# Mechatronics Final Project Report

## MENG 483L-02

## December 8th, 2019

## Team 7

Guillaume Austin

Christopher Bond

John Bradshaw

Michael Diswood

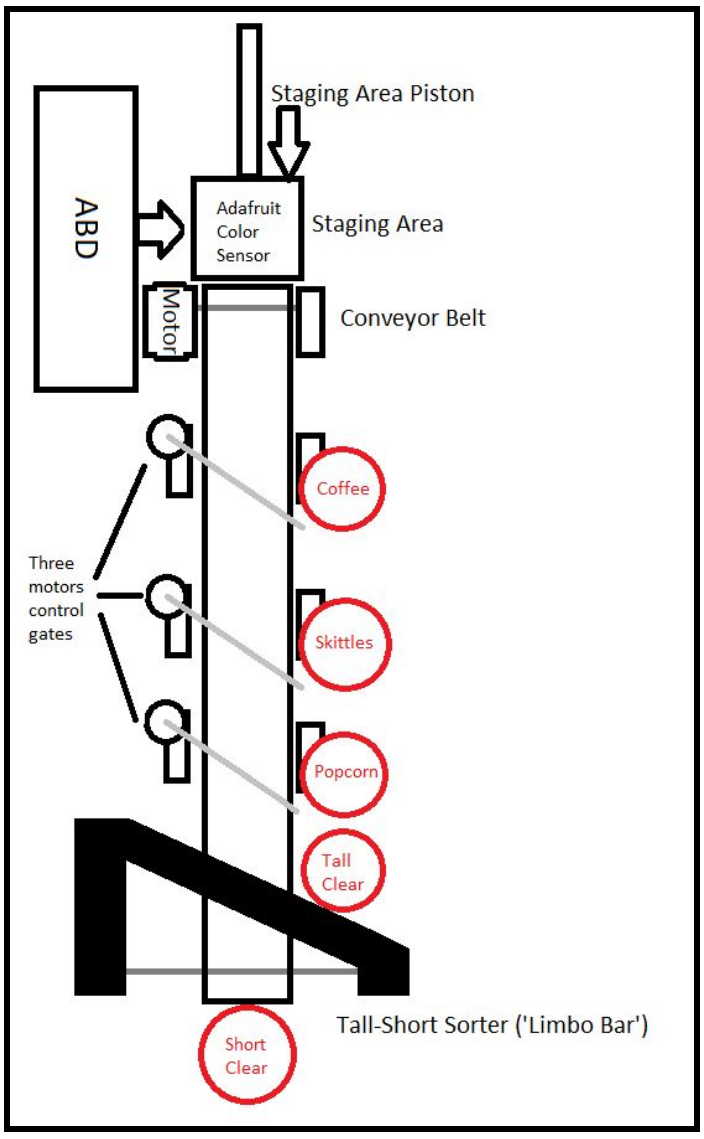
Christopher Mooty

