THE UNIVERSITY OF TRINIDAD AND TOBAGO

TITLE: *Mechatronics Project – Pepper Sorting Device*

COURSE: MENG 3011 – Mechatronics

LECTURER: Dr Nadine Sangster

GROUP MEMBERS:

Vernel Young - 64271

Sam Greenidge - 63240

Gerard Ragbir - 65728

DATE DUE: 14th April 2016

Table of Contents

1 DESIGN SUMMARY

This project sought to develop a device to sort items, specifically peppers. The sorting categories are by colours: especially green and ripe peppers. The device also counts and measures the weight of each respective bin where the sorted peppers are stored. The device, as shown in figure 1, consists of several key parts. The intake area where the unsorted peppers are placed (A). The automated transport system, which is a conveyor belt (B). A user interactive control panel which is connected to an Arduino Mega microcontroller (C); An actuated arm (D) switches to either of its positions to allow the peppers to change direction, either to the left or the right depending on the detected colour. Finally, the sorted peppers are collected in two bins (E). Each bin can check for the changes in weight, so as the weight increases to a desired cut off point, the machine will alert the operator then auto turn off.

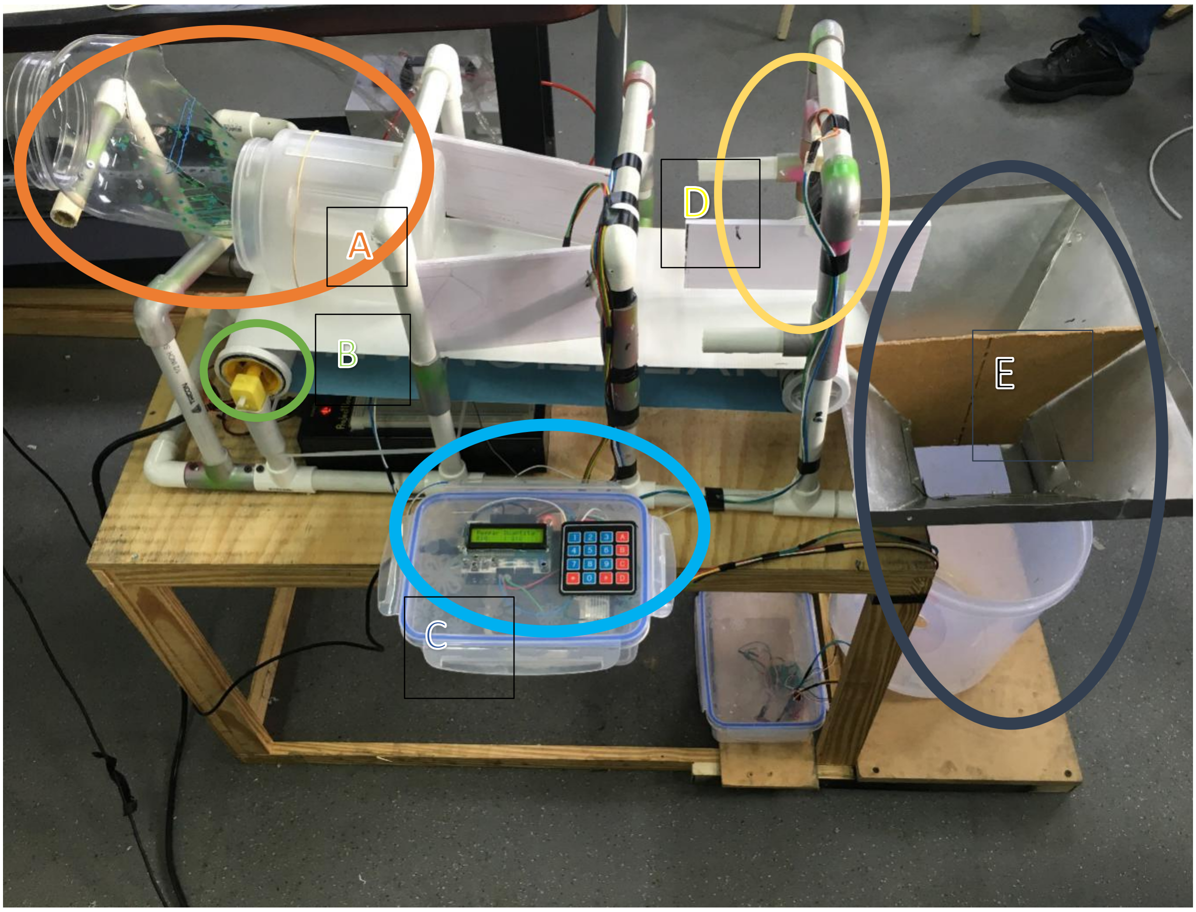

Figure 1: Picture showing a labelled side profile of the device prototype.

The dispenser assembly located at A, in figure 1, consists of a collection tray and a feeder tray coupled together. As items (peppers) are loaded onto the collection tray, they fall into the feeder and settle near the aperture. The agitator motor (not shown above, located behind the plane of the image) causes a vibration to constantly displace the peppers in single file onto the conveyor belt, which is turned and begins at B.

The arms at D will push aside any incoming peppers according to its determined colour. The peppers are resilient enough to fall into the short drop that is at E, where a collection pan will receive the separated peppers. The controller and interface are powered at C and is the only position where the operator can interact with the system.

The device sorts peppers by two categories: Green and Non-Green (includes orange, red and mixes), since the colour has often served as an indicator of spiciness, *a la* Scoville Heat Scale.

