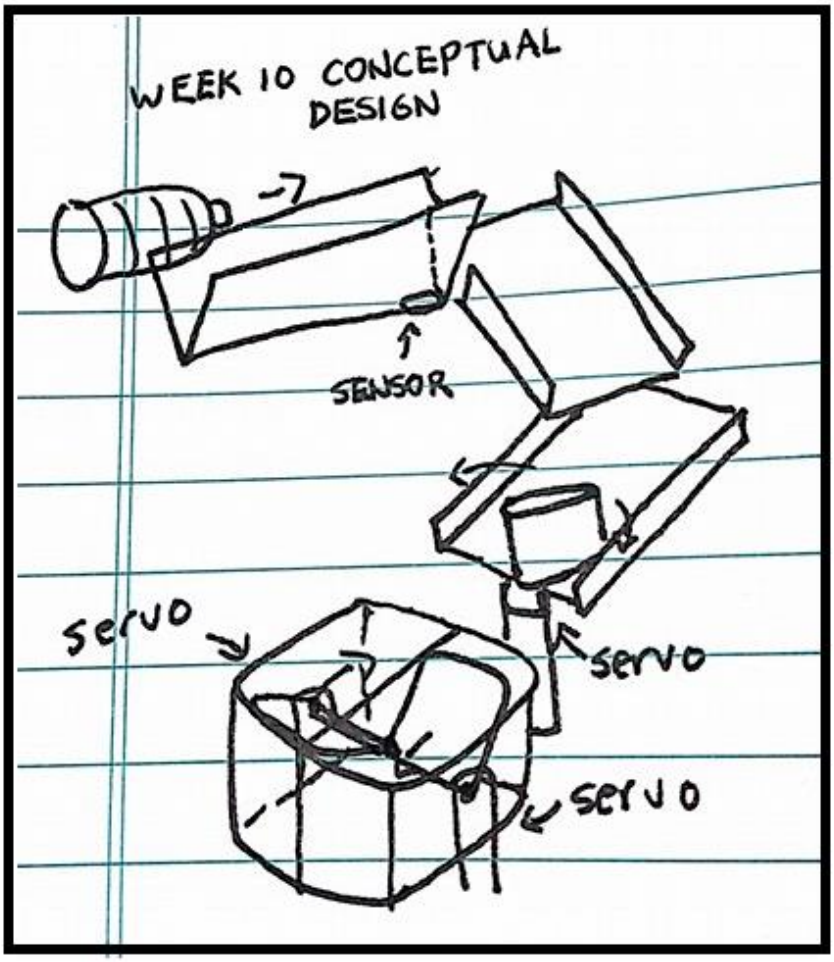


Figure 47: Week 10 Design

9.3.2.2 Week 12 Design

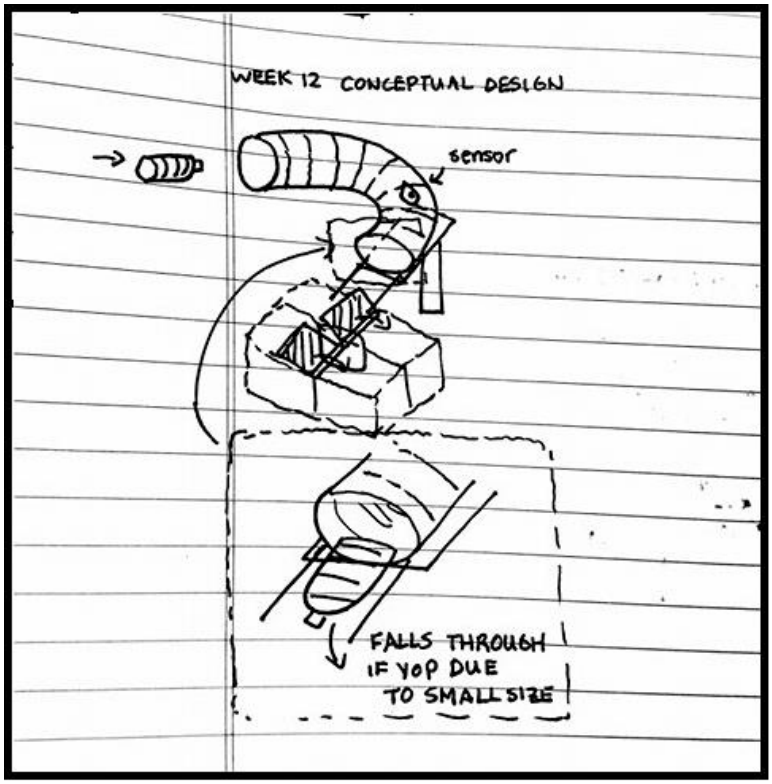


Figure 48: Week 12 Design

After creating a complete overhaul of the mechanical parts of the machine (as described in the electromechanical and project description sections), it was necessary to redo much of the steps that was intended to be performed earlier during the semester. This included redoing much of the calibration of the machine, as well as starting debugging from scratch, as a new mechanism introduced new bugs which needed to be fixed. However, the mechanical overhaul improved our jamming significantly, which was the intended goal, and while it initially set the schedule back, ended up improving our machine.

10. Description of Overall Machine

10.1 Project Design Overview

The solution that was developed over the semester is a 3-stage stationary robot of the dimensions 0.552m X 0.553m X 0.58m and weighs 4.9 kg. It is an automated machine that takes in a load of bottles containing Yop and Eska, and sorts the bottles according to their type and the presence of cap.

10.2 Full Design Schematics and Dimensions

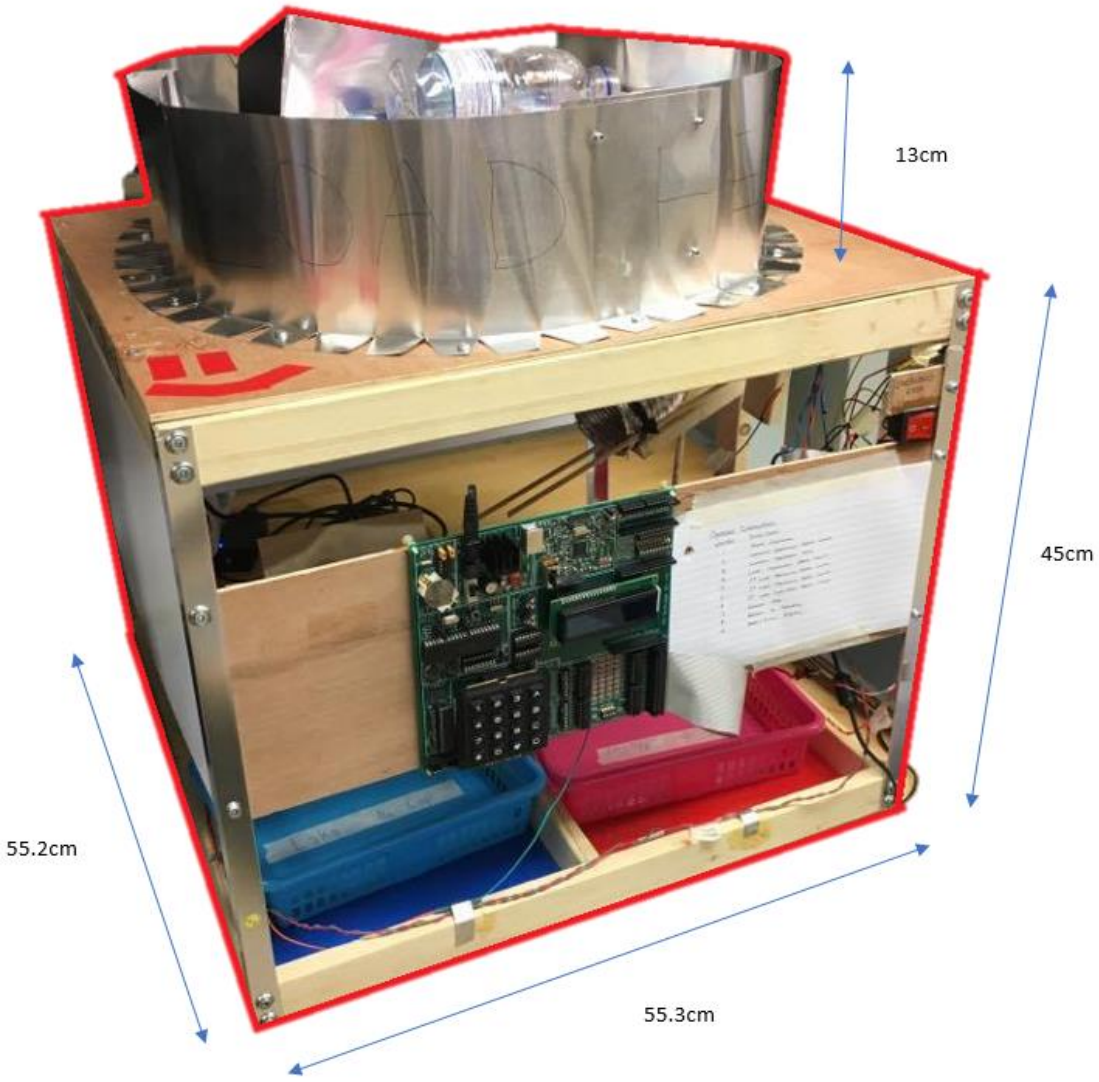


Figure 49: Overall Machine

10.3 Main Design Features

The chosen design has 3 main components which were chosen as a part of the design after a functional decomposition of the mechanisms necessary to complete the task that the robot is assigned. This is discussed further in the next section. These 3 mechanisms are a mechanism to feed the bottles through the machine, a mechanism to scan the bottles, and a mechanism to separate the bottles into their respective bins after they have been scanned and passed through the machine.

Design Feature 1: Centrifugal Loading Mechanism

The design features an area for the loading and feeding of the bottles into the rest of the machine. A Cone attached to a DC motor turns after the bottles are dumped onto the centrifuge, and turns, feeding the bottles one by one into the next part of the robot, for scanning. The centrifuge implements several features to prevent jamming of bottles and ensures that the bottles enter the main part of the robot one by one. There are three jamming prevention mechanism in place, two passive anti-jamming notches and one active spinning wheel that gives an extra acceleration to the bottles to alleviate jamming.

Figure 50: Overall Machine Top

Design Feature 2: Adjustable Tube Feed through machine, Color Sensor attached to side

This design feeds from the exit of the centrifugal loading mechanism, and is calibrated so that only one bottle may pass through the tube at once, while using gravity to turn the bottle 180 back towards the center of the machine for sorting. The tube has a crevice in the middle of the side which allows for the TCS34725 color sensor to be inserted. With the color sensor inserted, It is attached to the side of the tube and reads the bottles as they pass through the tube. The profile of the bottles is detected by the bottle and an algorithm is employed in order to determine the type of bottle, and whether it has a cap. By the time the bottle reaches the end of the pipe, the algorithm has already sent a signal to the sorting mechanism for the mechanism to sort the bottle into the correct bin.

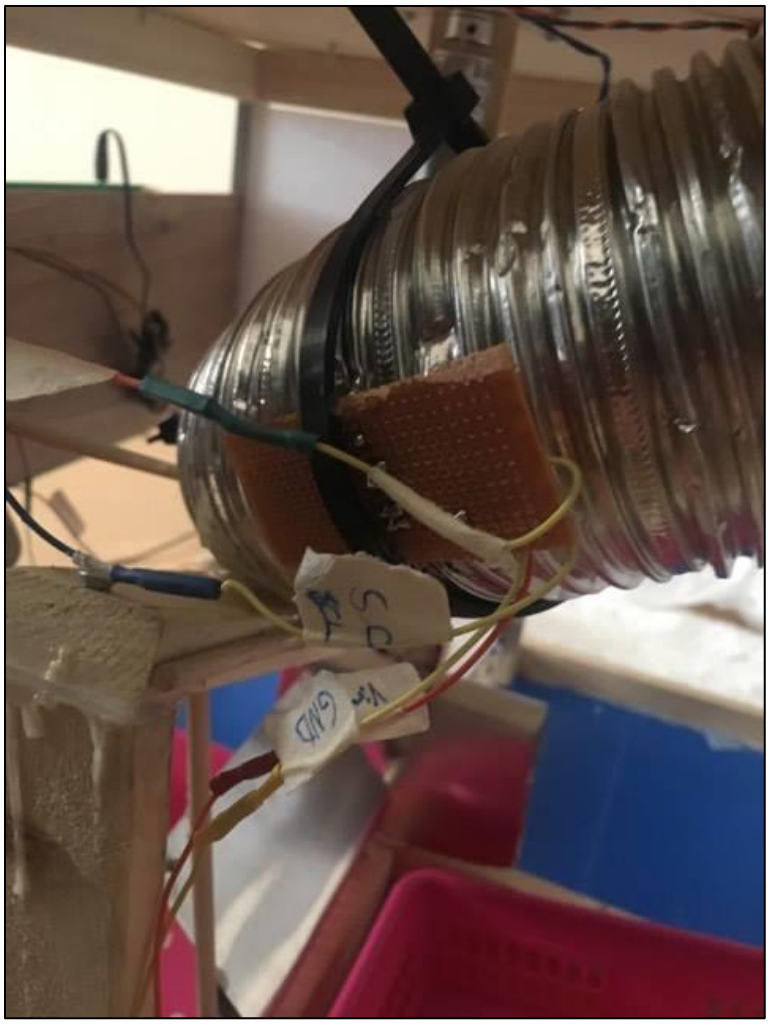


Figure 51: Color Sensor Setup

Design Feature 3: Sorting Rails and Sorting Gates

The design receives from the end of the adjustable tube feed and redirects the bottles into its corresponding bins based on their type and if they have cap or not. This feat is achieved by analyzing the color readings using an algorithm we have developed to differentiate between different bottle types. The sorting guide rails that continues after the adjustable tube feed serves as a mechanical differentiator to sort Eska and Yop based on their radius. Because Yop has a radius of 2.5cm while Eska has a radius of 2.9 cm, the distance between the guiding rails is 2.75 cm. This physical differentiator allows the Yop to drop through the guiding rails while the Eska will keep traveling through the rails. Then with the readings that the color sensor detected, the servo motors powering each sorting gates will receive signal to turn to a specific angle that will allow the bottles to drop into the correct bin.