**MENG-483L-02:**

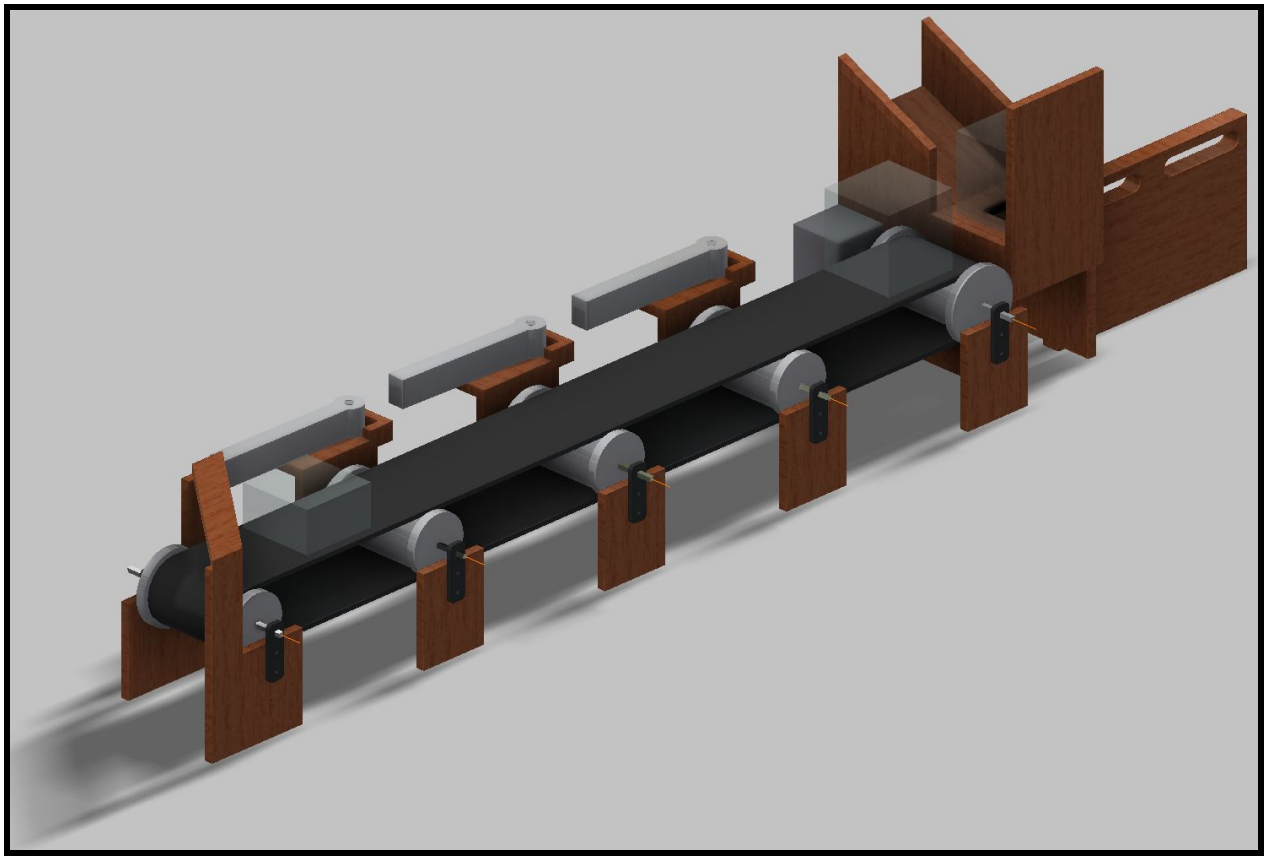
**To:** Dr. Lee
**From:** Guillaume Austin, Christopher Bond, John Bradshaw, Michael Diswood and Christopher Mooty
**Date:** 12/08/2019
**Subject:** Final Report