2 SYSTEM DETAILS

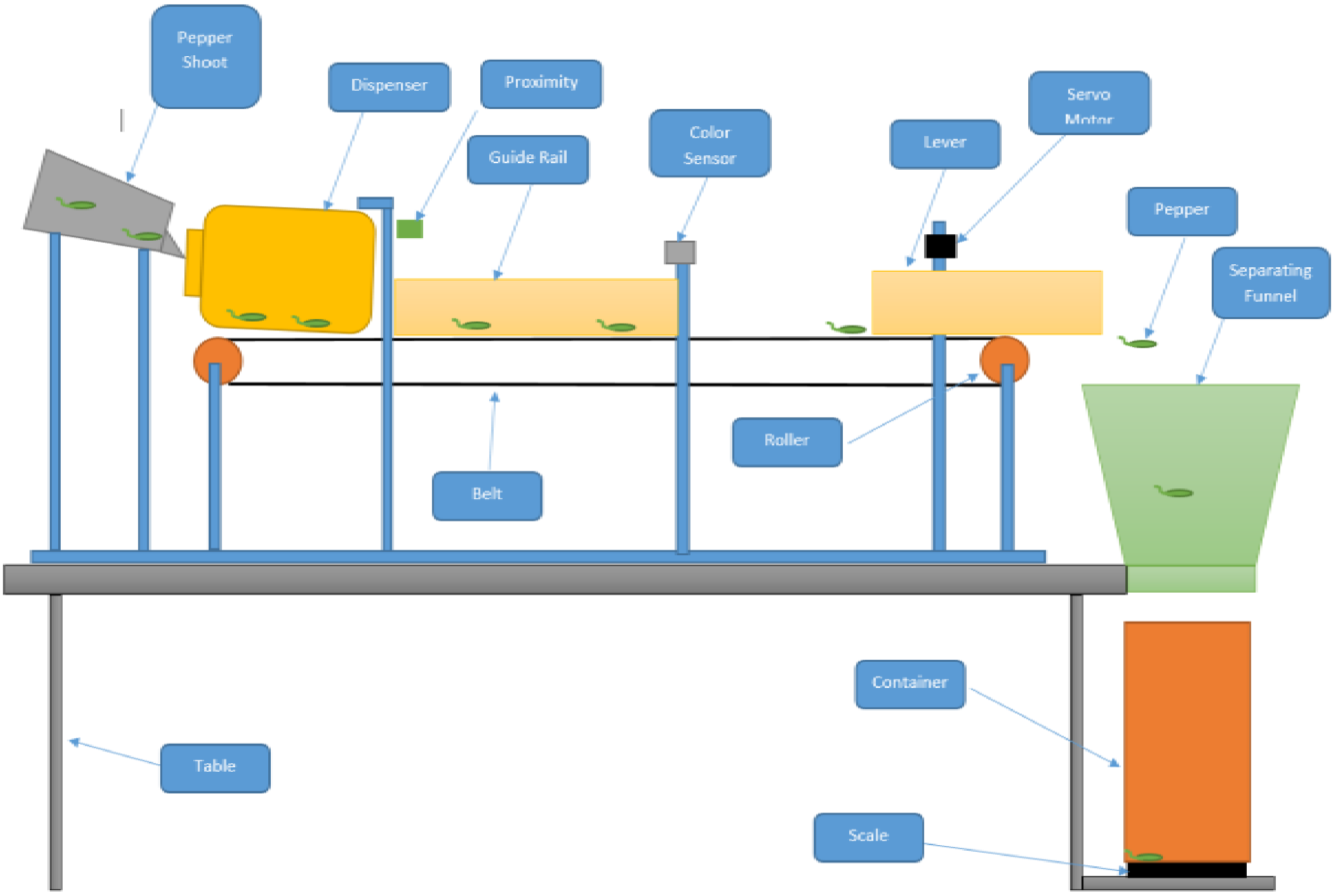

Figure 2: Showing a simplified view of the device.

2.1 Process flow

Figure 2 shows a simplified side view of the machine, while figure 3 shows the logical flow of the process. The system operates on a general principle of sorting items. However, for specifying a particular application, the items in use for the prototype were those of pimento peppers. When the machine process is started, the items are required to be loaded at the pepper shoot into a dispenser. The operator inputs the required quantity or weight desired into the keypad. The operator then starts the machine.

With the system started, the conveyor system would start to move and the dispenser's agitator (a controlled motor) applies a rapidly changing force to the dispenser. This allows for peppers lodged or stuck at the cavity/aperture to be move around until they become freed as well as to ensure that peppers leave the dispenser in single file. When a single pepper gets through the cavity, it becomes caught by the conveyor system and is carried to the sensing and sorting electronics.

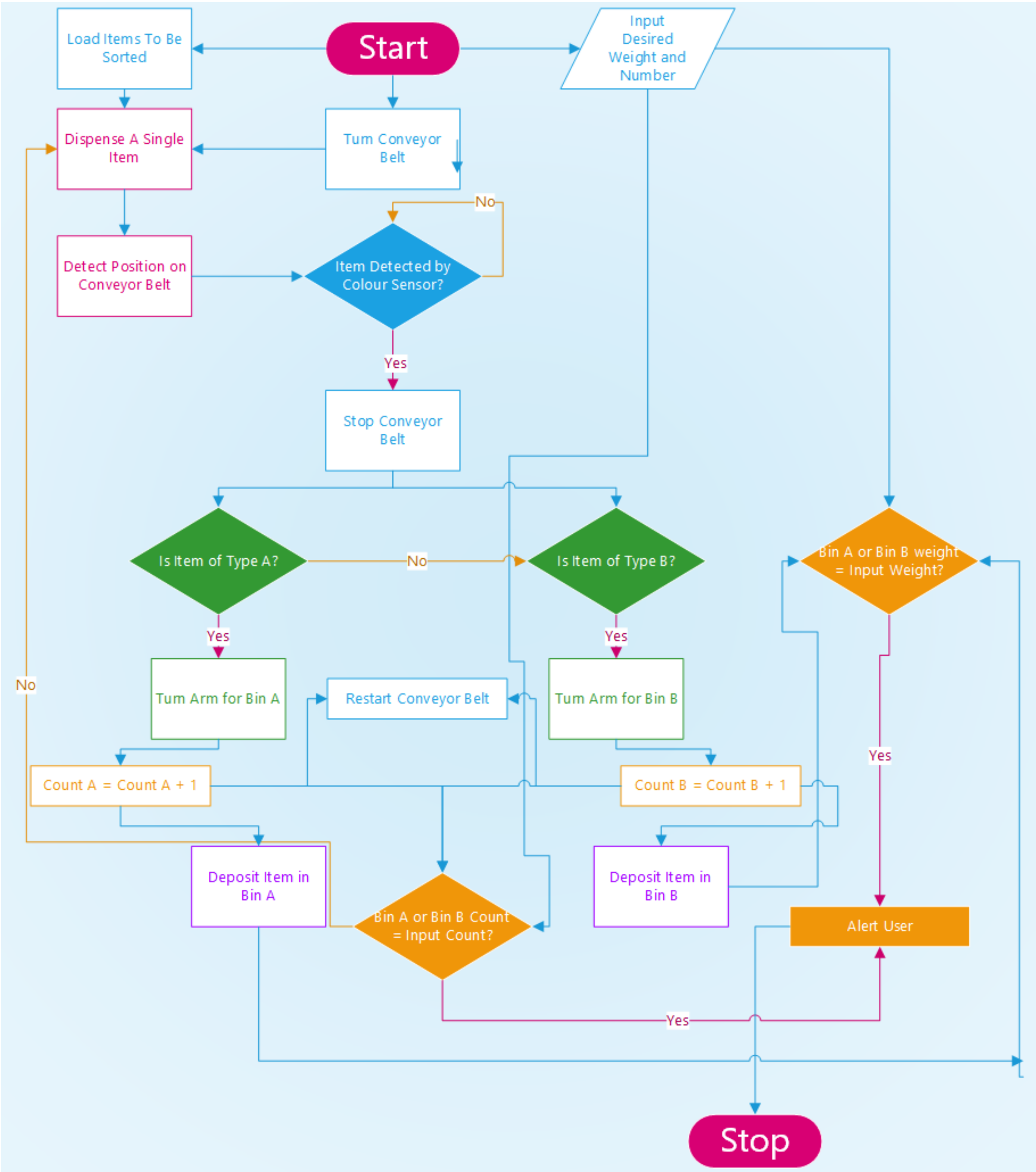

Figure 3: Showing Operational Flow Chart

An ultrasonic transducer pings the plane above the conveyor scanning for any item *en route* to the opposite end of the conveyor, and signals the controller to halt the belt in the event that it does detect one. This triggers the colour sensor to power on its array of light emitting diodes (LEDs) to illuminate the surface of the peppers and sense the colour. If the colour sensed is indeterminate, the sensor will resample the pepper. The controller, depending on the colour range selected, will adjust an arm suspended at the end of the conveyor belt and actuate it using a servomotor causing the pepper to be channelled in the desired direction.