Figure 52: Sorting Mechanism

10.4 Operating Flow Chart

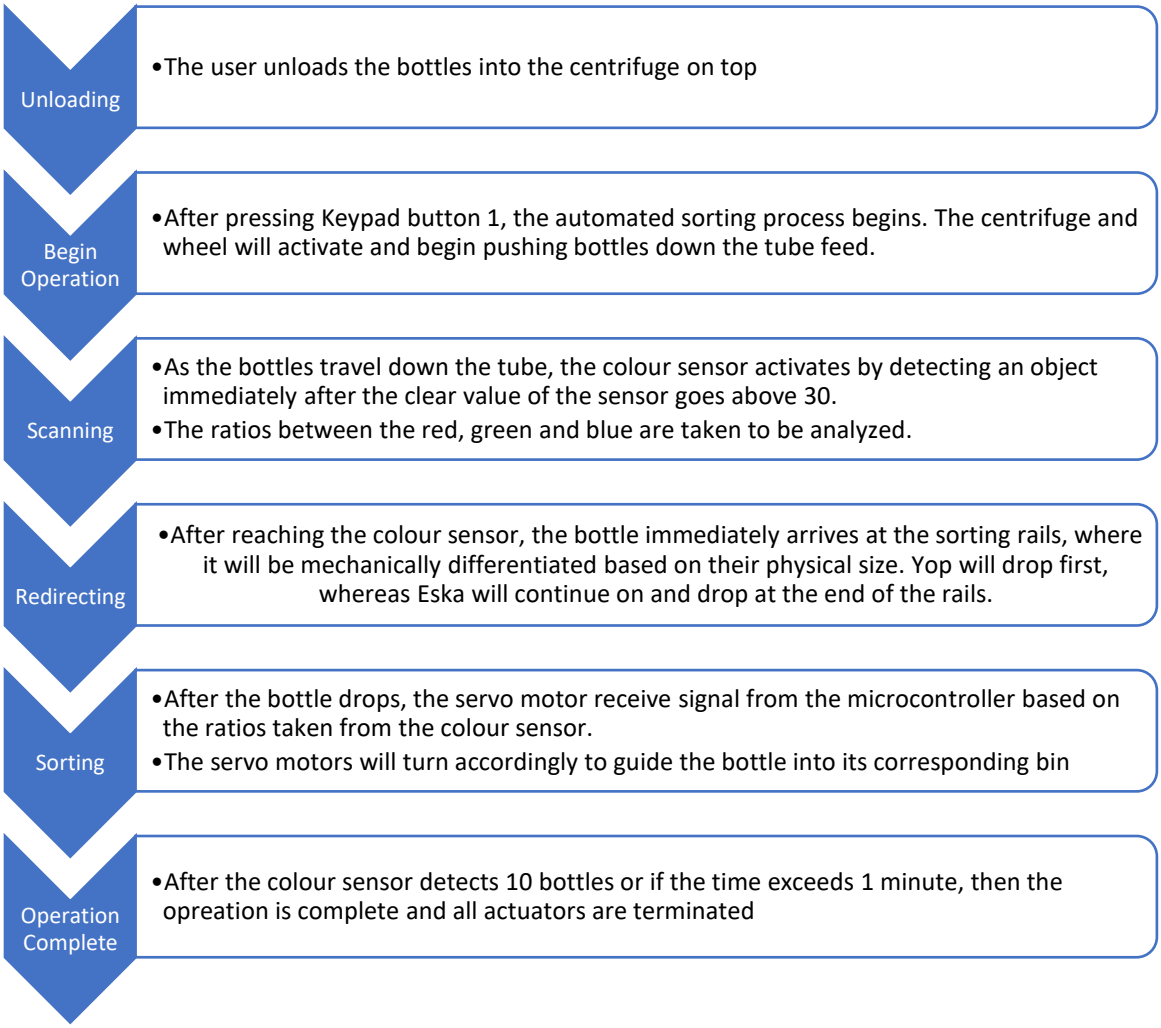


Figure 53: Operation Flow Chart

11. Standard Operating Procedure

11.1 Operating Conditions

In order to operate and run the machine, a standard North American 120V AC wall outlet must be available to power the machine. The machine must be placed on a flat surface. There must be no external weather conditions which may affect the functionality of the machine; the machine must be run indoors. During Operation, the machine's mechanical components should not be touched or otherwise moved. Only the PIC keypad and emergency stop button should be accessed during the operation of the machine.

11.2 Standard Operating Procedure

11.2.1 Setting up Machine and Sorting Bottles

1. After setting the machine down on a flat surface, plug the 3-pin power supply into a standard 120V/60Hz AC wall outlet. Place the removable bins into the 4 positions in the bottoms for the bins.

2. Switch the Emergency Stop Switch into the I position, and switch on the PIC board by toggling the power switch on the board to the on position.

3. Load 10 bottles into the top portion of the robot, within the circular metal area. The bottles must be loaded in the area indicated “Load here”.

4. To start Operation of the machine, Press 1 on the keypad of the PIC board. The LCD display should now display “Running”, and the motors should now be operating on the machine.

5. The machine should now autonomously sort the bottles over a period of approximately 1 minute. If at any time there is a need to stop the mechanical components of the machine from running, toggle the emergency stop into the O position to stop the functionality of the motor. If the emergency stop is toggled, the bottles must be removed from the machine and the procedure must be repeated from step 2.

6. After the bottles have been sorted, the LCD display should now display “Operation Done”, and the motors should automatically stop moving.

7. Remove the bottles from the bins at the bottom of the machine.

11.2.2 User Interface

The functionality of the user interface is indicated on the machine, for reference.

After at least one complete run of the machine (The LCD displayed “Running” and afterwards displayed “Operating Done”), The User interface may be accessed by pressing the buttons on the keypad. The Date/Time may be accessed at any time. Pressing the button on the keyboard will perform the following functions:

Table 23: Operating Instructions for Machine

Keypad Button	Function
1	Start Operation
2	Display Current Operation Log
3	Display Current Operation Time
A	Date/Time Display
4	Display Last Operation Bottle Count
5	Display 2nd Last Operation Log
6	Display 3rd Last Operation Log
B	Display 4th Last Operation Log
7	Manual Operation Stop

Note: this table from section 5.3.5 is reproduced here for convinience

12. Conclusion

Over the course of 16-weeks, we implemented a plan to construct, implement, and test a fully autonomous robot to sort plastic bottles into 4 different bins. The final design ended up being a product of many hours of hard work, prototyping, and debugging. The learning experience that was provided through this course was definitely unique, and helped developed each of the team member’s technical skills, writing skills, cooperative skills, and time management.

At demo, the machine was unable to sort the bottles fully to have a “qualified” run, due to jamming occurring within the mechanical mechanisms of the bottle. It sorted 2 bottles before being disqualified. Nevertheless, the functionality of the robot was deemed a success by the team members, as we fulfilled our personal goals in the construction and design of the robot, which was to observe functionality in the color sensor, and reduce jamming. In test runs in a different environment, we could sort the majority of the bottles several times.

Overall, the design of the robot turned out to be compact, elegant, and extremely fast. Our design values of simplicity and speed played a large role over the development of the project, and while this was a positive, it may have led to the failure of our robot during demo. In the proposal, we chose to go for a size and weight constraint that was significantly smaller than the maximum required, and this proved to lead to difficulties in implementations in the project. The project may have been improved by not imposing these more difficult constraints, or planning further in ahead as to the difficulty of a design with these additional constraints.

13. Reference and Bibliography

[1] Emami, Reza. Multidiscipliary Engineering Design. 2016. 2016. Print

[2]"Tcs34725 Color Sensor". N.p., 2017. Web. 13 Apr. 2017. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf

[3] T. D. A., “2010 ASME Design Competition - Waste Sorter,” YouTube, 29-Nov-2009. [Online]. Available: https://www.youtube.com/watch?v=f81ZA0ndc2o&app=desktop. [Accessed: 07-Apr-2017].

[4] ["Electrical Motor Efficiency", Engineeringtoolbox.com, 2017. [Online]. Available: http://www.engineeringtoolbox.com/electrical-motor-efficiency-d_655.html. [Accessed: 13- Apr- 2017].

[5] [1]"Friction and Friction Coefficients", Engineeringtoolbox.com, 2017. [Online]. Available: http://www.engineeringtoolbox.com/friction-coefficients-d_778.html. [Accessed: 13- Apr- 2017].

[6] "AER201 Vial Sorting Machine". YouTube. N.p., 2017. Web. 13 Apr. 2017. Available:https://www.youtube.com/watch?v=F1g0mAieCoo

[7]"Pololu 12V Step-Up Voltage Regulator U3V12F12". Pololu.com. N.p., 2017. Web. 13 Apr. 2017. Available :https://www.pololu.com/product/2117

[8]"TIP Data Sheet". N.p., 2017. Web. 13 Apr. 2017. Available :https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf

Appendix A: Circuitry Datasheets

STORAGE INFORMATION

Moisture Sensitivity

Optical characteristics of the device can be adversely affected during the soldering process by the release and vaporization of moisture that has been previously absorbed into the package. To ensure the package contains the smallest amount of absorbed moisture possible, each device is dry-baked prior to being packed for shipping. Devices are packed in a sealed aluminized envelope called a moisture barrier bag with silica gel to protect them from ambient moisture during shipping, handling, and storage before use.

The Moisture Barrier Bags should be stored under the following conditions:

Temperature Range	< 40°C
Relative Humidity	< 90%
Total Time	No longer than 12 months from the date code on the aluminized envelope if unopened.

Rebaking of the reel will be required if the devices have been stored unopened for more than 12 months and the Humidity Indicator Card shows the parts to be out of the allowable moisture region.

Opened reels should be used within 168 hours if exposed to the following conditions:

Temperature Range	< 30°C
Relative Humidity	< 60%

If rebaking is required, it should be done at 50°C for 12 hours.

The FN package has been assigned a moisture sensitivity level of MSL 3.

PRODUCTION DATA — information in this document is current at publication date. Products conform to specifications in accordance with the terms of Texas Advanced Optoelectronic Solutions, Inc. standard warranty. Production processing does not necessarily include testing of all parameters.

LEAD-FREE (Pb-FREE) and GREEN STATEMENT

Pb-Free (RoHS) TAOS' terms *Lead-Free* or *Pb-Free* mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TAOS Pb-Free products are suitable for use in specified lead-free processes.

Green (RoHS & no Sb/Br) TAOS defines *Green* to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information and Disclaimer The information provided in this statement represents TAOS' knowledge and belief as of the date that it is provided. TAOS bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TAOS has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TAOS and TAOS suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

NOTICE

Texas Advanced Optoelectronic Solutions, Inc. (TAOS) reserves the right to make changes to the products contained in this document to improve performance or for any other purpose, or to discontinue them without notice. Customers are advised to contact TAOS to obtain the latest product information before placing orders or designing TAOS products into systems.

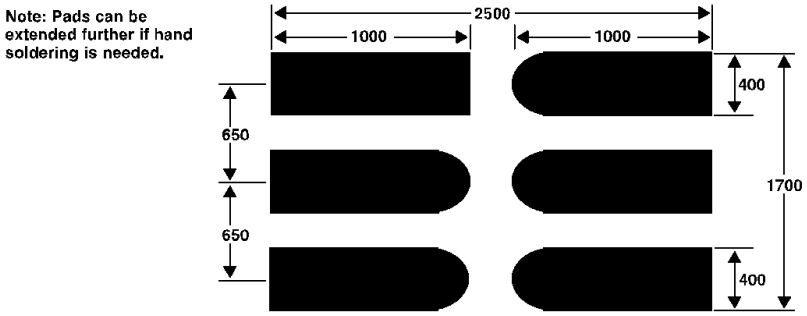
TAOS assumes no responsibility for the use of any products or circuits described in this document or customer product design, conveys no license, either expressed or implied, under any patent or other right, and makes no representation that the circuits are free of patent infringement. TAOS further makes no claim as to the suitability of its products for any particular purpose, nor does TAOS assume any liability arising out of the use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS, INC. PRODUCTS ARE NOT DESIGNED OR INTENDED FOR USE IN CRITICAL APPLICATIONS IN WHICH THE FAILURE OR MALFUNCTION OF THE TAOS PRODUCT MAY RESULT IN PERSONAL INJURY OR DEATH. USE OF TAOS PRODUCTS IN LIFE SUPPORT SYSTEMS IS EXPRESSLY UNAUTHORIZED AND ANY SUCH USE BY A CUSTOMER IS COMPLETELY AT THE CUSTOMER'S RISK.

APPLICATION INFORMATION: HARDWARE

PCB Pad Layout

Suggested PCB pad layout guidelines for the Dual Flat No-Lead (FN) surface mount package are shown in Figure 11.

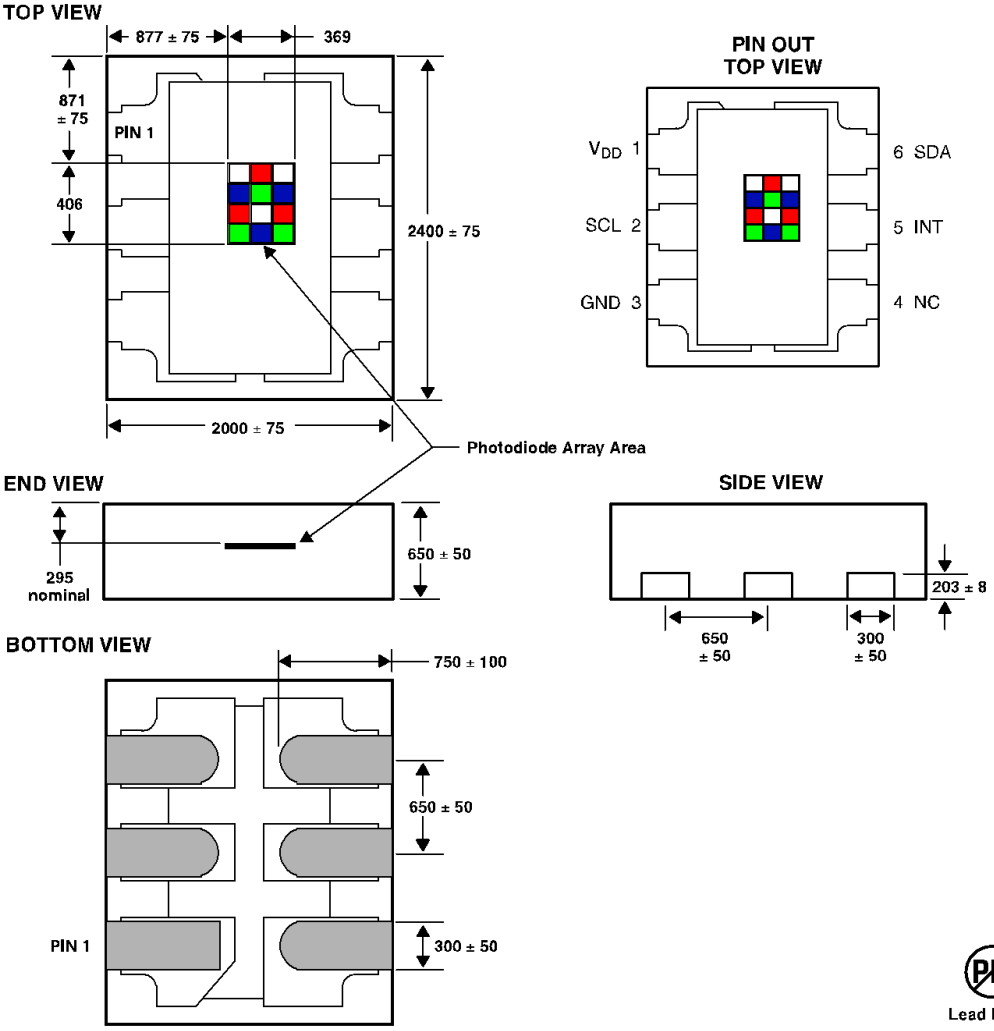


NOTES: A. All linear dimensions are in micrometers.
B. This drawing is subject to change without notice.