---

➢ **CAD model, hand drawings, or mechanical drawings**

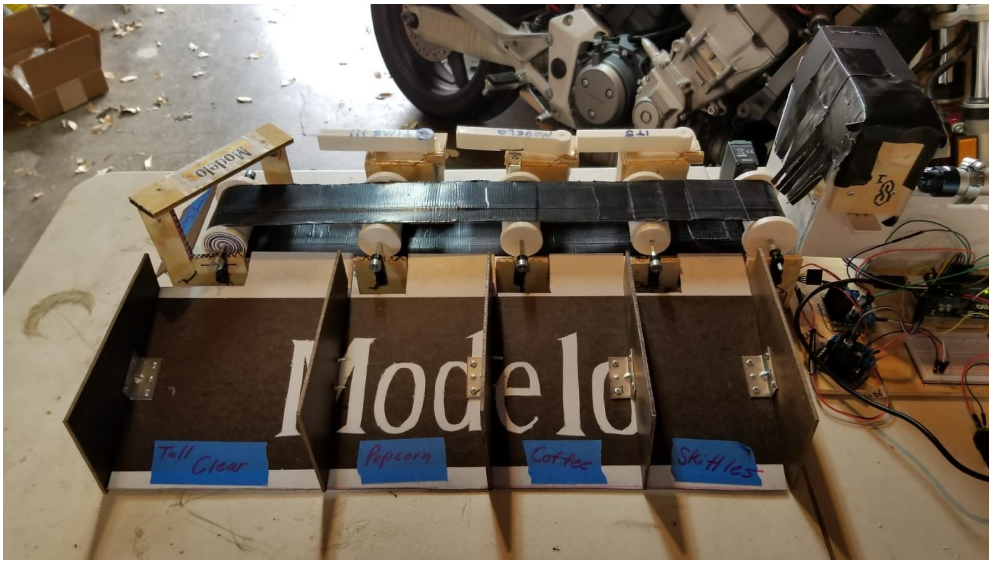



Outline depicting the general plan and characteristics of our BSD

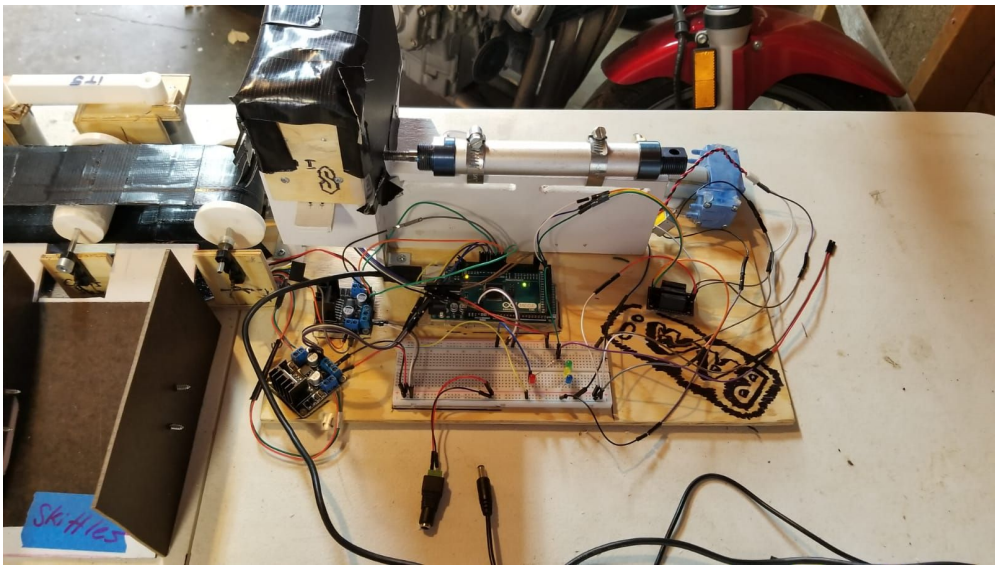


Rendered image of BSD assembly in Autodesk Inventor 2017

➢ **Photos of final product**




The conveyor belt assembly containing the limbo bar, actuating gates, and ramp for the boxes to safely slide down into their respective locations




The staging assembly containing the arduino, pneumatic pump, breadboard, converters securely attached to the base platform. The sensor platform includes the piston, hood, and color sensor.


Here is both the sensor platform and conveyor belt sorting system

➢ **Box sorting method description**

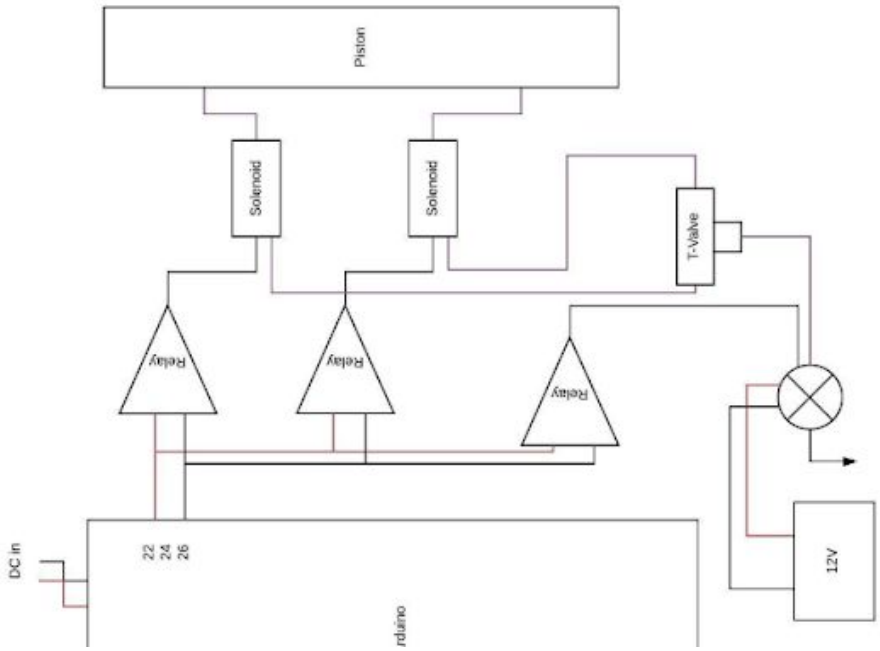
For this box sorter we used 1x piston, 1x servo motor, 3x DC motor, and 1x Adafruit Color Sensor. We had two parts to this process: 1) Decide which box is which and 2) sort according to data collected in part 1.