At the opposite end of the conveyor are two collection bins. Each collection bin sits atop a load cell, which is connected to a bridge and consequently the H-bridge system is then attached to a Load Cell Amplifier. Each load cell measures the weight distributed over it within the bin and feeds back to the controller. The controller in turn compares the desired weight input at time of initial setup on the keypad. Should the desired weight be reached for either or both bins, the machine will alert the user and auto shut off until the operator resets the process and restarts it. The same holds for when the ultrasonic transducer detects an item in its path, it will add a single integer to the count. The controller, to match if any initially set condition for quantity is met, also does a comparison. If this occurs, the machine will also halt operation and alert the user by playing a tune.

The entire process is automated requiring hardly any manual intervention. The conveyor adjusts based on the determined requirements of the system. In the event of failure, the conveyor has a redundant motor which is capable of supplying the same torque to the belt and keeping it in motion.

2.2 System design

Figure 4 shows a functional block diagram of the system, its input/output relations and the flow of information. The system uses a nonlinear two-position control method to make logical decisions. It consists of several high level functions, namely:

- Main logic - The main logic controls the general program flow and follows a predetermined order of execution for each of the other logical functions within the system. It also keeps a record of the system state, which is queried by the other functions. It receives digital input from the operator though the keypad or LCD push button menu and outputs information to the LCD screen or buzzer.
- Selection logic - The selection logic function works in conjunction with the Sorting logic function to detect, select and sort the peppers. It also controls the ON/OFF state of the conveyor belt. It receives digital information from a proximity sensor, which detects whenever a pepper passes within its scanning range, and digital system state data to the sorting logic and the main logic

- Capacity logic – The capacity logic function monitors each of the load sensor attached to each pepper colour bin against the weight limit set by the operator. It receives digital information from a 24-bit ADC/load cell amplifier. The information came as an analogy input from the load sensors, which is sent to the load cell amplifier/ADC unit.
- Dispenser logic – The dispenser logic controls the ON/OFF state of the dispenser. It in turn is controlled by system state data produced by the selection logic and the sorting logic.
- Sorting logic. The sorting logic controls the sorting of the peppers. It controls the operation of the sorting servomotor with in turn directs the pepper flow on the conveyor belt. It receives digital data from the colour sensor, which in turn came from the sampled pepper.

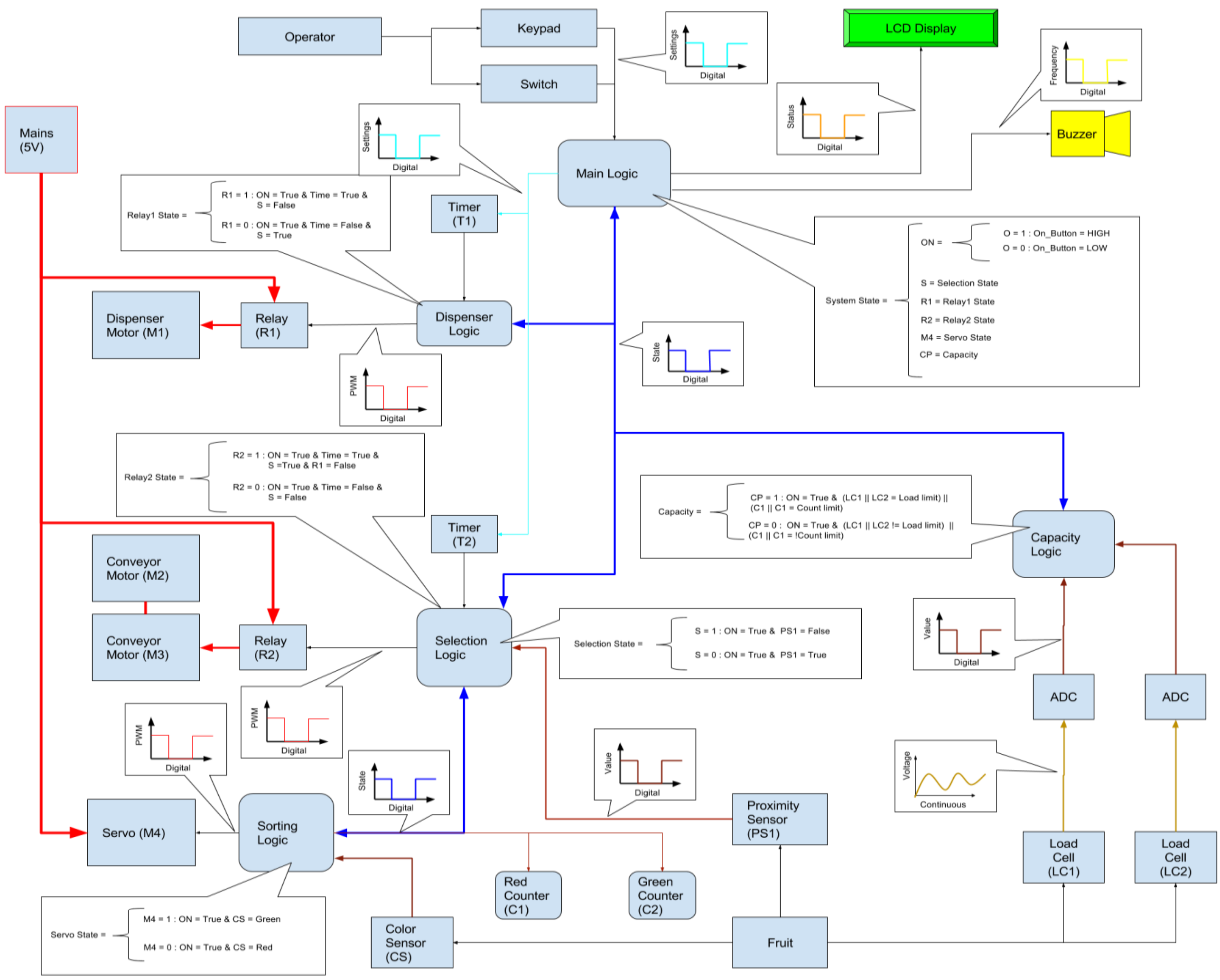

Figure 4: System FBD

Table 1 shows the design specification of the system.