Figure 11. Suggested FN Package PCB Layout

PACKAGE INFORMATION

PACKAGE FN **Dual Flat No-Lead**



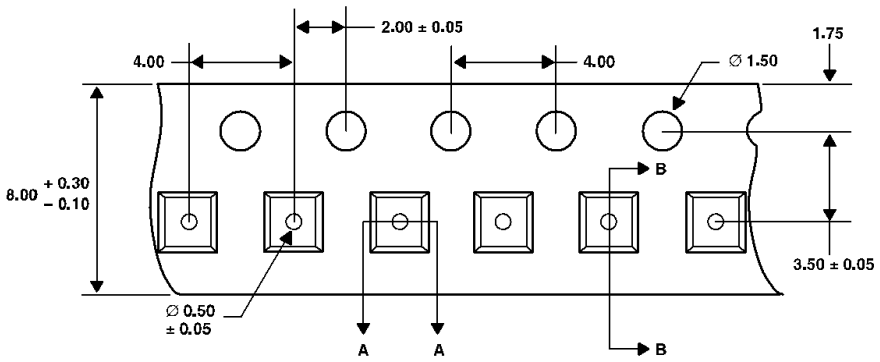
Lead Free

NOTES:
A. All linear dimensions are in micrometers. Dimension tolerance is ± 20 µm unless otherwise noted.
B. The die is centered within the package within a tolerance of ± 3 mils.
C. Package top surface is molded with an electrically nonconductive clear plastic compound having an index of refraction of 1.55.
D. Contact finish is copper alloy A194 with pre-plated NiPdAu lead finish.
E. This package contains no lead (Pb).
F. This drawing is subject to change without notice.

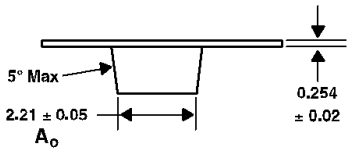
Figure 12. Package FN — Dual Flat No-Lead Packaging Configuration

CARRIER TAPE AND REEL INFORMATION

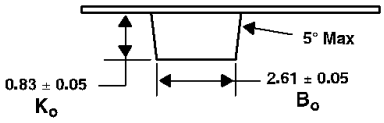
TOP VIEW

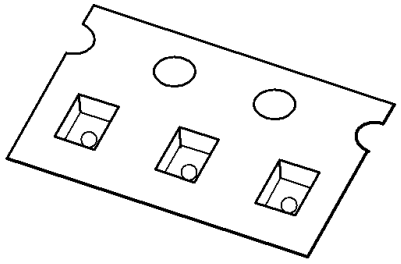


DETAIL A



DETAIL B





NOTES:
A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
B. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
C. Symbols on drawing A_o, B_o, and K_o are defined in ANSI EIA Standard 481–B 2001.
D. Each reel is 178 millimeters in diameter and contains 3500 parts.
E. TAOS packaging tape and reel conform to the requirements of EIA Standard 481–B.
F. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
G. This drawing is subject to change without notice.

Figure 13. Package FN Carrier Tape

SOLDERING INFORMATION

The FN package has been tested and has demonstrated an ability to be reflow soldered to a PCB substrate. The process, equipment, and materials used in these test are detailed below.

The solder reflow profile describes the expected maximum heat exposure of components during the solder reflow process of product on a PCB. Temperature is measured on top of component. The components should be limited to a maximum of three passes through this solder reflow profile.

Table 15. Solder Reflow Profile

PARAMETER	REFERENCE	DEVICE
Average temperature gradient in preheating		2.5°C/sec
Soak time	t_{soak}	2 to 3 minutes
Time above 217°C (T1)	t_1	Max 60 sec
Time above 230°C (T2)	t_2	Max 50 sec
Time above T_{peak} –10°C (T3)	t_3	Max 10 sec
Peak temperature in reflow	T_{peak}	260°C
Temperature gradient in cooling		Max –5°C/sec

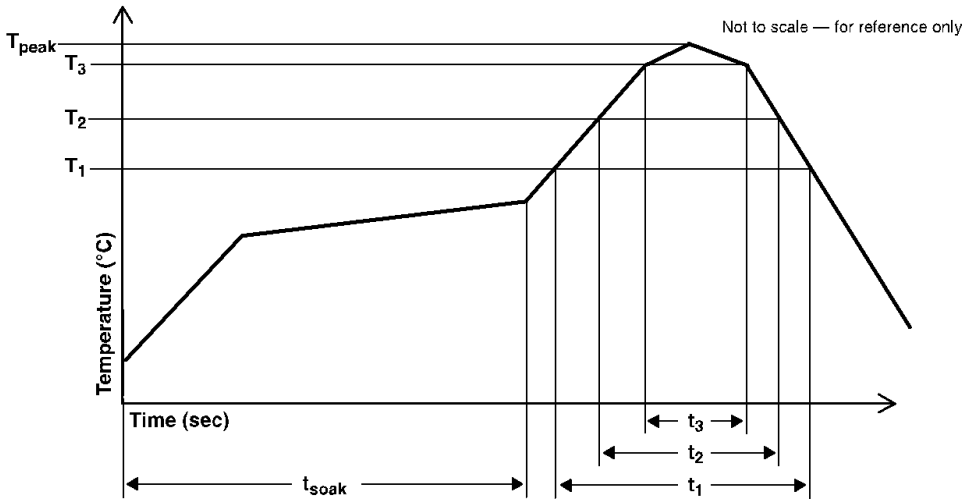


Figure 14. Solder Reflow Profile Graph

RGBC Timing Register (0x01)

The RGBC timing register controls the internal integration time of the RGBC clear and IR channel ADCs in 2.4-ms increments. Max RGBC Count = (256 – ATIME) × 1024 up to a maximum of 65535.

Table 6. RGBC Timing Register

FIELD	BITS	DESCRIPTION			
		VALUE	INTEG_CYCLES	TIME	MAX COUNT
ATIME	7:0	0xFF	1	2.4 ms	1024
		0xF6	10	24 ms	10240
		0xD5	42	101 ms	43008
		0xC0	64	154 ms	65535
		0x00	256	700 ms	65535

Wait Time Register (0x03)

Wait time is set 2.4 ms increments unless the WLONG bit is asserted, in which case the wait times are 12× longer. WTIME is programmed as a 2's complement number.

Table 7. Wait Time Register

FIELD	BITS	DESCRIPTION			
		REGISTER VALUE	WAIT TIME	TIME (WLONG = 0)	TIME (WLONG = 1)
WTIME	7:0	0xFF	1	2.4 ms	0.029 sec
		0xAB	85	204 ms	2.45 sec
		0x00	256	614 ms	7.4 sec

RGBC Interrupt Threshold Registers (0x04 – 0x07)

The RGBC interrupt threshold registers provides the values to be used as the high and low trigger points for the comparison function for interrupt generation. If the value generated by the clear channel crosses below the lower threshold specified, or above the higher threshold, an interrupt is asserted on the interrupt pin.

Table 8. RGBC Interrupt Threshold Registers

REGISTER	ADDRESS	BITS	DESCRIPTION
AILTL	0x04	7:0	RGBC clear channel low threshold lower byte
AILTH	0x05	7:0	RGBC clear channel low threshold upper byte
AIHTL	0x06	7:0	RGBC clear channel high threshold lower byte
AIHTH	0x07	7:0	RGBC clear channel high threshold upper byte

Persistence Register (0x0C)

The persistence register controls the filtering interrupt capabilities of the device. Configurable filtering is provided to allow interrupts to be generated after each integration cycle or if the integration has produced a result that is outside of the values specified by the threshold register for some specified amount of time.

Table 9. Persistence Register

	7	6	5	4	3	2	1	0	
PERS	Reserved				APERS				Address 0x0C

FIELD	BITS	DESCRIPTION		
PPERS	7:4	Reserved		
APERS	3:0	Interrupt persistence. Controls rate of interrupt to the host processor.		
		FIELD VALUE	MEANING	INTERRUPT PERSISTENCE FUNCTION
		0000	Every	Every RGBC cycle generates an interrupt
		0001	1	1 clear channel value outside of threshold range
		0010	2	2 clear channel consecutive values out of range
		0011	3	3 clear channel consecutive values out of range
		0100	5	5 clear channel consecutive values out of range
		0101	10	10 clear channel consecutive values out of range
		0110	15	15 clear channel consecutive values out of range
		0111	20	20 clear channel consecutive values out of range
		1000	25	25 clear channel consecutive values out of range
		1001	30	30 clear channel consecutive values out of range
		1010	35	35 clear channel consecutive values out of range
		1011	40	40 clear channel consecutive values out of range
		1100	45	45 clear channel consecutive values out of range
		1101	50	50 clear channel consecutive values out of range
		1110	55	55 clear channel consecutive values out of range
		1111	60	60 clear channel consecutive values out of range

Configuration Register (0x0D)

The configuration register sets the wait long time.

Table 10. Configuration Register

	7	6	5	4	3	2	1	0	
CONFIG	Reserved						WLONG	Reserved	Address 0x0D

FIELD	BITS	DESCRIPTION
Reserved	7:2	Reserved. Write as 0.
WLONG	1	Wait Long. When asserted, the wait cycles are increased by a factor 12× from that programmed in the WTIME register.
Reserved	0	Reserved. Write as 0.

Control Register (0x0F)

The Control register provides eight bits of miscellaneous control to the analog block. These bits typically control functions such as gain settings and/or diode selection.

Table 11. Control Register

	7	6	5	4	3	2	1	0	
CONTROL	Reserved						AGAIN		Address 0x0F

FIELD	BITS	DESCRIPTION	
Reserved	7:2	Reserved. Write bits as 0	
AGAIN	1:0	RGBC Gain Control.	
		FIELD VALUE	RGBC GAIN VALUE
		00	1× gain
		01	4× gain
		10	16× gain
		11	60× gain

ID Register (0x12)

The ID Register provides the value for the part number. The ID register is a read-only register.

Table 12. ID Register

	7	6	5	4	3	2	1	0	
ID	ID								Address 0x12

FIELD	BITS	DESCRIPTION	
ID	7:0	Part number identification	0x44 = TCS34721 and TCS34725
			0x4D = TCS34723 and TCS34727

Status Register (0x13)

The Status Register provides the internal status of the device. This register is read only.

Table 13. Status Register

	7	6	5	4	3	2	1	0	
STATUS	Reserved			AINT	Reserved			AVALID	Address 0x13

FIELD	BIT	DESCRIPTION
Reserved	7:5	Reserved.
AINT	4	RGBC clear channel Interrupt.
Reserved	3:1	Reserved.
AVALID	0	RGBC Valid. Indicates that the RGBC channels have completed an integration cycle.

RGBC Channel Data Registers (0x14 – 0x1B)

Clear, red, green, and blue data is stored as 16-bit values. To ensure the data is read correctly, a two-byte read I^2C transaction should be used with a read word protocol bit set in the command register. With this operation, when the lower byte register is read, the upper eight bits are stored into a shadow register, which is read by a subsequent read to the upper byte. The upper register will read the correct value even if additional ADC integration cycles end between the reading of the lower and upper registers.

Table 14. ADC Channel Data Registers

REGISTER	ADDRESS	BITS	DESCRIPTION
CDATA	0x14	7:0	Clear data low byte
CDATAH	0x15	7:0	Clear data high byte
RDATA	0x16	7:0	Red data low byte
RDATAH	0x17	7:0	Red data high byte
GDATA	0x18	7:0	Green data low byte
GDATAH	0x19	7:0	Green data high byte
BDATA	0x1A	7:0	Blue data low byte
BDATAH	0x1B	7:0	Blue data high byte

Register Set

The TCS3472 is controlled and monitored by data registers and a command register accessed through the serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Table 3.

Table 3. Register Address

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
—	COMMAND	W	Specifies register address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	ATIME	R/W	RGBC time	0xFF
0x03	WTIME	R/W	Wait time	0xFF
0x04	AILTL	R/W	Clear interrupt low threshold low byte	0x00
0x05	AILTH	R/W	Clear interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	Clear interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	Clear interrupt high threshold high byte	0x00
0x0C	PERS	R/W	Interrupt persistence filter	0x00
0x0D	CONFIG	R/W	Configuration	0x00
0x0F	CONTROL	R/W	Control	0x00
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	CDATAL	R	Clear data low byte	0x00
0x15	CDATAH	R	Clear data high byte	0x00
0x16	RDATAL	R	Red data low byte	0x00
0x17	RDATAH	R	Red data high byte	0x00
0x18	GDATAL	R	Green data low byte	0x00
0x19	GDATAH	R	Green data high byte	0x00
0x1A	BDATAL	R	Blue data low byte	0x00
0x1B	BDATAH	R	Blue data high byte	0x00

The mechanics of accessing a specific register depends on the specific protocol used. See the section on I^2C protocols on the previous pages. In general, the COMMAND register is written first to specify the specific control-status-data register for subsequent read/write operations.

Command Register

The command register specifies the address of the target register for future write and read operations.

Table 4. Command Register

	7	6	5	4	3	2	1	0	
COMMAND	CMD	TYPE		ADDR/SF					– –

FIELD	BITS	DESCRIPTION	
CMD	7	Select Command Register. Must write as 1 when addressing COMMAND register.	
TYPE	6:5	Selects type of transaction to follow in subsequent data transfers:	
		FIELD VALUE	**INTEGRATION TIME**
		00	Repeated byte protocol transaction
		01	Auto-increment protocol transaction
		10	Reserved — Do not use
		11	Special function — See description below
		Byte protocol will repeatedly read the same register with each data access. Block protocol will provide auto-increment function to read successive bytes.	
ADDR/SF	4:0	Address field/special function field. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-data register for subsequent read and write transactions. The field values listed below only apply to special function commands:	
		FIELD VALUE	**READ VALUE**
		00110	Clear channel interrupt clear
		other	Reserved — Do not write
		The Clear channel interrupt clear special function clears any pending interrupt and is self-clearing.	