1. Box loader will loaded a box onto a staging platform. This platform used a color sensor on the bottom and used the light from sensor. The color sensor detected the color that reflected back from the box. With this we semi accurately determined between popcorn, skittles, coffee and clear.

2. The piston behind the platform extended, pushing the box onto the conveyor belt. This conveyer belt carried the boxes and were diverted by swinging arms attached to motors as shown in the first diagram.
    a. The first three motors sorted via the motor arm extending or letting the box pass on the conveyor belt. This will be responsible for the skittles, popcorn and coffee boxes.
    b. The last two boxes are the clear tall and clear short. We will have a height divider that will allow the short to pass through freely and the tall to be redirected off the path.

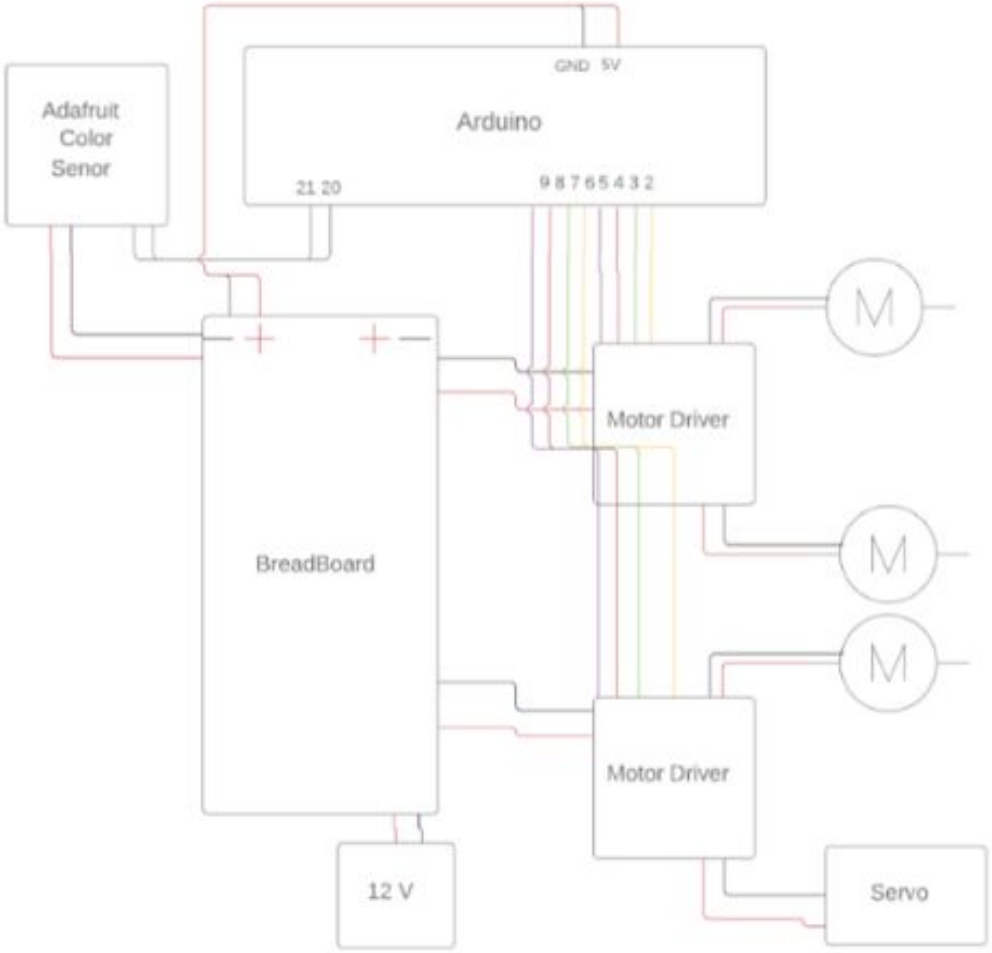
➢ **Pneumatic network diagrams w/ description**

The pneumatic network diagram is of a signal piston that only needed two solenoids and a single pump. The pneumatic network's purpose was to push the box onto the conveyor belt to be sorted after the color sensor reading was taken of the box.