Table 1: Design specification

Category	Requirements	Comments
GEOMETRY	• Maximum Length - 30 in • Maximum Width – 12 in • Maximum Height - 18 in	
PERFORMANCE	• Minimum sorting speed - 1 pepper/min • Maximum load capacity – 50kg	
ENVIRONMENT	• Temperature range – 15 - 25 deg • Humidity range – 40 – 70%	
SAFETY	• No loose parts • Have auto shut down • Prevents accidental start-up	
OPERATION	• Indoor use only	
COST	• Maximum manufacturing cost - $ 700.00	

2.3 Main Components

The following subsection takes a brief look at the main components in the system, which implements the systems functions. Appendix A provides a detailed wiring diagram of these components.

2.3.1 LCD Display

The LCD used in the project is a two row, sixteen-character display. To enable the viewing of information relating to the weight and quantity of green and ripe peppers sorted by the system, the data is broken up into two display events. Each event is cycled for in a four (4) second period. Therefore, the operator will observed the weight of green and ripe peppers shown for 4sec then the display will flip and show the quantity of green and ripe peppers. The display also came with a built-in push button menu. Each button was assigned the functions: System start/stop; System debug start/stop and Scale reset, see figure .

Figure 5: Control Interface

2.3.2 Load Sensor

The project contains two strain gauges, which acts as load sensors. Each sensor have a maximum capacity of 50kg. It should be pointed out; there are two types of load cells: Full Load Cells, which include the full part of the H-bridge, and Partial Load Cells, which act as a quarter-bridge (of a H-bridge setup). The ones in use are the Partial Load Cells/load sensors, and unfortunately, there was no documentation on the actual proper setup from the manufacturer to determine the difference between either categories.

2.3.3 Arduino Mega

The controller for the sorting device was implemented on an Arduino Mega microcontroller. This amounts to approximately 1000 lines of lines of code in the Arduino C/C++ language, see Appendix B for code details. The mega is therefore the ‘brain’ of the system. It controls all the inputs and outputs, which allows for the smooth interfacing of sensors, relays, transistor and other discrete electronic components as well as actuators.

2.3.4 Keypad

The keypad consists of sixteen buttons arranged in a 4-by-4 grid (4x4 array). Each row and column of this array is linked to a single pin arranged as 4-rows and 4-columns. When a button is pressed, it outputs a signal to the logic board in the form of a matrix coordinate. For example, button 1 would correspond to r1c1 (row 1 column 1) while the # button would coordinate to r4c3. As such, Arduino Mega digital pins 22 and 26 would signal for the former case, and pin 25 and 28 for the latter case. Figure 6 show the pin configuration of the keypad.

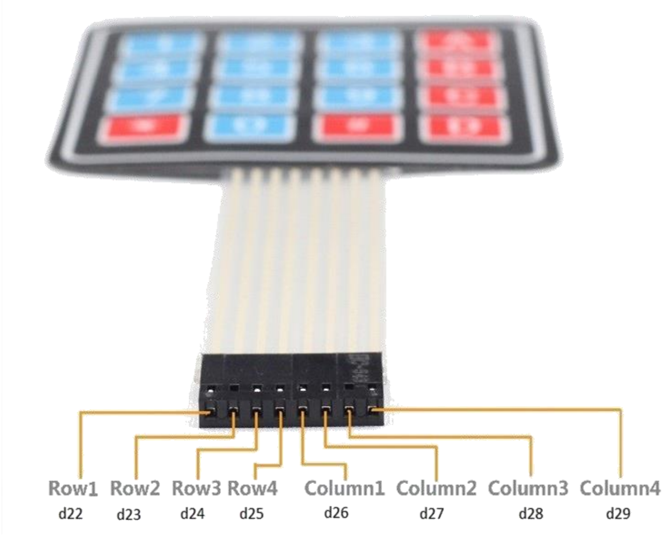

Figure 6: Pin Configuration for the Keypad

The keypad has the following four menu entry modes for each of its lettered buttons:

- A- Enter Quantity of Green Peppers,
- B- Enter Quantity of Red Peppers,
- C- Enter Weight of Green Peppers and
- D- Enter Weight of Red Peppers.

2.3.5 Colour sensor

The projects includes and RGB colour sensor. The colour sensor works by shining white light onto the peppers. The incident light is reflected from the surface of the peppers into the optoelectronic aperture measuring the respective Red-Green-Blue profile (RGB), each colour is calibrated to return a value within the range of 0-255. Zero being dark and 255 being full colour. The logic controller determines the ranges fall-in point, that is: more green or more red, as a means to determine the colour generalization. Typically, peppers may possess a wide range of colours or mixtures of colours due to their bio chromes, also known as chromophores

(Encyclopaedia Britannica, 2015). These natural pigments, especially for peppers, exists in the range of red, green or yellow. A mixture of these together may produce colours such as orange, a mix of green, red and/or yellow spots, see figure 7.

Figure 7: shows the variation of Pimento Peppers by shape, size and colours.

2.3.6 Proximity Sensor

The sorting device uses an ultrasonic sensor to detect the presence or absence of a pepper on the conveyor belt. This sensor have a range of 0 to 240cm but is only required to detect peppers within a 0 to 15cm range. This sensor uses the time of fight principle to measure song waves as they travel to and from the sensor.

2.3.7 Sorting Servo

The project is equipped with a micro servomotor that is coupled to an arm, which allows for the channelling of the direction of the pepper on the conveyor belt. The servomotor has a maximum range of pi-radians or 180 degrees (0-179 to be accurate), as such it has been programmed to throttle somewhere between the lower and upper limit angles. Since this now becomes the origin, the servo can rapidly switch between a negative displacement, i.e. rotate negative 90 degrees (half-pi radians) or positive displacement of positive 90 degrees. This allows the system to be aware of the angle, which the arm is extended toward, and as 6such, the arm acts as a guide to deposit the scanned pepper into its respective bins. There are two possible bins in this implementation, however more can be added with increased complexity whilst needing greater accuracy and tuning in the components.

3 DESIGN EVALUATION

Table 2 to table 4 show an evaluation of each design element and the scale used. In general all design element were able to perform at 80% or greater.

Table 2: Evaluation of each feature of the device.