Enable Register (0x00)

The Enable register is used primarily to power the TCS3472 device on and off, and enable functions and interrupts as shown in Table 5.

Table 5. Enable Register

	7	6	5	4	3	2	1	0	
ENABLE	Reserved			AIEN	WEN	Reserved	AEN	PON	Address 0x00

FIELD	BITS	DESCRIPTION
Reserved	7:5	Reserved. Write as 0.
AIEN	4	RGBC interrupt enable. When asserted, permits RGBC interrupts to be generated.
WEN	3	Wait enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
Reserved	2	Reserved. Write as 0.
AEN	1	RGBC enable. This bit actives the two-channel ADC. Writing a 1 activates the RGBC. Writing a 0 disables the RGBC.
PON [1, 2]	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.

NOTES: 1. See Power Management section for more information.
2. A minimum interval of 2.4 ms must pass after PON is asserted before an RGBC can be initiated.

Interrupts

The interrupt feature simplifies and improves system efficiency by eliminating the need to poll the sensor for light intensity values outside of a user-defined range. While the interrupt function is always enabled and its status is available in the status register (0x13), the output of the interrupt state can be enabled using the RGBC interrupt enable (AIEN) field in the enable register (0x00).

Two 16-bit interrupt threshold registers allow the user to set limits below and above a desired light level. An interrupt can be generated when the Clear data (CDATA) is less than the Clear interrupt low threshold (AILTx) or is greater than the Clear interrupt high threshold (AIHTx).

It is important to note that the thresholds are evaluated in sequence, first the low threshold, then the high threshold. As a result, if the low threshold is set above the high threshold, the high threshold is ignored and only the low threshold is evaluated.

To further control when an interrupt occurs, the device provides a persistence filter. The persistence filter allows the user to specify the number of consecutive out-of-range Clear occurrences before an interrupt is generated. The persistence filter register (0x0C) allows the user to set the Clear persistence filter (APERS) value. See the persistence filter register for details on the persistence filter value. Once the persistence filter generates an interrupt, it will continue until a special function interrupt clear command is received (see command register).

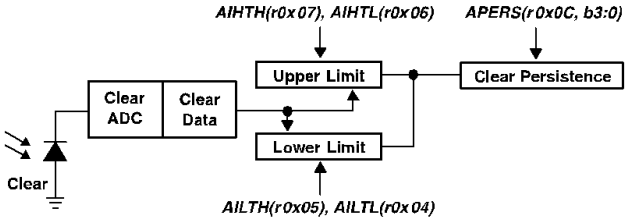


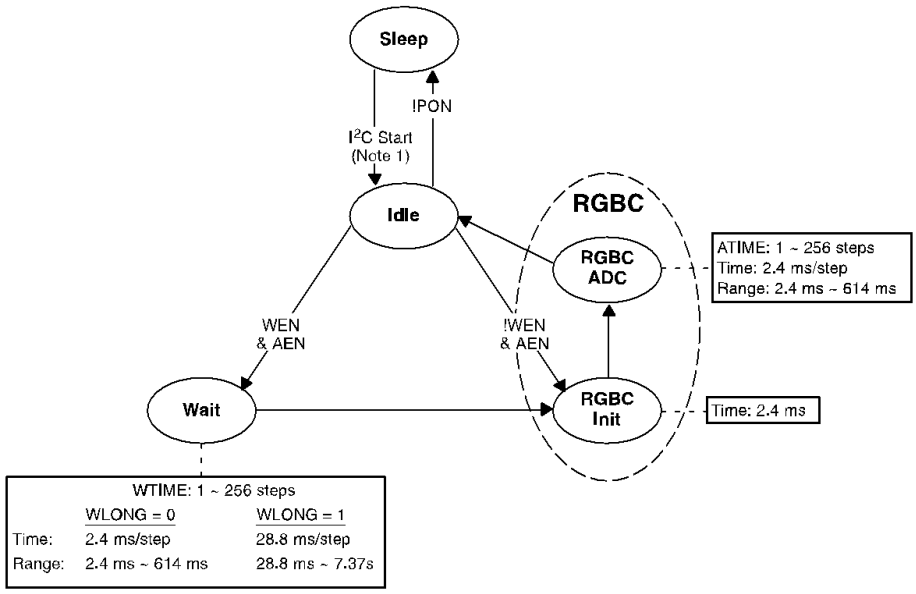
Figure 8. Programmable Interrupt

System Timing

The system state machine shown in Figure 5 provides an overview of the states and state transitions that provide system control of the device. This section highlights the programmable features, which affect the state machine cycle time, and provides details to determine system level timing.

When the power management feature is enabled (WEN), the state machine will transition to the Wait state. The wait time is determined by WLONG, which extends normal operation by 12× when asserted, and WTIME. The formula to determine the wait time is given in the box associated with the Wait state in Figure 9.

When the RGBC feature is enabled (AEN), the state machine will transition through the RGBC Init and RGBC ADC states. The RGBC Init state takes 2.4 ms, while the RGBC ADC time is dependent on the integration time (ATIME). The formula to determine RGBC ADC time is given in the associated box in Figure 9. If an interrupt is generated as a result of the RGBC cycle, it will be asserted at the end of the RGBC ADC.



Notes: 1. There is a 2.4 ms warm-up delay if PON is enabled. If PON is not enabled, the device will return to the Sleep state as shown.
2. PON, WEN, and AEN are fields in the Enable register (0x00).

Figure 9. Detailed State Diagram

Power Management

Power consumption can be managed with the Wait state, because the Wait state typically consumes only 65 µA of I_{DD} current. An example of the power management feature is given below. With the assumptions provided in the example, average I_{DD} is estimated to be 152 µA.

Table 1. Power Management

SYSTEM STATE MACHINE STATE	PROGRAMMABLE PARAMETER	PROGRAMMED VALUE	DURATION	TYPICAL CURRENT
Wait	WTIME	0xEE	43.2 ms	0.065 mA
	WLONG	0		
RGBC Init			2.40 ms	0.235 mA
RGBC ADC	ATIME	0xEE	43.2 ms	0.235 mA

Average I_{DD} Current = ((43.2 × 0.065) + (43.2 × 0.235) + (2.40 × 0.235)) / 89 ≈ 152 µA

Keeping with the same programmed values as the example, Table 2 shows how the average I_{DD} current is affected by the Wait state time, which is determined by WEN, WTIME, and WLONG. Note that the worst-case current occurs when the Wait state is not enabled.

Table 2. Average I_{DD} Current

WEN	WTIME	WLONG	WAIT STATE	AVERAGE I_{DD} CURRENT
0	n/a	n/a	0 ms	291 µA
1	0xFF	0	2.40 ms	280 µA
1	0xEE	0	43.2 ms	152 µA
1	0x00	0	614 ms	82 µA
1	0x00	1	7.37 s	67 µA

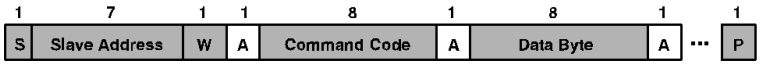
I²C Protocol

Interface and control are accomplished through an I²C serial compatible interface (standard or fast mode) to a set of registers that provide access to device control functions and output data. The devices support the 7-bit I²C addressing protocol.

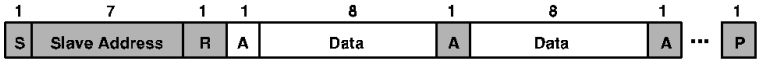
The I²C standard provides for three types of bus transaction: read, write, and a combined protocol (Figure 10). During a write operation, the first byte written is a command byte followed by data. In a combined protocol, the first byte written is the command byte followed by reading a series of bytes. If a read command is issued, the register address from the previous command will be used for data access. Likewise, if the MSB of the command is not set, the device will write a series of bytes at the address stored in the last valid command with a register address. The command byte contains either control information or a 5-bit register address. The control commands can also be used to clear interrupts.

The I²C bus protocol was developed by Philips (now NXP). For a complete description of the I²C protocol, please review the NXP I²C design specification at http://www.i2c–bus.org/references/.

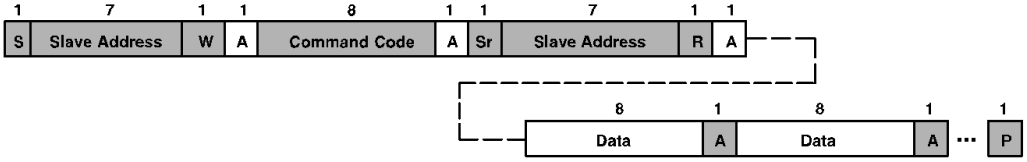
A Acknowledge (0)
N Not Acknowledged (1)
P Stop Condition
R Read (1)
S Start Condition
Sr Repeated Start Condition
W Write (0)
... Continuation of protocol
Master-to-Slave
Slave-to-Master



I²C Write Protocol



I²C Read Protocol



I²C Read Protocol — Combined Format

Figure 10. I²C Protocols

Wait Characteristics, V_{DD} = 3 V, T_A = 25°C, WEN = 1 (unless otherwise noted)

PARAMETER	TEST CONDITIONS	CHANNEL	MIN	TYP	MAX	UNIT
Wait step size	WTIME = 0xFF		2.27	2.4	2.56	ms
Wait number of integration steps (Note 1)			1		256	steps

NOTE 1: Parameter ensured by design and is not tested.

AC Electrical Characteristics, V_{DD} = 3 V, T_A = 25°C (unless otherwise noted)

PARAMETER†		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$f_{(SCL)}$	Clock frequency (I^2C only)		0		400	kHz
$t_{(BUF)}$	Bus free time between start and stop condition		1.3			µs
$t_{(HDSTA)}$	Hold time after (repeated) start condition. After this period, the first clock is generated.		0.6			µs
$t_{(SUSTA)}$	Repeated start condition setup time		0.6			µs
$t_{(SUSTO)}$	Stop condition setup time		0.6			µs
$t_{(HDDAT)}$	Data hold time		0			µs
$t_{(SUDAT)}$	Data setup time		100			ns
$t_{(LOW)}$	SCL clock low period		1.3			µs
$t_{(HIGH)}$	SCL clock high period		0.6			µs
t_F	Clock/data fall time				300	ns
t_R	Clock/data rise time				300	ns
C_i	Input pin capacitance				10	pF

† Specified by design and characterization; not production tested.

PARAMETER MEASUREMENT INFORMATION

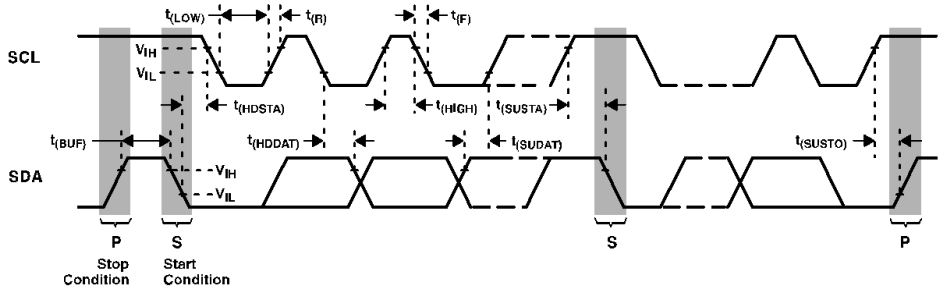


Figure 1. Timing Diagrams

TYPICAL CHARACTERISTICS

PHOTODIODE SPECTRAL RESPONSIVITY
RGBC

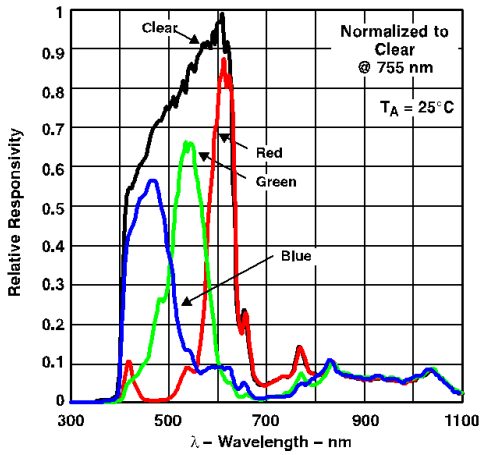


Figure 2

NORMALIZED RESPONSIVITY
vs.
ANGULAR DISPLACEMENT

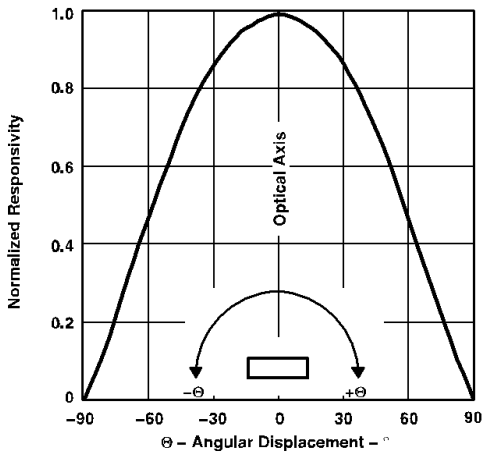


Figure 3

NORMALIZED I_{DD}
vs.
V_{DD} and TEMPERATURE

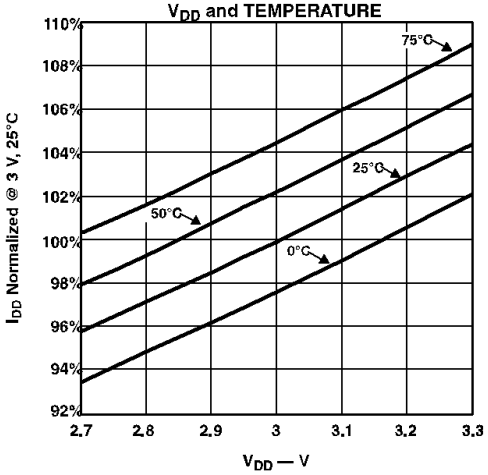


Figure 4

RESPONSIVITY TEMPERATURE
COEFFICIENT

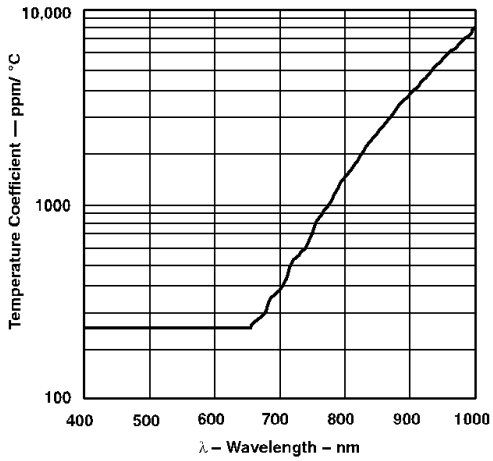


Figure 5

PRINCIPLES OF OPERATION

System States

An internal state machine provides system control of the RGBC and power management features of the device. At power up, an internal power-on-reset initializes the device and puts it in a low-power Sleep state.