➢ **Electrical diagrams & pneumatic network diagrams**

The diagram shows the PIN attachments to the PWM pins 2-9 for the two motor drives connected to three DC motors, and one servo motor for the conveyor belt. The position of the three motors sort the boxes by opening and closing of the gates, and the continuous running of the belt are controlled by these pins. For the adafruit color sensor was wired to pin numbers 20 and 21 to take readings of the box, which determine which gates opened and how long it stayed until the next box.


Adafruit
Color
Senor
GND 5V
Arduino
21 20
9 8 7 6 5 4 3 2
M
Motor Driver
BreadBoard
M
M
Motor Driver
12 V
Servo

➢ **Final Code uploaded into the Arduino**

boxsorter.ino // main.ino file

```
/*this code will run the box sorting machine.
  it contains both the setup and loop functions, and acts as
  the main arduino file.
  Created by John B. Bradshaw, 2019.
*/
// user defined class files
#include <...\Arduino\boxsorter\obj\abd_class\abd_class.ino>
#include <...\Arduino\boxsorter\obj\motor_class\motor_class.ino>
#include <...\Arduino\boxsorter\obj\piston_class\piston_class.ino>
#include <...\Arduino\boxsorter\obj\box_class\box_class.ino>
#include <Wire.h> //I2C library to communicate with the sensor
#include <Adafruit_TCS34725.h> //Color sensor library

const int ABDPIN = 28;
const int PUMP = 22, S_E = 24, S_R = 26;
const int CPIN1 = 4, CPIN2 = 5;
const int LEDPIN1 = 10, LEDPIN2 = 51;

// pwm on time 0 to 255 speed (digital)
int cSpeed = 175, gSpeed = 175, gTime = 50, gHold = 50;
int boxType;// box number 0 - 5
int sTime = 1000, cofTime = 1000, popTime = 1500, clearTime = 750;
int t = 750; // gate delay: set individualy for each gate
unsigned long i = 0; // counter for precise ABD control
uint16_t red, green, blue, clear; // create color values for sensor

// Create objects via constructor functions
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);
Piston piston(PUMP, S_E, S_R);// pump, PIN_E, PIN_R
Box clearBox(0), sBox(1), cofBox(2), popBox(3), noBox(4); // box objects
ABD abd(ABDPIN);// NEXT pin
Motor cs(CPIN1, CPIN2, NULL);//convoy motor; set motor number to NULL
Motor g1(3, 2, 1), g2(7, 6, 2), g3(8, 9, 3);//(PINA, PINB, motor number)
```

```
// piston control function that unloads the dock for the clear boxes
void unloadDock(int t)
{
  piston.push();
  delay(t);
  piston.pull();
  delay(t);
  piston.hold();
}

void setup()
{
  Serial.begin(9600);
  if (tcs.begin())
  { //sensor starts correctly
    Serial.println("Found sensor");
  }
  else
  { //sensor starts incorrectly
    Serial.println("No TCS34725 found ... restart Arduino!");
    while (1)
      ; // halt!
  }
  /*******SENSOR CALIBRATION*********/
  noBox.setRangeClear(0, 500);
  clearBox.setRangeClear(500, 1905);
  cofBox.setRangeClear(1906, 3000);
  sBox.setRangeClear(3001, 4251);
  popBox.setRangeClear(4250, 6000);
  /*********************************/
  piston.setPump(HIGH); // start piston pump
  g1.setSpeed(gSpeed, gTime, gHold); //motor setup functions
  g2.setSpeed(gSpeed, gTime, gHold);
  g3.setSpeed(gSpeed, gTime, gHold);
  cs.go(cSpeed); // counter strike: GO! (conveyor motor)

  pinMode(LEDPIN1, OUTPUT); // turn on supporting LED light
```

```
  digitalWrite(LEDPIN1, HIGH);
  pinMode(LEDPIN2, OUTPUT); // turn on led light on sensor
  digitalWrite(LEDPIN2, HIGH);
}

void loop()
{
  i++;
  if (i % 2 == 0)   //load a box from the abd every other loop iteration
    abd.dispense(); //reduce the risk of jams or overloading the machine
  delay(2000);
  tcs.getRawData(&red, &green, &blue, &clear); //read the sensor
  tcs.getRawData(&red, &green, &blue, &clear);
  tcs.getRawData(&red, &green, &blue, &clear);
  tcs.getRawData(&red, &green, &blue, &clear);
  tcs.getRawData(&red, &green, &blue, &clear);
  tcs.getRawData(&red, &green, &blue, &clear);
  // grab a few readings from the sensor to try and improve accuracy
  Serial.println(clear); //print color value

  // Box sorter algorithm based on clear variable
  if (clear >= sBox.getColorRangelow() && clear <=
sBox.getColorRangehigh())
    boxType = sBox.getBoxNum();
  else if (clear >= cofBox.getColorRangelow() && clear <=
cofBox.getColorRangehigh())
    boxType = cofBox.getBoxNum();
  else if (clear >= popBox.getColorRangelow() && clear <=
popBox.getColorRangehigh())
    boxType = popBox.getBoxNum();
  else if (clear >= clearBox.getColorRangelow() && clear <=
clearBox.getColorRangehigh())
    boxType = clearBox.getBoxNum();
  else
    boxType = noBox.getBoxNum();

  Serial.println(boxType);
```

```
// Truth table for motor control
if (boxType == g1.getMotorNumber()) // skittles box
{
  g1.openGate();
  piston.push();
  delay(t);
  piston.pull();
  g1.hold(sTime);
}
else if (boxType == g2.getMotorNumber()) // coffee box
{
  if (g1.getGateState())
    g1.closeGate();

  g2.openGate();
  piston.push();
  delay(t);
  piston.pull();
  g2.hold(cofTime);
}
else if (boxType == g3.getMotorNumber()) // popcorn box
{
  if (g1.getGateState())
    g1.closeGate();
  if (g2.getGateState())
    g2.closeGate();

  g3.openGate();
  piston.push();
  delay(t);
  piston.pull();
  g3.hold(popTime);
}
else if (boxType == clearBox.getBoxNum()) // clear box
{
  if (g1.getGateState())
```

```
      g1.closeGate();
    if (g2.getGateState())
      g2.closeGate();
    if (g3.getGateState())
      g3.closeGate();

    delay(100);
    unloadDock(clearTime);
  }
  else if (boxType == noBox.getBoxNum()) // no box
  {
    if (g1.getGateState())
      g1.closeGate();
    if (g2.getGateState())
      g2.closeGate();
    if (g3.getGateState())
      g3.closeGate();

    delay(100);
  }
}
```