Design Element	Assessment	Comment
Dispenser	Successful – Works Reliably	Dispenser was implemented fully and functioned as intended (90% reliable in trials).
Conveyor Belt	Successful – Works Reliably	Conveyor was implemented fully and functioned as intended (>90% reliable in trials).
Automated Counting	Successful – Works Reliably	A measurement system using the ultrasonic transducer was successfully implemented and works as intended with little to no failures/errors. (>90% reliable in trials).
Automated Stops	Successful – Somewhat Reliable	Using the ultrasonic transducer to issue stops based on positions, the conveyor belt was successfully stopped but is reliable sometimes. (>80% of trials).
Colour Sorting	Successful – Somewhat Reliable	The colours sorting algorithm was implemented but due to issues in the hardware received and the need for constant recalibration, the measurements are unreliable. Suggestion: May be improved by using a CMOS Imaging Sensor/Camera unit.
Weight Measurement	Implemented – Somewhat Reliable	The load sensors were implemented as expected but due to the sensors being half sensors (quarter of an H-bridge), readings are quite unreliable due to rapid fluctuations. With smoothing of the data implemented (using hardware smoothing), the data points fluctuate less but the sensitivity is reduced significantly.
Sorting Arm	Successful – Works Reliably	The sorting arm was implemented successfully and functioned in its task reliably (>90% reliable in trials).

Table 3: Reliability Scale (modified from Likert Scale)

Unreliable	Somewhat Unreliable	Reliable	Somewhat Reliable	Works Reliably
<=60%	>60%	>70%	>80%	>90%

Table 4: Assessment Categories

Successfully Implemented	Implemented	Not Implemented
Feature was fully implemented and operational.	Feature was implemented, partially implemented and operational or partially operational.	Feature was not Implemented or was Implemented but not used.

4 PARTS LIST

Table 5: Showing a partial, basic list of components used, sorted by functional elements.

Category	*Part*	*Quantity*
Output Display	LCD Board	1
Audio Output Device	Buzzer	1
Manual Data Input	Keypad (16 keys)	1
Sensor Input	Ultrasonic Transducer	1
	Colour Sensor	1
	Load Cell	2
	Encoder (on servo)	1
Actuators, Mechanisms, Hardware	Small DC motor (reversible)	2
	Medium DC motor (reversible)	1
	Small Servo Motor with arm attachment	1
	Conveyor Belt (on one active roller, one passive roller and two support rollers)	1
Logic, Processing, Control	Closed-loop feedback control	1
	Menu System (for interactive input of parameters)	1
Misc.	PVC and fittings	-

Table 6: Showing details of specialized components in the build.

Part	Description	Model No.	Vendor	Price
Ultrasonic Transducer	A range finding sensor, which uses emits and detects ultrasonic sound waves.	HC-SR04	Amazon	8.99 USD
Colour Sensor	An optoelectronic specialized photocell, which detects red, green and blue (RGB) channels of light.	TCS-3200	Dfrobot Electronics	8.99 USD
Load Cell	A quarter of an H-bridge setup, this cell measures the force or weight exerted on its surface.	SEN-10245	Sparkfun Electronics	9.95 USD
Load Cell Amplifier	A breakout board, which increases the accuracy of the resistance, changes of a load cell. Requires a Full Sized Load Cell or a pair of Partial Load Cells.	SEN-13230 HX711	Sparkfun Electronics	9.95 USD

5 CHALLENGES

The project was split into three general phases, each of which possessed its own set of difficulties:

1) Planning and Procurement

2) Build

3) Testing

During planning and procurement, despite ordering equipment as early as possible, the supplier found it necessary to delay the shipment by as much as a month due to two factors: firstly, lack of stock for one component and secondly for security issues involved in the logistics of specialized pieces of electronic equipment destined for outside the United States.

During the build stage, the dispenser design was debated a number of times and went through a number of concepts, each one had to be redesigned at some point. This was because they often dispensed more items than necessary, or the delay between dispenses were too vast. Another issue with the dispenser was that of the ability for the aperture to be able to account for the large variations in shape and sizes of the items. This was solved by limiting the aperture size to an estimated value of the quantity, which were possessed just before the time of testing. Additionally, suspending the dispenser over the conveyor allowed the excesses in sizes not accounted for to slip through the base slot. In the cases where the peppers would become stuck, an agitator was created to actuate a rapid shake, which would be controlled by the logic circuit to not allow the agitator to operate when a single item was already present on the conveyor. Due to the small length of the conveyor, the dispenser was limited to one per cycle with stops occurring when an item arrived at a desired checkpoint. If the conveyor was larger, it could have been possible to have the rate increased with a continuous process.

During the test stage, there were issues in getting many of the sensors to operate with their associated code including examples provided. This provided a bit of a delay with which made it difficult to assess the reliability and limitations of each sensor individually. As such, of the three colour sensors received, one was found to output values which did not correlate to actual RGB coordinates of a RGB colour chart. While the remaining two colour sensors did give much more accurate ranges (not necessarily values), their ability to supply repeatable reliable readings were questionable at most. This was found to be due to lighting present and the displacement of the sensor to the surface under measurement. Larger displacements allowed better lighting from the source white LEDs but decreased the ability of the surface to be detected by the optical cavity, whilst the shorter displacements tended to provide better ranges of value but decreasing the actual accuracy of the colour detected as the surface would be washed out by the incident white light. To solve this problem, a sufficient distance was determined which balanced either problem and the code written was set for an inequality comparison, that is, using greater than or less than compares. Another problem encountered was that the sensor output was greatly influence by the ambient light intensity. This cause the sensor to become uncalibrated when the ambient light intensity increases or decreases significantly. This in turn causes the machine to fluctuate in terms of its sorting accuracy. No reasonable solution was found for this problem.

Additionally, there was a problem encountered with the load cells. During procurement, it was not specified at the manufacturer's end that the system was only part of a full cell. As such, it was attempted to reengineer the remainder of the full H-bridge setup by matching the resistance values required. However, this introduced a lot of swings in the measurement especially noise most often swung by micro fluctuations in temperature. When attempting to reduce this noise by filtering, the issue which arose was that the accuracy was greatly affected. However, the sensors underwent an attempted calibration, and the smallest unit of measure possible was 1lb in step of 1lb.

The final issue which was encountered was that of dispensed peppers not being near the sensors. To solve this issue, a pair of 'funnelling' partitions were shaped and suspended over the conveyor.