When a start condition is detected on the I^2C bus, the device transitions to the Idle state where it checks the Enable Register (0x00) PON bit. If PON is disabled, the device will return to the Sleep state to save power. Otherwise, the device will remain in the Idle state until the RGBC function is enabled (AEN). Once enabled, the device will execute the Wait and RGBC states in sequence as indicated in Figure 5. Upon completion and return to Idle, the device will automatically begin a new Wait-RGBC cycle as long as PON and AEN remain enabled.

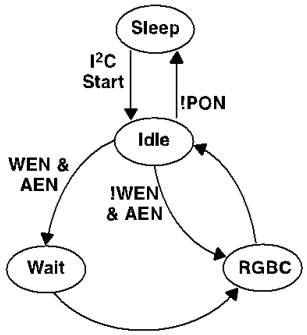


Figure 6. Simplified State Diagram

RGBC Operation

The RGBC engine contains RGBC gain control (AGAIN) and four integrating analog-to-digital converters (ADC) for the RGBC photodiodes. The RGBC integration time (ATIME) impacts both the resolution and the sensitivity of the RGBC reading. Integration of all four channels occurs simultaneously and upon completion of the conversion cycle, the results are transferred to the color data registers. This data is also referred to as channel *count*.

The transfers are double-buffered to ensure that invalid data is not read during the transfer. After the transfer, the device automatically moves to the next state in accordance with the configured state machine.

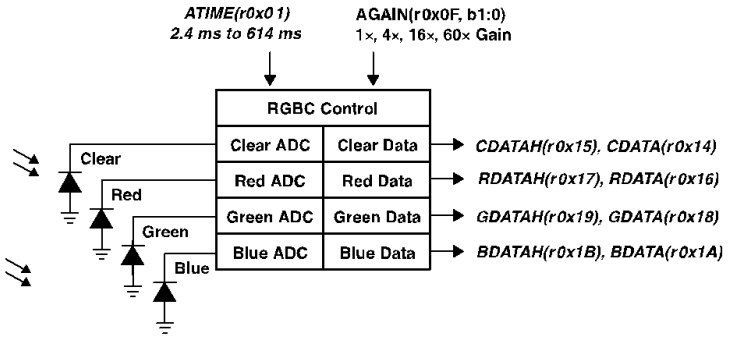


Figure 7. RGBC Operation

NOTE: In this document, the nomenclature uses the bit field name in italics followed by the register address and bit number to allow the user to easily identify the register and bit that controls the function. For example, the power on (PON) is in register 0x00, bit 0. This is represented as *PON (r0x00:b0)*.

The registers for programming the integration and wait times are a 2's compliment values. The actual time can be calculated as follows:

ATIME = 256 – Integration Time / 2.4 ms

Inversely, the time can be calculated from the register value as follows:

Integration Time = 2.4 ms × (256 – ATIME)

For example, if a 100-ms integration time is needed, the device needs to be programmed to:

256 – (100 / 2.4) = 256 – 42 = 214 = 0xD6

Conversely, the programmed value of 0xC0 would correspond to:

(256 – 0xC0) × 2.4 = 64 × 2.4 = 154 ms.

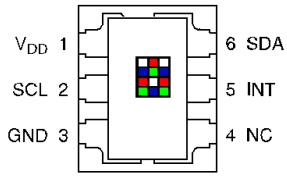
TCS3472
COLOR LIGHT-TO-DIGITAL CONVERTER with IR FILTER

TAOS135 – AUGUST 2012

Features

- Red, Green, Blue (RGB), and Clear Light Sensing with IR Blocking Filter
  - Programmable Analog Gain and Integration Time
  - 3,800,000:1 Dynamic Range
  - Very High Sensitivity — Ideally Suited for Operation Behind Dark Glass
- Maskable Interrupt
  - Programmable Upper and Lower Thresholds with Persistence Filter
- Power Management
  - Low Power — 2.5-μA Sleep State
  - 65-μA Wait State with Programmable Wait State Time from 2.4 ms to > 7 Seconds
- I^2C Fast Mode Compatible Interface
  - Data Rates up to 400 kbit/s
  - Input Voltage Levels Compatible with V_{DD} or 1.8 V Bus
- Register Set and Pin Compatible with the TCS3x71 Series
- Small 2 mm × 2.4 mm Dual Flat No-Lead (FN) Package

PACKAGE FN
DUAL FLAT NO-LEAD
(TOP VIEW)



Package Drawing Not to Scale

Applications

- RGB LED Backlight Control
- Light Color Temperature Measurement
- Ambient Light Sensing for Display Backlight Control
- Fluid and Gas Analysis
- Product Color Verification and Sorting

End Products and Market Segments

- TVs, Mobile Handsets, Tablets, Computers, and Monitors
- Consumer and Commercial Printing
- Medical and Health Fitness
- Solid State Lighting (SSL) and Digital Signage
- Industrial Automation

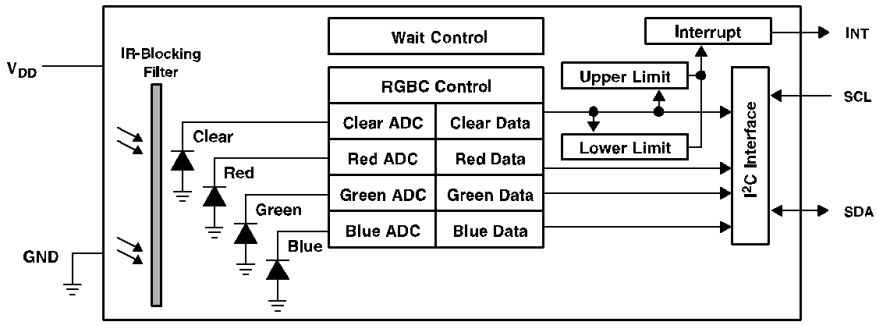
Description

The TCS3472 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials.

The TCS3472 color sensor has a wide range of applications including RGB LED backlight control, solid-state lighting, health/fitness products, industrial process controls and medical diagnostic equipment. In addition, the IR blocking filter enables the TCS3472 to perform ambient light sensing (ALS). Ambient light sensing is widely used in display-based products such as cell phones, notebooks, and TVs to sense the lighting environment and enable automatic display brightness for optimal viewing and power savings. The TCS3472, itself, can enter a lower-power wait state between light sensing measurements to further reduce the average power consumption.

Texas Advanced Optoelectronic Solutions Inc.
1001 Klein Road • Suite 300 • Plano, TX 75074 • (972) 673-0759
www.taosinc.com

Functional Block Diagram



Detailed Description

The TCS3472 light-to-digital converter contains a 3 × 4 photodiode array, four analog-to-digital converters (ADC) that integrate the photodiode current, data registers, a state machine, and an I^2C interface. The 3 × 4 photodiode array is composed of red-filtered, green-filtered, blue-filtered, and clear (unfiltered) photodiodes. In addition, the photodiodes are coated with an IR-blocking filter. The four integrating ADCs simultaneously convert the amplified photodiode currents to a 16-bit digital value. Upon completion of a conversion cycle, the results are transferred to the data registers, which are double-buffered to ensure the integrity of the data. All of the internal timing, as well as the low-power wait state, is controlled by the state machine.

Communication of the TCS3472 data is accomplished over a fast, up to 400 kHz, two-wire I^2C serial bus. The industry standard I^2C bus facilitates easy, direct connection to microcontrollers and embedded processors.

In addition to the I^2C bus, the TCS3472 provides a separate interrupt signal output. When interrupts are enabled, and user-defined thresholds are exceeded, the active-low interrupt is asserted and remains asserted until it is cleared by the controller. This interrupt feature simplifies and improves the efficiency of the system software by eliminating the need to poll the TCS3472. The user can define the upper and lower interrupt thresholds and apply an interrupt persistence filter. The interrupt persistence filter allows the user to define the number of consecutive out-of-threshold events necessary before generating an interrupt. The interrupt output is open-drain, so it can be wire-ORed with other devices.

Terminal Functions

TERMINAL NAME	TERMINAL NO.	TYPE	DESCRIPTION
GND	3		Power supply ground. All voltages are referenced to GND.
INT	5	O	Interrupt — open drain (active low).
NC	4	O	No connect — do not connect.
SCL	2	I	I^2C serial clock input terminal — clock signal for I^2C serial data.
SDA	6	I/O	I^2C serial data I/O terminal — serial data I/O for I^2C .
V_{DD}	1		Supply voltage.

Available Options

DEVICE	ADDRESS	PACKAGE – LEADS	INTERFACE DESCRIPTION	ORDERING NUMBER
TCS34721†	0x39	FN–6	I^2C Vbus = V_{DD} Interface	TCS34721FN
TCS34723†	0x39	FN–6	I^2C Vbus = 1.8 V Interface	TCS34723FN
TCS34725	0x29	FN–6	I^2C Vbus = V_{DD} Interface	TCS34725FN
TCS34727	0x29	FN–6	I^2C Vbus = 1.8 V Interface	TCS34727FN

† Contact TAOS for availability.

Absolute Maximum Ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{DD} (Note 1) ..... 3.8 V
Input terminal voltage ..... –0.5 V to 3.8 V
Output terminal voltage ..... –0.5 V to 3.8 V
Output terminal current ..... –1 mA to 20 mA
Storage temperature range, T_{stg} ..... –40°C to 85°C
ESD tolerance, human body model ..... 2000 V

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltages are with respect to GND.

Recommended Operating Conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{DD} (TCS34721 & TCS34725) (I^2C V_{bus} = V_{DD})	2.7	3	3.6	V
Supply voltage, V_{DD} (TCS34723 & TCS34727) (I^2C V_{bus} = 1.8 V)	2.7	3	3.3	V
Operating free-air temperature, T_A	–30		70	°C

Operating Characteristics, V_{DD} = 3 V, T_A = 25°C (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{DD}	Supply current	Active		235	330	µA
		Wait state		65		
		Sleep state — no I^2C activity		2.5	10	
V_{OL}	INT, SDA output low voltage	3 mA sink current	0		0.4	V
		6 mA sink current	0		0.6	
I_{LEAK}	Leakage current, SDA, SCL, INT pins		–5		5	µA
I_{LEAK}	Leakage current, LDR pin		–5		5	µA
V_{IH}	SCL. SDA input high voltage	TCS34721 & TCS34725	0.7 V_{DD}			V
		TCS34723 & TCS34727	1.25			
V_{IL}	SCL, SDA input low voltage	TCS34721 & TCS34725			0.3 V_{DD}	V
		TCS34723 & TCS34727			0.54	

Optical Characteristics, V_{DD} = 3 V, T_A = 25°C, AGAIN = 16×, ATIME = 0xF6 (unless otherwise noted) (Note 1)

PARAMETER		TEST CONDITIONS	Red Channel			Green Channel			Blue Channel			Clear Channel			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	MIN	TYP	MAX	
R_e	Irradiance responsivity	λ_D = 465 nm Note 2	0%		15%	10%		42%	65%		88%	11.0	13.8	16.6	counts/ µW/ cm^2
		λ_D = 525 nm Note 3	4%		25%	60%		85%	10%		45%	13.2	16.6	20.0	
		λ_D = 615 nm Note 4	80%		110%	0%		14%	5%		24%	15.6	19.5	23.4	

NOTES: 1. The percentage shown represents the ratio of the respective red, green, or blue channel value to the clear channel value.
2. The 465 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics: dominant wavelength λ_D = 465 nm, spectral halfwidth $\Delta\lambda_{1/2}$ = 22 nm.
3. The 525 nm input irradiance is supplied by an InGaN light-emitting diode with the following characteristics: dominant wavelength λ_D = 525 nm, spectral halfwidth $\Delta\lambda_{1/2}$ = 35 nm.
4. The 615 nm input irradiance is supplied by a AlInGaP light-emitting diode with the following characteristics: dominant wavelength λ_D = 615 nm, spectral halfwidth $\Delta\lambda_{1/2}$ = 15 nm.

RGBC Characteristics, VDD = 3 V, TA = 25°C, AGAIN = 16×, AEN = 1 (unless otherwise noted)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Dark ADC count value	E_e = 0, AGAIN = 60×, ATIME = 0xD6 (100 ms)	0	1	5	counts
ADC integration time step size	ATIME = 0xFF	2.27	2.4	2.56	ms
ADC number of integration steps (Note 5)		1		256	steps
ADC counts per step (Note 5)		0		1024	counts
ADC count value (Note 5)	ATIME = 0xC0 (153.6 ms)	0		65535	counts
Gain scaling, relative to 1× gain setting	4×	3.8	4	4.2	×
	16×	15	16	16.8	
	60×	58	60	63	

NOTE 5: Parameter ensured by design and is not tested.

TIP140, TIP141, TIP142, (NPN); TIP145, TIP146, TIP147, (PNP)

Darlington Complementary Silicon Power Transistors

Designed for general−purpose amplifier and low frequency switching applications.

Features

- High DC Current Gain −
  Min h_{FE} = 1000 @ I_C = 5.0 A, V_{CE} = 4 V
- Collector−Emitter Sustaining Voltage − @ 30 mA
  $V_{CEO(sus)}$ = 60 Vdc (Min) − TIP140, TIP145
  = 80 Vdc (Min) − TIP141, TIP146
  = 100 Vdc (Min) − TIP142, TIP147
- Monolithic Construction with Built−In Base−Emitter Shunt Resistor
- These are Pb−Free Devices*

MAXIMUM RATINGS

Rating	Symbol	TIP140 TIP145	TIP141 TIP146	TIP142 TIP147	Unit
Collector − Emitter Voltage	V_{CEO}	60	80	100	Vdc
Collector − Base Voltage	V_{CB}	60	80	100	Vdc
Emitter − Base Voltage	V_{EB}	5.0			Vdc
Collector Current − Continuous − Peak (Note 1)	I_C	10 15			Adc
Base Current − Continuous	I_B	0.5			Adc
Total Power Dissipation @ T_C = 25°C	P_D	125			W
Operating and Storage Junction Temperature Range	T_J, T_{stg}	−65 to +150			°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Max	Unit
Thermal Resistance, Junction−to−Case	$R_{\theta JC}$	1.0	°C/W
Thermal Resistance, Junction−to−Ambient	$R_{\theta JA}$	35.7	°C/W

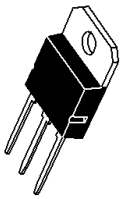
Stresses exceeding Maximum Ratings may damage the device. Maximum Ratings are stress ratings only. Functional operation above the Recommended Operating Conditions is not implied. Extended exposure to stresses above the Recommended Operating Conditions may affect device reliability.
1. 5 ms, ≤ 10% Duty Cycle.