# abd_class.ino // automatic box dispenser class file

```
/* Defines the automatic box dispenser class
   this class has a pin value and a pulse function
   Created by John B. Bradshaw, 2019.
*/
class ABD
{
    private:
        int npin;
    public:
        ABD(int pin)
        {
            npin = pin; // accpet and set pin
            pinMode(npin, OUTPUT);
```

```
            digitalWrite(npin, LOW); // default pin off
        }
    void dispense(){
        digitalWrite(npin, HIGH);
        delay(15);
        digitalWrite(npin, LOW);
    }
};
```

# box_class.ino // box class file

```
/* This class defines the box object. Each box has a specific name, number
   and range of color values that is seen by the color sensor
   Created by John B. Bradshaw, 2019.
*/
class Box{
    private:
        int boxNum;
        char boxName;
        int colorRangelow;
        int colorRangehigh;
    public:
        Box(int bn) // ~constructor - assign a box number to each box 0-5
    {
        boxNum = bn;
        switch(boxNum)
        {
            case 0 : boxName = 'no box ';
            case 1 : boxName = 'skittles ';
            case 2 : boxName = 'coffee ' ;
            case 3 : boxName = 'popcorn ';
            case 4 : boxName = 'clear box ';
            case 5 : boxName = 'clear short box ';
            default : boxName = 'Invalid Input! ';
        }
    }
    // Set functions
```

```
    void setRangeClear(int lr, int hr) // set clear lower and upper bounds
    { colorRangelow = lr; colorRangehigh = hr;  }
    int getColorRangelow() // Get functions
    { return colorRangelow; }
    int getColorRangehigh()
    { return colorRangehigh; }
    int getBoxNum()
    {   return boxNum;  }
    char getBoxName()
    {  return boxName;  }
};
```