6 Works Cited

Encyclopaedia Britannica. (2015, October 29). *Coloration - Biology*. Retrieved April 12, 2016, from Encyclopaedia Britannica: http://www.britannica.com/science/coloration-biology

7 APPENDIX A

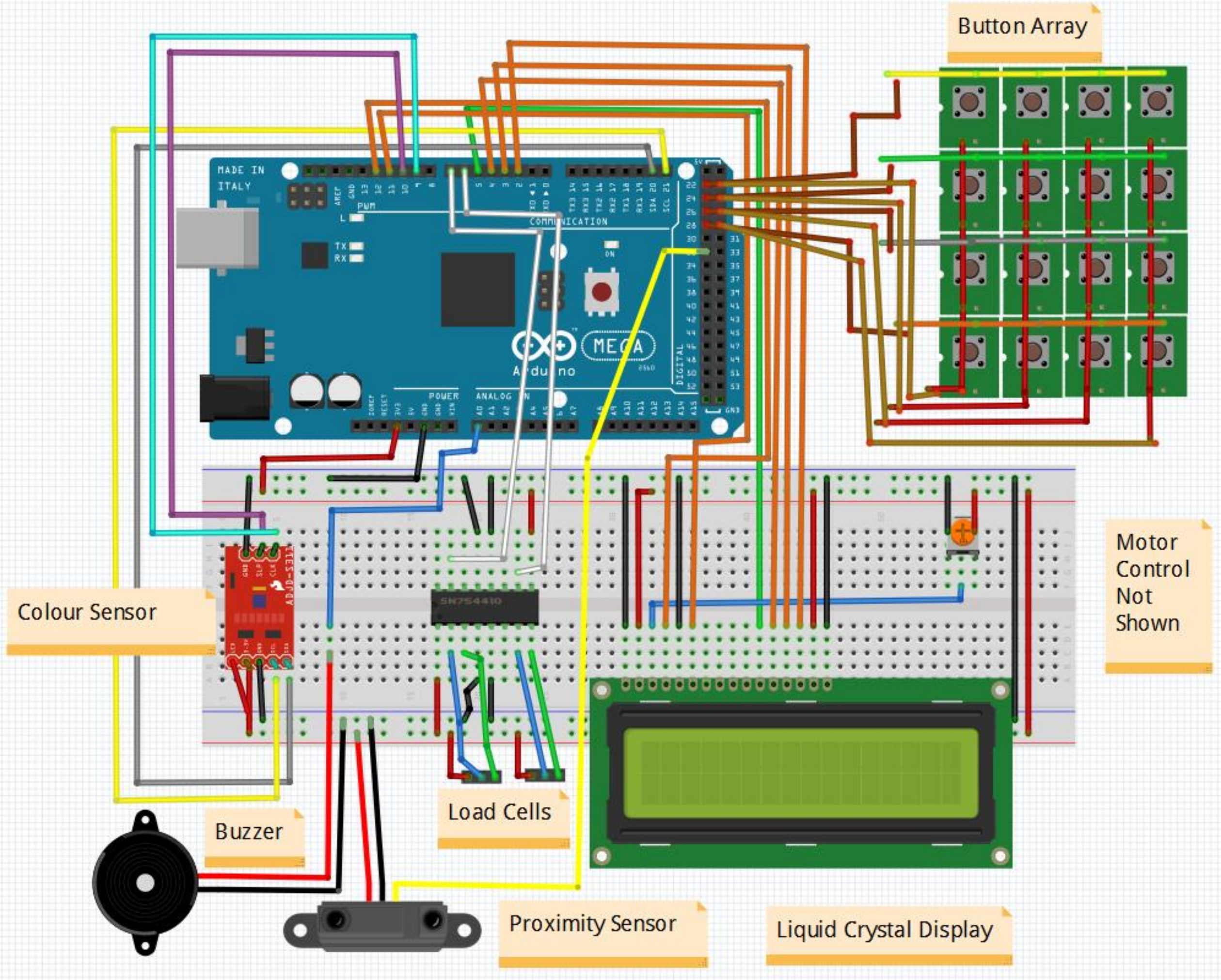

Figure 3: Fritzing Diagram of the Circuit Used, not including Motor Control

8 APPENDIX B

```
//Title: Pepper Sorting Machine Program Code
//
//Load external libraries
#include <LiquidCrystal.h>
#include <LCDKeypad.h>
#include <Keypad.h>
#include <HX711.h>
#include <SignalFilter.h>
#include <Servo.h>

//Load local libraries
#include "pitches.h"
#include "config.h"


/* ----------- Initialise System ----------------*/
void setup() {
  // Initialize the LCD
  lcd.begin(16, 2);
  Serial.begin(115200);

  pinMode(BELTMOTOR, OUTPUT);
  pinMode(DISPENSERMOTOR, OUTPUT);
  pinMode(BUZZER, OUTPUT);

  //initialize servo
  selectionServo.attach(SERVO);
  selectionServo.write(80);

  //Show Software welcome message
  lcd.print("PEPPER SORTING");
  lcd.setCursor(0, 1);
  lcd.print("MACHINE V1.0");

  //initialize scale and color sensor
  scaleSetup();
  colorSetup();
  delay(1500);
}


/* ----------- Main Loop ----------------*/
void loop() {

  keypadCheck();    //Run Keypad control function
  lcdInputCheck(); //Run LCD menu keys control function
  mainDisplay();    //Run LCD display control function
  controlLogic();  //Run Main Sorting Control logic

  //Send debug data to serial
  if (systemState.debug) {
    Serial.println();
    Serial.print("Distance:");
    Serial.println( Ultrasonic_GetDist(TRIGGER, ECHO));
    colorSerialOut();
    scaleCheck();
  }
}


/*------ LCD display control function -----*/
void mainDisplay() {

  if (menuValue == '.')
  {
    if (millis() - timer.t1 >= 4000)
    {//Display state control timer
      timer.t1 = millis();
      systemState.displayFlip = !systemState.displayFlip;
    }

    if (systemState.displayFlip)
```

```
    { //If LCD display is in state one

      //Read and filter load cell values for green and red peppers
      ripePepper.curWeight = Filter.run(scale.get_units()); // Ripe pepper
      greenPepper.curWeight = Filter1.run(scale1.get_units()); // Green pepper

      //Format LCD display to show Pepper Weight values
      lcdWrite(0, 0, "Pepper Weight", true);
      lcdWrite(1, 0, "R:", false);
      lcdWrite(1, 2, (String)ripePepper.curWeight, false);
      lcdWrite(1, 5, "lb| ", false);
      lcdWrite(1, 9, "G:", false);
      lcdWrite(1, 11, (String)greenPepper.curWeight, false);
      lcdWrite(1, 14, "lb", false);
    }


    if (!systemState.displayFlip)
    { //If LCD display is in state two

      // Delay to reduce lcd flickering
      delay(20);

      //Format LCD display to show pepper quantity
      lcdWrite(0, 0, "Pepper Quantity", true);
      lcdWrite(1, 0, "R:", false);
      lcdWrite(1, 2, (String)ripePepper.curQuantity, false);
      lcdWrite(1, 6, " | ", false);
      lcdWrite(1, 9, "G:", false);
      lcdWrite(1, 11, (String)greenPepper.curQuantity, false);

    }
  }
}


/*------- Main sorting control logic -----------*/
void controlLogic() {

  //Initialize local variables
  static int counter1;
  const int pulse0 = 20;
  const int delay1 = 300;
  const byte distance = 10;


  if (systemState.systemON)
  { //If user turn on the machine

    //Read ultrasonic sensor
    int detect = Ultrasonic_GetDist(TRIGGER, ECHO);

    //-----Pepper detection logic------//
    if (detect < distance && systemState.dispenserON && systemState.conveyorON)
    { //If the sensor detects a pepper and the dispenser and conveyor are ON

      //Turn off the dispenser motor
      digitalWrite(DISPENSERMOTOR, LOW);

      //Increment items detected counter
      counter1 += 1;
      Serial.println();
      Serial.print("Pepper detected: ");
      Serial.println(counter1);

      //Delay to allow pepper to pass ultrasonic sensor
      delay(delay1);

      //Turn off conveyor
      digitalWrite(BELTMOTOR, LOW);

      //Read and discard color sensor values
      colorSerialOut();
      colorSerialOut();

      //Pulse conveyor and Sample the pepper color
      samplePepperColor(pulse0);
```

```
      samplePepperColor(pulse0);

      //Change system state variables
      systemState.conveyorON  = false;
      systemState.dispenserON = false;
      Serial.println("Conveyor OFF");
      Serial.println("dispenser OFF");

      //Turn off color sensor LED
      digitalWrite(LED, LOW);


    } else if (detect < distance && systemState.conveyorON ||
               detect < distance && systemState.dispenserON)
    { //If pepper is detected and conveyor or dispenser is ON

      //Increment items detected counter
      counter1 += 1;
      Serial.println();
      Serial.print("Pepper detected 2: ");
      Serial.println(counter1);

      //Delay to allow pepper to pass ultrasonic sensor
      delay(delay1);

      //Pulse conveyor and Sample the pepper color
      samplePepperColor(pulse0);
      samplePepperColor(pulse0);

      //Change system state variable
      systemState.conveyorON  = false;
    }

    //-----Pepper Selection logic-------//
    if ( !systemState.conveyorON)
    { //If the conveyor is OFF

      if (color.red > color.green && color.red > color.blue)
      { //If red is the dominant color

        //Set color selection state to 1
        Serial.println("Pepper is Red.");
        color.selection = 1;

      } else if (color.green > color.blue && color.green > color.red)
      { //If green is the dominant color

        //Set color selection state to 2
        Serial.println("Pepper is Green.");
        color.selection = 2;

      } else if (color.red != 0 && color.green != 0 && color.blue != 0)