*For additional information on our Pb−Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

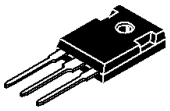
ON Semiconductor®

http://onsemi.com

10 AMPERE DARLINGTON COMPLEMENTARY SILICON POWER TRANSISTORS 60−100 VOLTS, 125 WATTS



SOT−93 (TO−218)
CASE 340D
STYLE 1



TO−247
CASE 340L
STYLE 3

NOTE: Effective June 2012 this device will be available only in the TO−247 package. Reference FPCN# 16827.

ORDERING INFORMATION

See detailed ordering and shipping information in the package dimensions section on page 2 of this data sheet.

Appendix B: Main Program Code

Main.c

```
/*
 * Main file for PIC microcontroller
 */

#include <xc.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>
#include "configBits.h"
#include "constants.h"
#include "lcd.h"
#include "I2C.h"
#include "macros.h"
#include "main.h"
#include "eeprom_routines.h"

void main(void) {

   // <editor-fold defaultstate="collapsed" desc=" STARTUP SEQUENCE ">

   //TRIS Sets Input/Output
   //0 = output
   //1 = input
   TRISA = 0b11111011;        //Set Port A as all input, except A2 for motor
   TRISB = 0xFF;              //Keypad
   TRISC = 0x00;              //RC3 and RC4 output for I2C (?)
```

```
TRISD = 0x00;            //All output mode for LCD
TRISE = 0x00;

LATA = 0x00;
LATB = 0x00;
LATC = 0x00;
LATD = 0x00;
LATE = 0x00;

ADCON0 = 0x00;           //Disable ADC
ADCON1 = 0xFF;           //Set PORTB to be digital instead of analog default

//ei();                  //Global Interrupt Mask
GIE = 1;
PEIE = 1;
INT1IE = 1;              //Enable KP interrupts
INT0IE = 0;              //Disable external interrupts
INT2IE = 0;

nRBPU = 0;

initLCD();
I2C_Master_Init(10000);    //Initialize I2C Master with 100KHz clock
I2C_ColorSens_Init();      //Initialize TCS34725 Color Sensor

//Set Timer Properties
TMR0 = 0;
T08BIT = 0;
T0CS = 0;
PSA = 0;
```

```
T0PS2 = 1;
T0PS1 = 1;
T0PS0 = 1;

TMR1 = 0;
servo0_flag = 0;
servo0_timer = 1;
T1CON = 0b10000001;
TMR1ON = 0;
TMR1CS = 0;
T1CKPS1 = 0;
T1CKPS0 = 0;
TMR1IE = 1;

TMR3 = 0;
servo1_flag = 0;
servo1_timer = 1;
T3CON = 0b1000001;
TMR3ON = 0;
TMR3CS = 0;
T3CKPS1 = 0;
T3CKPS0 = 0;
TMR3IE = 1;


//</editor-fold>
for(i=0;i<5;i++) bottle_count_disp[i] = -1;
for(i=20;i<50;i++) eeprom_writebyte(i, 0);

curr_state = STANDBY;
```

```
while(1){
   switch(curr_state){
      case STANDBY:
         standby();
         __delay_ms(500);
         break;
      case EMERGENCYSTOP:
         emergencystop();
         break;
      case OPERATION:
         operation();
         __delay_ms(2);
         __delay_us(400);
         break;
      case OPERATIONEND:
         operationend();
         __delay_ms(500);
         break;
      case DATETIME:
         date_time();
         __delay_ms(300);
         break;
      case BOTTLECOUNT:
         bottle_count();
         __delay_ms(300);
         break;
      case BOTTLECOUNT1:
         bottle_count1();
         __delay_ms(300);
```

```
            break;
        case BOTTLECOUNT2:
            bottle_count2();
            __delay_ms(300);
            break;
        case BOTTLECOUNT3:
            bottle_count3();
            __delay_ms(300);
            break;
        case BOTTLECOUNT4:
            bottle_count4();
            __delay_ms(300);
            break;
        case BOTTLETIME:
            bottle_time();
            __delay_ms(300);
            break;
      }
      //__delay_ms(MAINPOLLINGDELAYMS);
   }

   return;
}

void interrupt isr(void){
   if (INT1IF) {
      switch(PORTB>>4){
         case 0:    //KP_1 -- OPERATION START
            LATAbits.LATA2 = 1; //Start centrifuge motor
            TMR0IE = 1;        //Start timer with interrupts
```

```
            TMR0ON = 1;
            TMR0 = 0;
            TMR1ON = 1;
            TMR3ON = 1;
            operation_timeout = 0;

            read_time();
            start_time[1] = time[1];
            start_time[0] = time[0];
            for(i=0;i<5;i++){
               bottle_count_array[i] = 0;
               bottle_count_disp[i] = -1;
            }
            __lcd_clear();
            __delay_ms(100);
            __lcd_home();
            printf("running            ");

            curr_state = OPERATION;
            break;
        case 1:    //KP_2 -- BOTTLECOUNT
//             bottle_count_disp[0] += 1;
//             curr_state = BOTTLECOUNT;
            temp = bottle_count_disp[0];
            for(i=0;i<5;i++) bottle_count_disp[i] = -1;
            bottle_count_disp[0] = temp + 1;
            bottle_count_array[0] = eeprom_readbyte(20);
            bottle_count_array[1] = eeprom_readbyte(21);
            bottle_count_array[2] = eeprom_readbyte(22);
            bottle_count_array[3] = eeprom_readbyte(23);
```

```
    bottle_count_array[4] = eeprom_readbyte(24);
    curr_state = BOTTLECOUNT;
    while((PORTB>>4) == 1){}
    break;
case 2:    //KP_3
    operation_time = etime - stime;
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    curr_state = BOTTLETIME;
    break;
case 3:    //KP_A
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    curr_state = DATETIME;
    break;
case 4:    //KP_4
    temp = bottle_count_disp[1];
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    bottle_count_disp[1] = temp + 1;
    bottle_count_array[0] = eeprom_readbyte(25);
    bottle_count_array[1] = eeprom_readbyte(26);
    bottle_count_array[2] = eeprom_readbyte(27);
    bottle_count_array[3] = eeprom_readbyte(28);
    bottle_count_array[4] = eeprom_readbyte(29);
    curr_state = BOTTLECOUNT1;
    while((PORTB>>4) == 4){}
    break;
case 5:    //KP_5
    temp = bottle_count_disp[2];
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    bottle_count_disp[2] = temp + 1;
    bottle_count_array[0] = eeprom_readbyte(30);
```

```
    bottle_count_array[1] = eeprom_readbyte(31);
    bottle_count_array[2] = eeprom_readbyte(32);
    bottle_count_array[3] = eeprom_readbyte(33);
    bottle_count_array[4] = eeprom_readbyte(34);
    curr_state = BOTTLECOUNT2;
    while((PORTB>>4) == 5){}
    break;
case 6:    //KP_6
    temp = bottle_count_disp[3];
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    bottle_count_disp[3] = temp + 1;
    bottle_count_array[0] = eeprom_readbyte(35);
    bottle_count_array[1] = eeprom_readbyte(36);
    bottle_count_array[2] = eeprom_readbyte(37);
    bottle_count_array[3] = eeprom_readbyte(38);
    bottle_count_array[4] = eeprom_readbyte(39);
    curr_state = BOTTLECOUNT3;
    while((PORTB>>4) == 6){}
    break;
case 7:    //KP_B
    temp = bottle_count_disp[4];
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    bottle_count_disp[4] = temp + 1;
    bottle_count_array[0] = eeprom_readbyte(40);
    bottle_count_array[1] = eeprom_readbyte(41);
    bottle_count_array[2] = eeprom_readbyte(42);
    bottle_count_array[3] = eeprom_readbyte(43);
    bottle_count_array[4] = eeprom_readbyte(44);
    curr_state = BOTTLECOUNT4;
    while((PORTB>>4) == 7){}
```

```
    break;
case 8:    //KP_7
    LATAbits.LATA2 = 0; //Stop centrifuge motor
    TMR0IE = 0;        //Disable timer
    TMR0ON = 0;
    TMR1ON = 0;
    TMR3ON = 0;

    read_time();
    end_time[1] = time[1];
    end_time[0] = time[0];
    stime = 60*dec_to_hex(start_time[1])+dec_to_hex(start_time[0]);
    etime = 60*dec_to_hex(end_time[1])+dec_to_hex(end_time[0]);
    __lcd_clear();
    for(i=0;i<5;i++) bottle_count_disp[i] = -1;
    savedata();
    curr_state = OPERATIONEND;
    break;
case 9:    //KP_8 -- TESTING
    read_colorsensor();
    __lcd_home();
    printf("C%u R%u              ", color[0], color[1]);
    __lcd_newline();
    printf("G%u B%u              ", color[2], color[3]);
    break;
case 12:   //KP_*
    LATAbits.LATA2 = 0; //Stop centrifuge motor
    di();              //Disable all interrupts
    TMR0ON = 0;
    __lcd_clear();
```

```
            curr_state = EMERGENCYSTOP;
            break;
        case 14:   //KP_#
            for(i=0;i<5;i++) bottle_count_disp[i] = -1;
            curr_state = STANDBY;
            break;
        case 10:   //KP_9 -- TESTING
            //set_time();
            break;
        case 11:   //KP_C -- TESTING
            //savedata();
            break;
    }
    INT1IF = 0;
}
else if (TMR1IF){
    if(servo0_flag){
        LATCbits.LATC0 = 0;
        TMR1 = 16517;
        servo0_flag = 0;
    }
    else{
        LATCbits.LATC0 = 1;
        if(servo0_timer) TMR1 = 63000;
        else TMR1 = 62000;
        servo0_flag = 1;
    }
    TMR1IF = 0;
}
else if (TMR3IF){
```

```
    if(servo1_flag){
        LATCbits.LATC1 = 0;
        TMR3 = 16517;
        servo1_flag = 0;
    }
    else{
        LATCbits.LATC1 = 1;
        if(servo1_timer) TMR3 = 62000;
        else TMR3 = 63000;
        servo1_flag = 1;
    }
    TMR3IF = 0;
}
else if (TMR0IF){
    if(operation_timeout > 2){
        LATAbits.LATA2 = 0; //Stop centrifuge motor
        TMR0IE = 0;         //Disable timer
        TMR0ON = 0;
        TMR1ON = 0;
        TMR3ON = 0;

        read_time();
        end_time[1] = time[1];
        end_time[0] = time[0];
        stime = 60*dec_to_hex(start_time[1])+dec_to_hex(start_time[0]);
        etime = 60*dec_to_hex(end_time[1])+dec_to_hex(end_time[0]);
        __lcd_clear();
        for(i=0;i<5;i++) bottle_count_disp[i] = -1;
        savedata();
        curr_state = OPERATIONEND;
```

```
        }
        else operation_timeout += 1;
        TMR0IF = 0;
    }
    else{
        while(1){
            __lcd_home();
            printf("ERR: BAD ISR");
            __delay_1s();
        }
    }
    return;
}

void standby(void){
    __lcd_home();
    printf("standby         ");
    __lcd_newline();
    read_colorsensor();
    printf("%d      ", color[0]);
    return;
}

void set_time(void){
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(0b11010000); //7 bit RTC address + Write
    I2C_Master_Write(0x00); //Set memory pointer to seconds
    for(char i=0; i<7; i++){
        I2C_Master_Write(timeset[i]);
    }
```

```
  I2C_Master_Stop(); //Stop condition
}

int dec_to_hex(int num) {              //Convert decimal unsigned char to hexadecimal int
  int i = 0, quotient = num, temp, hexnum = 0;

  while (quotient != 0) {
    temp = quotient % 16;
    hexnum += temp*pow(10,i);
    quotient = quotient / 16;
    i += 1;
  }
  return hexnum;
}

void date_time(void){
  //Reset RTC memory pointer
  I2C_Master_Start(); //Start condition
  I2C_Master_Write(0b11010000); //7 bit RTC address + Write
  I2C_Master_Write(0x00); //Set memory pointer to seconds
  I2C_Master_Stop(); //Stop condition

  //Read Current Time
  I2C_Master_Start();
  I2C_Master_Write(0b11010001); //7 bit RTC address + Read
  for(unsigned char i=0;i<0x06;i++){
    time[i] = I2C_Master_Read(1);
  }
  time[6] = I2C_Master_Read(0);      //Final Read without ack
  I2C_Master_Stop();
```

```
    //LCD Display
    __lcd_home();
    printf("Date: %02x/%02x/%02x  ", time[5],time[4],time[6]);    //Print date in MM/DD/YY
    __lcd_newline();
    printf("Time: %02x:%02x:%02x  ", time[2],time[1],time[0]);    //HH:MM:SS

    return;
}

void read_time(void){
    //Reset RTC memory pointer
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(0b11010000); //7 bit RTC address + Write
    I2C_Master_Write(0x00); //Set memory pointer to seconds
    I2C_Master_Stop(); //Stop condition

    //Read Current Time
    I2C_Master_Start();
    I2C_Master_Write(0b11010001); //7 bit RTC address + Read
    for(unsigned char i=0;i<0x06;i++){
        time[i] = I2C_Master_Read(1);   //Read with ack
    }
    time[6] = I2C_Master_Read(0);       //Final Read without ack
    I2C_Master_Stop();
    return;
}

void bottle_count(void){
    switch(bottle_count_disp[0] % 3){
```

```
        case 0:
          __lcd_home();
          printf("Bottle Count        ");
          __lcd_newline();
          printf("Total: %d           ", bottle_count_array[0]);
          break;
        case 1:
          __lcd_home();
          printf("YOP W/ CAP: %d  ", bottle_count_array[1]);
          __lcd_newline();
          printf("YOP NO CAP: %d  ", bottle_count_array[2]);
          break;
        case 2:
          __lcd_home();
          printf("ESKA W/ CAP: %d ", bottle_count_array[3]);
          __lcd_newline();
          printf("ESKA NO CAP: %d ", bottle_count_array[4]);
          break;
        default:
          while(1){
            __lcd_home();
            printf("ERR: BAD BTLCNT");
          }
          break;
    }
    return;
}

void bottle_count1(void){
    switch(bottle_count_disp[1] % 3){
```

```
        case 0:
          __lcd_home();
          printf("BttlCnt Prev 1       ");
          __lcd_newline();
          printf("Total: %d          ", bottle_count_array[0]);
          break;
        case 1:
          __lcd_home();
          printf("YOP W/ CAP: %d  ", bottle_count_array[1]);
          __lcd_newline();
          printf("YOP NO CAP: %d  ", bottle_count_array[2]);
          break;
        case 2:
          __lcd_home();
          printf("ESKA W/ CAP: %d ", bottle_count_array[3]);
          __lcd_newline();
          printf("ESKA NO CAP: %d ", bottle_count_array[4]);
          break;
        default:
          while(1){
            __lcd_home();
            printf("ERR: BAD BTLCNT");
          }
          break;
    }
    return;
}

void bottle_count2(void){
    switch(bottle_count_disp[2] % 3){
```

```
        case 0:
          __lcd_home();
          printf("BttlCnt Prev 2        ");
          __lcd_newline();
          printf("Total: %d           ", bottle_count_array[0]);
          break;
        case 1:
          __lcd_home();
          printf("YOP W/ CAP: %d  ", bottle_count_array[1]);
          __lcd_newline();
          printf("YOP NO CAP: %d  ", bottle_count_array[2]);
          break;
        case 2:
          __lcd_home();
          printf("ESKA W/ CAP: %d ", bottle_count_array[3]);
          __lcd_newline();
          printf("ESKA NO CAP: %d ", bottle_count_array[4]);
          break;
        default:
          while(1){
            __lcd_home();
            printf("ERR: BAD BTLCNT");
          }
          break;
    }
    return;
}

void bottle_count3(void){
    switch(bottle_count_disp[3] % 3){
```

```
        case 0:
          __lcd_home();
          printf("BttlCnt Prev 3        ");
          __lcd_newline();
          printf("Total: %d           ", bottle_count_array[0]);
          break;
        case 1:
          __lcd_home();
          printf("YOP W/ CAP: %d  ", bottle_count_array[1]);
          __lcd_newline();
          printf("YOP NO CAP: %d  ", bottle_count_array[2]);
          break;
        case 2:
          __lcd_home();
          printf("ESKA W/ CAP: %d ", bottle_count_array[3]);
          __lcd_newline();
          printf("ESKA NO CAP: %d ", bottle_count_array[4]);
          break;
        default:
          while(1){
            __lcd_home();
            printf("ERR: BAD BTLCNT");
          }
          break;
    }
    return;
}

void bottle_count4(void){
    switch(bottle_count_disp[4] % 3){
```

```
      case 0:
         __lcd_home();
         printf("BttlCnt Prev 4         ");
         __lcd_newline();
         printf("Total: %d          ", bottle_count_array[0]);
         break;
      case 1:
         __lcd_home();
         printf("YOP W/ CAP: %d  ", bottle_count_array[1]);
         __lcd_newline();
         printf("YOP NO CAP: %d  ", bottle_count_array[2]);
         break;
      case 2:
         __lcd_home();
         printf("ESKA W/ CAP: %d ", bottle_count_array[3]);
         __lcd_newline();
         printf("ESKA NO CAP: %d ", bottle_count_array[4]);
         break;
      default:
         while(1){
            __lcd_home();
            printf("ERR: BAD BTLCNT");
         }
         break;
   }
   return;
}

void bottle_time(void){
   __lcd_home();
```

```
    printf("Total Operation         ");
    __lcd_newline();
    printf("Time: %d s            ", operation_time);
    return;
}

void operation(void){
    if(bottle_count_array[0] > 9){
        __delay_ms(1000);
        LATAbits.LATA2 = 0; //Stop centrifuge motor
        TMR0IE = 0;        //Disable timer
        TMR0ON = 0;
        TMR1ON = 0;
        TMR3ON = 0;

        read_time();
        end_time[1] = time[1];
        end_time[0] = time[0];
        stime = 60*dec_to_hex(start_time[1])+dec_to_hex(start_time[0]);
        etime = 60*dec_to_hex(end_time[1])+dec_to_hex(end_time[0]);
        curr_state = OPERATIONEND;
        __lcd_clear();
        for(i=0;i<5;i++) bottle_count_disp[i] = -1;
        savedata();
        return;
    }
    colorprev[0] = color[0];
    colorprev[1] = color[1];
    colorprev[2] = color[2];
    colorprev[3] = color[3];
```

```
    GIE = 0;
    read_colorsensor();
    if(color[0]>AMBIENTTCSCLEAR){
      flag_bottle = 1;
      flag_picbug += 1;
      if(color[3]>color[1] && !flag_top_read) flag_eskaC += 1;
      if(color[1]>NOCAPDISTINGUISH || color[2]>NOCAPDISTINGUISH)flag_yopNC = 1;
      if(color[0]>TCSBOTTLEHIGH){
        if(!flag_top_read){
          r = (float) color[1];
          b = (float) color[3];
//          __lcd_home();
//          printf("%u, %u, %u,     ", color[1], color[2], color[3]);
          if(r/b > 2 && r>16) bottle_read_top = 1;
          else if(r/b < 0.75) bottle_read_top = 2;
          else bottle_read_top = 0;
          flag_top_read = 1;
        }     //FOR FINAL REPORT SIMPLICITY REMOVE CERTAIN MINOR CODE OPTIMIZATIONS
        flag_bottle_high = 1;
      }
      else if(color[0]<TCSBOTTLEHIGH){
        if(flag_bottle_high){
          r_p = (float) colorprev[1];
          b_p = (float) colorprev[3];
          if(r_p/b_p > 3.2 && r_p>18) bottle_read_bot = 1;
          else if(r_p/b_p < 0.75) bottle_read_bot = 2;
          else bottle_read_bot = 0;
          flag_bottle_high = 0;
        }
```

```
        }
    }
    else if(flag_bottle && flag_picbug > 20){
        flag_picbug = 0;
        bottle_count_array[0] += 1;
        TMR0 = 0;
        if(bottle_read_top == 2 || bottle_read_bot == 2 || flag_eskaC>1){
            bottle_count_array[3] += 1;
            servo1_timer = 1;
        }
        else if(bottle_read_top == 1 || bottle_read_bot == 1){
            bottle_count_array[1] += 1;
            servo0_timer = 1;
        }
        else if(flag_yopNC){
            bottle_count_array[2] += 1;
            servo0_timer = 0;
        }
        else{
            bottle_count_array[4] += 1;
            servo1_timer = 0;
        }
        flag_bottle = 0;
        flag_bottle_high = 0;
        flag_top_read = 0;
        flag_yopNC = 0;
//      __lcd_home();
//      __lcd_newline();   //TESTING
//      printf("%d      ", flag_eskaC);
        flag_eskaC = 0;
```

```
//      printf("%d, %d, %d", color[1], color[2], color[3]);
//      printf("%f", r_p/b_p);
//      printf("%d, %d, %d", bottle_count_array[0], bottle_read_top, bottle_read_bot);
   }
   else if(flag_picbug < 3 && flag_picbug > 0) flag_picbug -= 1;
   GIE  = 1;
   return;
}

void operationend(void){
   __lcd_home();
   printf("Operation Done!         ");
   return;
}

void emergencystop(void){
   di();
   PORTAbits.RA2 = 0;
   __lcd_clear();
   __lcd_home();
   printf("EMERGENCY STOP          ");
   while(1){}
   return;
}

void servo_rotate0(int degree){        //depreciated
   int duty = degree;
   for (i=0; i<20; i++) {
      LATCbits.LATC0 = 1;
```

```
        for(j=0; j<duty; j++) __delay_us(100);
        LATCbits.LATC0 = 0;
        for(j=0; j<(200 - duty); j++) __delay_us(100);
    }
    return;
}

void servo_rotate1(int degree){
    int duty = degree;
    for (i=0; i<20; i++) {
        LATCbits.LATC1 = 1;
        for(j=0; j<duty; j++) __delay_us(100);
        LATCbits.LATC1 = 0;
        for(j=0; j<(200 - duty); j++) __delay_us(100);
    }
    return;
}

void read_colorsensor(void){
//Reading Color
//    I2C_Master_Start();
//    I2C_Master_Write(0b01010010);   //7bit address 0x29 + Write
//    I2C_Master_Write(0b10110100);   //Write to cmdreg + access&increment clear low reg
//    I2C_Master_Start();     //Repeated start command for combined I2C
//    I2C_Master_Write(0b01010011);   //7bit address 0x29 + Read
//    for(i=0; i<3; i++){
//        color_low[i] = I2C_Master_Read(1); //Reading with acknowledge, continuous
//        color_high[i] = I2C_Master_Read(1);
//    }
//    color_low[3] = I2C_Master_Read(1);
```

```
//    color_high[3] = I2C_Master_Read(0);    //Final read, no ack
//    I2C_Master_Stop();             //Stop condition
//
//    for(i=0; i<4; i++){
//        color[i] = (color_high[i] << 8)|(color_low[i]);
//    }

    I2C_Master_Start();               //Repeated start command for combined I2C
    I2C_Master_Write(0b01010011);       //7bit address 0x29 + Read
    color_low[0] = I2C_Master_Read(1);  //Reading clear with acknowledge, continuous
    color_high[0] = I2C_Master_Read(1);
    color_low[1] = I2C_Master_Read(1);  //Reading red with acknowledge, continuous
    color_high[1] = I2C_Master_Read(1);
    color_low[2] = I2C_Master_Read(1);  //Reading green with acknowledge, continuous
    color_high[2] = I2C_Master_Read(1);
    color_low[3] = I2C_Master_Read(1);
    color_high[3] = I2C_Master_Read(0); //Final read for blue, no ack
    I2C_Master_Stop();                //Stop condition
    color[0] = (color_high[0] << 8)|(color_low[0]);
    color[1] = (color_high[1] << 8)|(color_low[1]);
    color[2] = (color_high[2] << 8)|(color_low[2]);
    color[3] = (color_high[3] << 8)|(color_low[3]);

    return;
}

uint8_t eeprom_readbyte(uint16_t address) {

    // Set address registers
    EEADRH = (uint8_t)(address >> 8);
```

```
    EEADR = (uint8_t)address;

    EECON1bits.EEPGD = 0;      // Select EEPROM Data Memory
    EECON1bits.CFGS = 0;       // Access flash/EEPROM NOT config. registers
    EECON1bits.RD = 1;         // Start a read cycle

    // A read should only take one cycle, and then the hardware will clear
    // the RD bit
    while(EECON1bits.RD == 1);

    return EEDATA;             // Return data
}

void eeprom_writebyte(uint16_t address, uint8_t data) {
    // Set address registers
    EEADRH = (uint8_t)(address >> 8);
    EEADR = (uint8_t)address;

    EEDATA = data;         // Write data we want to write to SFR
    EECON1bits.EEPGD = 0;   // Select EEPROM data memory
    EECON1bits.CFGS = 0;    // Access flash/EEPROM NOT config. registers
    EECON1bits.WREN = 1;    // Enable writing of EEPROM (this is disabled again after the write completes)

    // The next three lines of code perform the required operations to
    // initiate a EEPROM write
    EECON2 = 0x55;         // Part of required sequence for write to internal EEPROM
    EECON2 = 0xAA;         // Part of required sequence for write to internal EEPROM
    EECON1bits.WR = 1;      // Part of required sequence for write to internal EEPROM

    // Loop until write operation is complete
```

```
   while(PIR2bits.EEIF == 0)
   {
      continue;   // Do nothing, are just waiting
   }

   PIR2bits.EEIF = 0;      //Clearing EEIF bit (this MUST be cleared in software after each write)
   EECON1bits.WREN = 0;    // Disable write (for safety, it is re-enabled next time a EEPROM write is performed)
}

void savedata(void) {
//    for(i=19;i--;i>=0){
//       eeprom_writebyte(25+i, eeprom_readbyte(20+i));
//    }
//    for(i=0;i++;i<5){
//       eeprom_writebyte((20+i), bottle_count_array[i]);
//    }
   eeprom_writebyte(44, eeprom_readbyte(39));
   eeprom_writebyte(43, eeprom_readbyte(38));
   eeprom_writebyte(42, eeprom_readbyte(37));
   eeprom_writebyte(41, eeprom_readbyte(36));
   eeprom_writebyte(40, eeprom_readbyte(35));

   eeprom_writebyte(39, eeprom_readbyte(34));
   eeprom_writebyte(38, eeprom_readbyte(33));
   eeprom_writebyte(37, eeprom_readbyte(32));
   eeprom_writebyte(36, eeprom_readbyte(31));
   eeprom_writebyte(35, eeprom_readbyte(30));

   eeprom_writebyte(34, eeprom_readbyte(29));
```

```
    eeprom_writebyte(33, eeprom_readbyte(28));
    eeprom_writebyte(32, eeprom_readbyte(27));
    eeprom_writebyte(31, eeprom_readbyte(26));
    eeprom_writebyte(30, eeprom_readbyte(25));

    eeprom_writebyte(29, eeprom_readbyte(34));
    eeprom_writebyte(28, eeprom_readbyte(23));
    eeprom_writebyte(27, eeprom_readbyte(22));
    eeprom_writebyte(26, eeprom_readbyte(21));
    eeprom_writebyte(25, eeprom_readbyte(20));

    eeprom_writebyte(24, bottle_count_array[4]);
    eeprom_writebyte(23, bottle_count_array[3]);
    eeprom_writebyte(22, bottle_count_array[2]);
    eeprom_writebyte(21, bottle_count_array[1]);
    eeprom_writebyte(20, bottle_count_array[0]);
}
```