# Piston_class.ino // piston class file

```
/* This file defines the piston class by setting relay Pins
   contains a state variable for piston position and pump state,
   and contains member functions for push, pull and hold
   Created by John B. Bradshaw, 2019.
*/
class Piston{
    private:
        int pump_relay,
            relay_extend,
            relay_retract; // Relay Set Pins
        int time; // used for arduinos delay function
        volatile int switchState;
        volatile int pumpState;
    public: // constructor function
    Piston(int rP, int rE, int rR)// PUMP, EXTEND, RETRACT
    {
        pump_relay = rP;
        relay_extend = rE;
        relay_retract = rR;
        switchState = LOW;
        time = 0;

        pinMode(pump_relay, OUTPUT);
```

```
        pinMode(relay_extend, OUTPUT);
        pinMode(relay_retract, OUTPUT);

        digitalWrite(pump_relay, switchState);
        digitalWrite(relay_extend, switchState);
        digitalWrite(relay_retract, switchState); // Set all to off
    }
    void setPump(volatile int S) // turn on pump
    {
        pumpState = S;
        digitalWrite(pump_relay, pumpState);
    }
    void push()
    {
        switchState = HIGH;
        digitalWrite(relay_extend, !switchState);
        digitalWrite(relay_retract, switchState);
    }
    void pull()
    {
        switchState = HIGH;
        digitalWrite(relay_extend, switchState);
        digitalWrite(relay_retract, !switchState);
    }
    void hold()
    {
        switchState = LOW;
        digitalWrite(relay_extend,switchState);
        digitalWrite(relay_retract,switchState);
    }
};
```

motor_class.ino // motor class file

```
/* This defines the gate motor class. The data struct sets the motor pins
    and sets the default state to closed (a box can pass through).
    member functions spin the motor cw or ccw based on given inputs
    and change if the gate is open or closed based on a boolean value.
    ***The go function is used ONLY for the conveor belt motor***
    Created by John B. Bradshaw, 2019.
*/
class Motor{
    private:
        int PinCW, PinCCW; // pin A and B (analog)
        int speed; // 0 to 255 speed
        int onTime; // time the motor spins
        int motorNum; // comparator for sensor input (which gate?)
        int holdval; // set holding strength
        bool gateState; // tell is open(can pass) or closed(cannot pass)
    public:   //Consructor: create motor, initalize member vars and state
    Motor(int a, int b, int num){
        PinCW = a;  // accept and set encoder pin A and B
        PinCCW = b;
        pinMode(PinCW, OUTPUT);
        pinMode(PinCCW, OUTPUT);
        motorNum = num; // number 1, 2, 3...
        gateState = false; // set the gate to be closed by default
    }
    void go(int s) // Make it go fast (conveyor only)!
    {
        speed = s;
        analogWrite(PinCW, speed);
        analogWrite(PinCCW, 0);
    }
    void openGate()
    { //move the motor a bit cw
        analogWrite(PinCW, speed);
        analogWrite(PinCCW, 0);
        delay(onTime);
        analogWrite(PinCW, holdval);
        analogWrite(PinCCW, 0);
```

```
        gateState = true;
    }
    void closeGate()
    { //move the motor a bit ccw
        analogWrite(PinCW, 0);
        analogWrite(PinCCW, speed);
        delay(onTime);
        analogWrite(PinCW, 0);
        analogWrite(PinCCW, holdval);
        gateState = false;
    }
    void hold(int t){
      analogWrite(PinCW, holdval);
      analogWrite(PinCCW, 0);
      delay(t);
    }
    int getMotorNumber() // motor comparator for boxnum variable
    { return motorNum; }
    int setSpeed(int s, int t, int h)
    {
        speed = s; // 0 to 255
        onTime = t; // time of pulse
        holdval = h; // set hold strength
    }
    bool setGateState(bool state)
    { gateState = state; }
    bool getGateState()
    {  return gateState;  }
};
```

➢ **Notes on the Software Design:**

The software for the machine was designed with object-oriented programming, using one Main.ino file, four user-defined-class files (UDCs), and two premade header files from the Arduino library. Arduino C is built from C++, and allows for OOP design through UDCs. A UDC was created for every part of the machine needing coding, including the piston, motors, automatic box dispensor, and the boxes to be sorted. It was not necessary to create a UDC for the Adafruit color sensor because one already exists in the Arduino library. This design was chosen

for ease of testing and debugging. Use of OOP allowed each class and object to be tested individually, and was especially useful for the motors and piston controls, since it allowed the controls subteam to conduct testing independent of the build team early on. Several different main.ino files were created and then discarded for this purpose.

The primary IDE used was MS Code, loaded with the Arduino C language and debugging package with the 8-bit dark theme (seen above). Testing was done with Arduino Genuino IDE.