      { //If color is indeterminate

        //Reset color selection state
        color.selection = 0;

        //Pulse conveyor and resample pepper color
        samplePepperColor(pulse0);
      }

      //Run red and green pepper sorting functions
      selectRedPepper();
      selectGreenPepper();
    }

    //dispenser Control timer
    if (millis() - timer.t2 >= 1000 && !systemState.dispenserON)
    { //If time out interval and dispenser is OFF

      //Reset timer
      timer.t2 = millis();

      //Turn on dispenser
      digitalWrite(DISPENSERMOTOR, HIGH);

      //Change system state
```

```
      systemState.dispenserON = true;
      Serial.println("dispenser ON");

      //Reset selection servo
      servoTurn(selectionServo, 80, 1);
      //Turn off color sensor led
      digitalWrite(LED, LOW);
      delay(200);
    }

    //Run Capacity logic
    checkMenuSettings();
  } else {
    stopSorter();
  }
}


/* Pulse and Sample color function */
void samplePepperColor(int pulse0)
{
  digitalWrite(BELTMOTOR, HIGH);
  delay(pulse0);
  digitalWrite(BELTMOTOR, LOW);
  colorSerialOut();
  colorSerialOut();
  Serial.print("White: ");
  Serial.println(countW);
  timer.t2 = millis();
}


/* Green pepper sorting function */
void selectGreenPepper() {
  if (color.selection == 1 && !systemState.conveyorON)
  { //If a ripe pepper was dectected and conveyor is OFF

    //Reset selection and color values
    color.selection = 0;
    color.red, color.green, color.blue = 0;
    //Increment ripe pepper quantity
    ripePepper.curQuantity += 1;
    //Active selection servo to direct pepper to
    //red pepper bin
    servoTurn(selectionServo, 125, 0);
    delay(200);

    //Turn on conveyor and change system state
    digitalWrite(BELTMOTOR, HIGH);
    systemState.conveyorON  = true;
    Serial.println("Conveyor ON");
    Serial.print("Ripe Pepper selected: ");
    Serial.println(ripePepper.curQuantity);
    //Reset timer
    timer.t2 = millis();
  }
}


/* Red pepper sorting function */
void selectRedPepper() {
  if (color.selection == 2  && !systemState.conveyorON)//
  { //Green pepper

    color.selection = 0;
    color.red, color.green, color.blue = 0;
    greenPepper.curQuantity += 1;
    servoTurn(selectionServo, 45, 0);
    delay(200);

    digitalWrite(BELTMOTOR, HIGH);
    systemState.conveyorON  = true;
    Serial.println("Conveyor ON");
    Serial.print("Green Pepper selected: ");
    Serial.println(greenPepper.curQuantity);
    timer.t2 = millis();
  }
```

```
}


/*---------- Capacity Logic -----------*/
void checkMenuSettings() {
  if (ripePepper.weight > 0 && systemState.conveyorON)
  {//If value was set for red pepper weight and conveyor is ON
    if (ripePepper.curWeight >= ripePepper.weight)
    {//If Ripe pepper target weight is reach

      playSong();
      delay(500);
      stopSorter();
    }
  }

  if (greenPepper.weight > 0 && systemState.conveyorON)
  {//If value was set for green pepper weight and conveyor is ON
    if (greenPepper.curWeight >= greenPepper.weight)
    {//If Green pepper target weight is reach

      playSong();
      delay(500);
      stopSorter();
    }
  }

  if (ripePepper.quantity > 0 && systemState.conveyorON)
  {//If value was set for ripe pepper quantity and conveyor is ON
    if (ripePepper.curQuantity >= ripePepper.quantity)
    {//If Ripe pepper target quantity is reach

      playSong();
      delay(500);
      stopSorter();
    }
  }

  if (greenPepper.quantity > 0 && systemState.conveyorON)
  {//If value was set for green pepper quantity and conveyor is ON
    if (greenPepper.curQuantity >= greenPepper.quantity)
    {//If Green pepper target quantity reach

      playSong();
      delay(500);
      stopSorter();
    }
  }
}


/* Machine Stop function*/
void stopSorter() {
  digitalWrite(BELTMOTOR, LOW);
  digitalWrite(DISPENSERMOTOR, LOW);
  systemState.conveyorON  = false;
  systemState.dispenserON = false;
  systemState.systemON = false;
}


/* Servo turning function */
void servoTurn(Servo servo, int angle, int rate) {
  if (servo.read() <= angle) {
    for (int i = servo.read(); i <= angle; i++) { // turn the servo forward
      servo.write(i);             // turn 1 degree per rate(ms)
      delay(rate);               // delay time control turning speed
    }
  }
  else {
    for (int i = servo.read(); i >= angle; i--) { // turn the servo backwards
      servo.write(i);
      delay(rate);            // control turning speed
    }
  }
}
```

```

/* Shutdown melody function */
void playSong() {

  for (int thisNote = 0; thisNote < 8; thisNote++) {
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(BUZZER, melody[thisNote], noteDuration);

    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);

    noTone(BUZZER);
  }
}


/* Load Sensor/Scale setup function */
void scaleSetup() {

  //Initialise noise filters
  Filter.begin();
  Filter.setFilter('b');
  Filter.setOrder(1);

  Filter1.begin();
  Filter1.setFilter('b');
  Filter1.setOrder(1);

  //Set A channel
  scale.set_gain(64);
  scale1.set_gain(64);

  //Reset the scale to 0
  scale.set_scale();
  scale.tare();
  scale1.set_scale();
  scale1.tare();

  long zero_factor = scale.read_average(); //Get a baseline reading
  Serial.print("Zero factor: ");
  Serial.println(zero_factor);
  scale.set_scale(calibration_factor);
  scale1.set_scale(calibration_factor);
  scale.tare();
}


/* Scale calibration function */
void scaleCheck() {

  //Adjust to this calibration factor
  scale.set_scale(calibration_factor);
  scale1.set_scale(calibration_factor);

  Serial.print("Scale Reading: ");
  float value = scale.read_average(2);