Main.h

```
#ifndef MAIN_H
#define  MAIN_H
//FUNCTIONS
void set_time(void);
int dec_to_hex(int num);
void date_time(void);
void read_time(void);
void bottle_count(void);
void bottle_count1(void);
void bottle_count2(void);
void bottle_count3(void);
void bottle_count4(void);
void bottle_time(void);
void standby(void);
void operation(void);
void operationend(void);
void emergencystop(void);
void servo_rotate0(int degree);
void servo_rotate1(int degree);
void read_colorsensor(void);
uint8_t eeprom_readbyte(uint16_t);
void eeprom_writebyte(uint16_t, uint8_t);
void savedata(void);
//VARIABLES
int i;
int j;
const char keys[] = "123A456B789C*0#D";
const char timeset[7] = {   0x30, //Seconds
```

```
                0x19, //Minutes
                0x13, //Hour, 24 hour mode
                0x02, //Day of the week, Monday = 1
                0x11, //Day/Date
                0x04, //Month
                0x17};//Year, last two digits

enum state {
      STANDBY,
      EMERGENCYSTOP,
      OPERATION,
      OPERATIONEND,
      DATETIME,
      BOTTLECOUNT,
      BOTTLECOUNT1,
      BOTTLECOUNT2,
      BOTTLECOUNT3,
      BOTTLECOUNT4,
      BOTTLETIME
   };
enum state curr_state;

unsigned char time[7];
unsigned char start_time[2];
unsigned char end_time[2];
int stime;
int etime;
int operation_time;
int temp;
```

```
//For bottle count
//0 = Total
//1 = YOP + CAP
//2 = YOP - CAP
//3 = ESKA + CAP
//4 = ESKA - CAP
int bottle_count_disp[5] = -1; //Data for bottle display screen
int bottle_count_array[5];

int operation_disp = 0;        //Data for operation running animation
int operation_timeout = 0;
unsigned int color[4];         //Stores TCS data in form clear, red, green, blue
unsigned int colorprev[4];
unsigned char color_low[4];    //For reading colors
unsigned char color_high[4];

int servo0_timer;      //1250 = 1ms
int servo1_timer;
char servo0_flag;
char servo1_flag;

//Bottle Detection Logic
int flag_bottle;
int flag_bottle_high;
int flag_top_read;
int flag_yopNC;
int flag_picbug;
int flag_eskaC;
```

```
int bottle_read_top;

int bottle_read_bot;

float r, b, r_p, b_p;


//CONSTANTS

#define MAINPOLLINGDELAYMS  10

#define AMBIENTTCSCLEAR     18

#define TCSBOTTLEHIGH       30

#define NOCAPDISTINGUISH    130


#endif    /* MAIN_H */
```