Once each class was defined and tested, class member functions could easily be created or expanded to meet the needs of the machine as it was put together. Once all class files were created, the next challenge was creating the logic (sorting algorithm), timing optimization, and sensor calibration. These tasks were greatly simplified by the use of OOP and modular functionality. One additional function was created in the main.ino file, that being the unloadDock() function. This function was useful as a failsafe if the machine became jammed and needed a reset, and also as a catchall for clear boxes (which required no motor control to be sorted); the function itself used member functions from the Piston class. This modular programming approach was essential to the success of the design.

The design could be improved by a redesign of the control system, it would be interesting to explore the use of IR sensors in place of a color sensor for improved sorting accuracy. Another avenue to explore would be optimizing the speed of the machine and improving timing calibration. Both these options would serve as a next step for a team wanting to improve this design.

- **Description of the Machine's performance prior to the competition**

The machine worked about as well before our competition as it did during the competition. In several informal runs we calculated about a 75% success rate in sorting the boxes, when the sorter was working perfectly. However, we did not feel that this was a good enough so we did many things to try and improve the sorting ability. Our corrections did work well because during the competition it sorted 70% correct when there were several misshapes that lowered out score. For example: the belt malfunctioned halfway through our second trial, which significantly slowed the speed of the machine. The corrections we made to make our box sorter is described in the following section.

- **Describe changes or revisions of your machine**

We were able to build the mechanical components just as well as we anticipated. Using plywood, wood glue, and L brackets we were able to build everything so the electrical components integrated with little complications. We did have trouble getting consistent readings out of the light sensor so we spent most of our time coming up with ways to get a more consistent reading. The first major thing we did was make a hood to keep the light away from the boxes so the sensor could get an accurate reading. We still

found that just changing rooms changed the calibration enough to mess with the results. This is why our first trial did not go as well as we thought it would, and after we recalibrated the sensor to the competition room the accuracy of the sorter went way up.

The next thing we did was add a foam backing to keep the boxes from bouncing around so much when they slide down the ramp. The reasoning for this was to get the boxes in a more consistent position so the light sensors reading was accurate. Along this same line of reasoning we colored the inside black so the was absorbed instead of reflected around, diluting the sensors reading.

The final big change we made was to use the light from the light sensor instead of an LED hanging above it. This is because to make the overhead light work the light would need to be almost touching the box which would be hard to accomplish with the way our staging area worked. The underneath light provided more consistency which improved the accuracy of the box sorter overall.

➢ **Describe your machine's performance during the competition**

For the first trial our box sorter started out by performing well, until the box launcher had jammed and stopped the sorting process. From that point on our sorting method was switching the coffee with short clear and popcorn and skittles. But this was before we recalibrated the color sensor's set of trigger values.

The second trial, our box sorter started working well at the start but the conveyor belt got out of line and bound up the whole system. This really strained the motor and slowed down the sorting process. We do not know if any boxes were sorted incorrectly because of this.

➢ **Team member responsibilities and accomplishments**

1. Guillaume Austin- Designed and help construct the conveyor belt and supporting components. Supporting components include, conveyor belt construction/machining, driving motor fixturing, stepper-motor fixturing, stands, 'limbo-bar' fixturing, etc.

2. Christopher Bond- Helped build and manufacture the mechanical components that helped sort the boxes in the correct places. Also helped diagnose and determine ways to improve the accuracy of the color sensor.

3. John Bradshaw- Coded the sensors, gate control motors, conveyor belt motor, and launch mechanism that were built by the other team members. Made suggestions on mechanism design to be compatible with computer control. Developed the sorting algorithm and gate logic. Involved in calibrating the color sensor to accurately determine the difference

between boxes. Created and tested all class files, member functions, and object constructors used in the software. Created and tested the main.ino file.

4. Michael Diswood- Wired all the electrical components in the box sorter so they work correctly with the coding written by John. Also testing each electrical component individually to ensure no faulty equipment was being used in the final design.This also involves simplifying the wiring so that it worked seamlessly with the mechanical components that were designed and built by other team members

5. Christopher Mooty- Modeled and help build the staging area to work seamlessly with the conveyor belt, electronics and the launcher to ensure smooth transitions. The staging area was where the sensors determined what box was about to be sorted. This required working with John and Michael to ensure that everything was wired correctly and that everything was compatible.