I2c.c

```
/*
 * File:   I2C.c
 * Author: Administrator
 *
 * Created on August 4, 2016, 3:22 PM
 */


#include <xc.h>
#include "I2C.h"
#include "configBits.h"
#include "constants.h"

void I2C_Master_Init(const unsigned long c)
{
  // See Datasheet pg171, I2C mode configuration
  SSPSTAT = 0b00000000;
  SSPCON1 = 0b00101000;
  SSPCON2 = 0b00000000;
  SSPADD = (_XTAL_FREQ/(4*c))-1;
  TRISC3 = 1;       //Setting as input as given in datasheet
  TRISC4 = 1;       //Setting as input as given in datasheet
}

void I2C_Master_Wait()
{
  while ((SSPSTAT & 0x04) || (SSPCON2 & 0x1F));
}
```

```
void I2C_Master_Start()
{
  I2C_Master_Wait();
  SEN = 1;
}

void I2C_Master_RepeatedStart()
{
  I2C_Master_Wait();
  RSEN = 1;
}

void I2C_Master_Stop()
{
  I2C_Master_Wait();
  PEN = 1;
}

void I2C_Master_Write(unsigned d)
{
  I2C_Master_Wait();
  SSPBUF = d;
}

void I2C_ColorSens_Init(void){
    I2C_Master_Start();           //Write Start condition
    I2C_Master_Write(0b01010010);   //7bit address for TCS (0x29) + Write
    I2C_Master_Write(0b10000000);   //Write to cmdreg + access enable reg
    I2C_Master_Write(0b00000001);   //Start POWER
```

```
    I2C_Master_Stop();

    __delay_ms(3);              //TCS requires 2.4ms delay before other actions

    I2C_Master_Start();         //Write Start condition
    I2C_Master_Write(0b01010010);   //7bit address for TCS (0x29) + Write
    I2C_Master_Write(0b10000000);   //Write to cmdreg + access enable reg
    I2C_Master_Write(0b00000011);   //Start RGBC
    I2C_Master_Stop();

    I2C_Master_Start();         //Write Start condition
    I2C_Master_Write(0b01010010);   //7bit address for TCS (0x29) + Write
    I2C_Master_Write(0b10001111);   //Write to cmdreg + access control reg
    I2C_Master_Write(0b00000010);   //Set analog gain to 16
    I2C_Master_Stop();

    I2C_Master_Start();         //Write Start condition
    I2C_Master_Write(0b01010010);   //7bit address for TCS (0x29) + Write
    I2C_Master_Write(0b10000001);   //Write to cmdreg + access RGBC timing register
    I2C_Master_Write(0b11111111);   //Set RGBC timing register
    I2C_Master_Stop();

    I2C_Master_Start();         //Initializing for reading
    I2C_Master_Write(0b01010010);   //7bit address 0x29 + Write
    I2C_Master_Write(0b10110100);   //Write to cmdreg + access&increment clear low reg
    I2C_Master_Stop();
}

void I2C_ColorSens_ClearInt(void){
    I2C_Master_Start();         //Write Start condition
```

```
    I2C_Master_Write(0b01010010);   //7bit address for TCS (0x29) + Write
    I2C_Master_Write(0b11100110);   //Write to cmdreg + special func clear int
    I2C_Master_Write(0b11100110);   //TESTING -- works but not sure if needed
    I2C_Master_Stop();
}

unsigned char I2C_Master_Read(unsigned char a)
{
  unsigned char temp;
  I2C_Master_Wait();
  RCEN = 1;
  I2C_Master_Wait();
  temp = SSPBUF;
  I2C_Master_Wait();
  ACKDT = (a)?0:1;
  ACKEN = 1;
  return temp;
}

void delay_10ms(unsigned char n) {
    while (n-- != 0) {
        __delay_ms(5);
    }
}
```

I2c.h

```
void I2C_Master_Init(const unsigned long c);

//void I2C_ColorSens_Init(void);

void I2C_Master_Write(unsigned d);

unsigned char I2C_Master_Read(unsigned char a);

void delay_10ms(unsigned char n);
```

Lcd.c

```
/*
 * File:   lcd.c
 * Author: True Administrator
 *
 * Created on July 18, 2016, 12:11 PM
 */

#include <xc.h>
#include "configBits.h"
#include <stdio.h>
#include "lcd.h"
#include "constants.h"

void initLCD(void) {
    __delay_ms(15);
    lcdInst(0b00110011);       //Force into 8bit mode
    lcdInst(0b00110011);       //Should require only three commands, but
    lcdInst(0b00110010);       //Seems to be demanding five in this case.
    lcdInst(0b00101000);
    lcdInst(0b00001111);
    lcdInst(0b00000110);
    lcdInst(0b00000001);
    __delay_ms(15);
}

void lcdInst(char data) {
    RS = 0;
    lcdNibble(data);
}
```

```
void putch(char data){
   RS = 1;
   lcdNibble(data);
}

void lcdNibble(char data){
   // Send of 4 most sig bits, then the 4 least sig bits (MSD,LSD)
   char temp = data & 0xF0;
   LATD = LATD & 0x0F;
   LATD = temp | LATD;

   E = 0;
   __delay_us(LCD_DELAY);
   E = 1;
   __delay_us(LCD_DELAY);

   data = data << 4;

   temp = data & 0xF0;
   LATD = LATD & 0x0F;
   LATD = temp | LATD;

   E = 0;
   __delay_us(LCD_DELAY);
   E = 1;
   __delay_us(LCD_DELAY);
}
```

Lcd.h

```
void lcdInst(char data);

void lcdNibble(char data);

void initLCD(void);
```

eeprom_routines.h

```
// This header file should not be included directly
// Inclusion of this file is provided indirectly by including htc.h

/*********************************************************************/
/****** EEPROM memory read/write macros and function definitions *******/
/*********************************************************************/

#if _EEPROM_INT == _EEREG_INT
#ifdef __FLASHTYPE
#define _ADJ_EECON1()  EECON1 &= 0x3F
#else
#define _ADJ_EECON1()  (void)0
#endif
#endif

/*
 * EEPROM_WRITE Implementations
 */

#define _EEREG_EEPROM_WRITE(addr, value)      \
do{                                                                     \
        while (WR) {                                            \
                continue;                                               \
        }                                                               \
        EEADR = (addr);                                         \
        EEDATA = (value);                                       \
        _ADJ_EECON1();                                          \
        CARRY = 0;                                                      \
        if (GIE) {                                              \
```

```
        CARRY = 1;                                              \
    }                                                           \
    GIE = 0;                                                    \
    WREN = 1;                                                   \
    EECON2 = 0x55;                                              \
    EECON2 = 0xAA;                                              \
    WR = 1;                                                     \
    WREN = 0;                                                   \
    if (CARRY) {                                                \
        GIE = 1;                                                \
    }                                                           \
} while (0)

#define _NVMREG_EEPROM_WRITE(addr, value)                       \
do {                                                            \
    while (NVMCON1bits.WR) {                                    \
        continue;                                               \
    }                                                           \
    NVMCON1bits.NVMREGS = 1;                                    \
    NVMADRL = (addr) & 0xFF;                                    \
    NVMADRH = 0x70;                                             \
    NVMDATH = 0;                                                \
    NVMDATL = (value) & 0xFF;                                   \
    STATUSbits.CARRY = 0;                                       \
    if (INTCONbits.GIE) {                                       \
        STATUSbits.CARRY = 1;                                   \
```

```
        }
                \
        INTCONbits.GIE = 0;
        \
        NVMCON1bits.WREN = 1;
        \
        NVMCON2 = 0x55;
        \
        NVMCON2 = 0xAA;
        \
        NVMCON1bits.WR = 1;
        \
        while (NVMCON1bits.WR) {                                                \
                continue;
        \
        }
                \
        NVMCON1bits.WREN = 0;
        \
        if (STATUSbits.CARRY) {                                                 \
                INTCONbits.GIE = 1;
        \
        }
                \
}
                \
while (0)


/*
 * EEPROM_READ Implementations
 *
 * NOTE WELL:
 * The macro EEPROM_READ() is NOT safe to use immediately after any
 * write to EEPROM, as it does NOT wait for WR to clear.  This is by
```

```
 * design, to allow minimal code size if a sequence of reads is
 * desired.  To guarantee uncorrupted writes, use the function
 * eeprom_read() or insert
 *     while(WR)continue;
 * before calling EEPROM_READ().
*/

#define _EEREG_EEPROM_READ(addr) (   \
                    EEADR = addr,                           \
                    _ADJ_EECON1(),                          \
                    RD = 1,                                         \
                    EEDATA)

#define _NVMREG_EEPROM_READ(addr) (         \
                    NVMCON1bits.NVMREGS = 1,    \
                    NVMADRL = (addr) & 0xFF,        \
                    NVMADRH = 0x70,                         \
                    NVMCON1bits.RD = 1,                     \
                    NVMDATL)

//
// General Macro Functions
//

#if EEPROM_SIZE > 0
#if _EEPROM_INT == _EEREG_INT
#define EEPROM_WRITE(addr, value)       _EEREG_EEPROM_WRITE(addr, value)
#define EEPROM_READ(addr)                       _EEREG_EEPROM_READ(addr)
#elif _EEPROM_INT == _NVMREG_INT
#define EEPROM_WRITE(addr, value)       _NVMREG_EEPROM_WRITE(addr, value)
```

```
#define EEPROM_READ(addr)                       _NVMREG_EEPROM_READ(addr)
#else
#error "Unknonwn EEPROM register interface"
#endif

/* library function versions */
extern void eeprom_write(unsigned char addr, unsigned char value);
extern unsigned char eeprom_read(unsigned char addr);
extern void eecpymem(volatile unsigned char *to, __eeprom unsigned char *from, unsigned char size);
extern void memcpyee(__eeprom unsigned char *to, const unsigned char *from, unsigned char size);
#endif    // end EEPROM routines
```

Configbits.h

```
/*
 * PIC18F4620 Configuration Bits
 * To generate this configuration, Window->PIC Memory Views->Configuration Bits
 * and select from the options provided. You can then generate the source and
 * paste it into a header file like this one.
 */

// CONFIG1H
#pragma config OSC = HS        // Oscillator Selection bits (HS oscillator)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config PWRT = OFF       // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))
#pragma config BORV = 3         // Brown Out Reset Voltage bits (Minimum setting)

// CONFIG2H
#pragma config WDT = OFF        // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
#pragma config PBADEN = ON      // PORTB A/D Enable bit (PORTB<4:0> pins are configured as analog input channels on Reset)
#pragma config LPT1OSC = OFF    // Low-Power Timer1 Oscillator Enable bit (Timer1 configured for higher power operation)
#pragma config MCLRE = ON       // MCLR Pin Enable bit (MCLR pin enabled; RE3 input pin disabled)
```

```
// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection bit (Block 0 (000800-003FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection bit (Block 1 (004000-007FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection bit (Block 2 (008000-00BFFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection bit (Block 3 (00C000-00FFFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection bit (Block 0 (000800-003FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection bit (Block 1 (004000-007FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection bit (Block 2 (008000-00BFFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection bit (Block 3 (00C000-00FFFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
```

```
// CONFIG7L

#pragma config EBTR0 = OFF      // Table Read Protection bit (Block 0 (000800-003FFFh) not protected from table reads executed in other blocks)

#pragma config EBTR1 = OFF      // Table Read Protection bit (Block 1 (004000-007FFFh) not protected from table reads executed in other blocks)

#pragma config EBTR2 = OFF      // Table Read Protection bit (Block 2 (008000-00BFFFh) not protected from table reads executed in other blocks)

#pragma config EBTR3 = OFF      // Table Read Protection bit (Block 3 (00C000-00FFFFh) not protected from table reads executed in other blocks)


// CONFIG7H

#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block (000000-0007FFh) not protected from table reads executed in other blocks)


// #pragma config statements should precede project file includes.

// Use project enums instead of #define for ON and OFF.


#include <xc.h>

#include <string.h>


#define _XTAL_FREQ 10000000      // Define osc freq for use in delay macros
```

Macros.h

```
/*
 * File:   macros.h
 * Author: Administrator
 *
 * Created on August 17, 2016, 2:42 PM
 */

#ifndef MACROS_H
#define  MACROS_H

#define __delay_1s() for(char i=0;i<100;i++){__delay_ms(10);}
#define __lcd_shift() lcdInst(0b11111000)
#define __bcd_to_num(num) (num & 0x0F) + ((num & 0xF0)>>4)*10
#define __lcd_newline() lcdInst(0b11000000)
#define __lcd_clear() lcdInst(0b00000001)
#define __lcd_home() lcdInst(0b10000000)


#endif   /* MACROS_H */
```

Constants.h

```
/*
 * File:   macros.h
 * Author: Administrator
 *
 * Created on August 17, 2016, 2:42 PM
 */

#ifndef MACROS_H
#define  MACROS_H

#define __delay_1s() for(char i=0;i<100;i++){__delay_ms(10);}
#define __lcd_shift() lcdInst(0b11111000)
#define __bcd_to_num(num) (num & 0x0F) + ((num & 0xF0)>>4)*10
#define __lcd_newline() lcdInst(0b11000000)
#define __lcd_clear() lcdInst(0b00000001)
#define __lcd_home() lcdInst(0b10000000)


#endif   /* MACROS_H */
```