  Serial.print(value, 1);
  //Serial.print(" lbs");
  Serial.print(" calibration_factor: ");
  Serial.print(calibration_factor);
  Serial.println();

  Serial.print("Scale0 reading:\t");
  value = Filter.run(scale.get_units());
  //Serial.print(value, 2);
  Serial.print(scale.get_units(), 2);

  Serial.print("\t| Scale1 reading:\t");
  value = Filter.run(scale1.get_units());
  //Serial.println(value, 2);
  Serial.println(scale1.get_units(), 2);

  if (Serial.available() > 0)
  {
    char temp = Serial.read();
    if (temp == '+' || temp == 'a')
      calibration_factor += 2;
```

```
    else if (temp == '-' || temp == 'z')
      calibration_factor -= 2;
  }
}


/* Ultrasonic measurement function */
double Ultrasonic_GetDist(byte triggerPin, byte echoPin)
{
  long duration, inches, cm;
  double m;
  cm = 0;

  pinMode(echoPin, INPUT);
  pinMode(triggerPin, OUTPUT);

  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);

  duration = pulseIn(echoPin, HIGH, 38000);

  if (duration != 0) {
    // convert the time into a distance
    cm = microsecondsToCentimeters(duration);
  }

  return cm;
}


long microsecondsToCentimeters(long microseconds)
{
  return (microseconds / 29 / 2);
}


/*------- keypad Control and menu display function ----------*/
void keypadCheck()
{
  char key = kpd.getKey();
  const int time = 1500;
  if (key) // Check for a valid key.
  {
    switch (key)
    {
      case '*': //Clear input Value
        if (menuValue == 'A' || menuValue == 'B' ||
            menuValue == 'C' || menuValue == 'D') {
          pos = 0;
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("Enter new Value:");

          memset(inputValue, 0, sizeof(inputValue));
        }
        break;

      case '#': // Enter key
        if (menuValue == 'A' || menuValue == 'B' ||
            menuValue == 'C' || menuValue == 'D') {
          switch (menuValue) {
            case 'A': //
              greenPepper.quantity = 0;
              greenPepper.weight = 0;
              greenPepper.quantity = atoi(inputValue);

              lcdWrite(0, 0, "Value Saved", true);
              lcdWrite(1, 0, (String)greenPepper.quantity, false);
              menuValue = '.';
              delay(time);
              break;

            case 'B': //
              ripePepper.quantity = 0;
```

```
                ripePepper.weight = 0;
                ripePepper.quantity = atoi(inputValue);

                lcdWrite(0, 0, "Value Saved", true);
                lcdWrite(1, 0, (String)ripePepper.quantity, false);
                menuValue = '.';
                delay(time);
                break;

              case 'C': //
                greenPepper.weight = 0;
                greenPepper.quantity = 0;
                greenPepper.weight = atoi(inputValue);

                lcdWrite(0, 0, "Value Saved", true);
                lcdWrite(1, 0, (String)greenPepper.weight, false);
                menuValue = '.';
                delay(time);
                break;

              case 'D': //
                ripePepper.weight = 0;
                ripePepper.quantity = 0;
                ripePepper.weight = atoi(inputValue);

                lcdWrite(0, 0, "Value Saved", true);
                lcdWrite(1, 0, (String)ripePepper.weight, false);
                menuValue = '.';
                delay(time);
                break;

              default:
                pos = 0;
                break;
            }
            break;
          }
          break;

        case 'A': // Set green pepper quantity menu
          menuValue = key;
          pos = 0;
          lcdWrite(0, 0, "Green Pepper", true);
          lcdWrite(1, 0, "Quantity Menu", false);
          delay(time);
          lcdWrite(0, 0, "Enter Quantity:", true);

          memset(inputValue, 0, sizeof(inputValue));
          break;

        case 'B': // Set ripe pepper quantity menu
          menuValue = key;
          pos = 0;
          lcdWrite(0, 0, "Ripe Pepper", true);
          lcdWrite(1, 0, "Quantity Menu", false);
          delay(time);
          lcdWrite(0, 0, "Enter Quantity:", true);

          memset(inputValue, 0, sizeof(inputValue));
          break;

        case 'C': // Set green pepper weight menu
          menuValue = key;
          pos = 0;
          lcdWrite(0, 0, "Green Pepper", true);
          lcdWrite(1, 0, "Weight Menu", false);
          delay(time);
          lcdWrite(0, 0, "Enter Weight:", true);

          memset(inputValue, 0, sizeof(inputValue));
          break;

        case 'D': // Set ripe pepper weight menu
          menuValue = key;
          pos = 0;
          lcdWrite(0, 0, "Ripe Pepper", true);
          lcdWrite(1, 0, "Weight Menu", false);
```

```
        delay(time);
        lcdWrite(0, 0, "Enter Weight:", true);

        memset(inputValue, 0, sizeof(inputValue));
        break;

      default:
        if (menuValue == 'A' || menuValue == 'B' ||
            menuValue == 'C' || menuValue == 'D') {
          inputValue[pos] = key;
          lcd.setCursor(pos, 1);
          lcd.print(inputValue[pos]);

          if (pos < 2) {
            pos ++;
          }
        } else menuValue = '.';
        break;
      }
    }
}


/* Function to format LCD output */
void lcdWrite(byte row, byte col, String value, bool Clear) {
  if (Clear)
    lcd.clear();
  lcd.setCursor(col, row);
  lcd.print(value);
}


/*-------- LCD Menu key function --------------------*/
void lcdInputCheck()
{
  adc_key_in = analogRead(0);    // read the value from the sensor
  key = get_key(adc_key_in);  // convert into key press
  if (key != oldkey)   // if keypress is detected
  {
    delay(50);  // wait for debounce time
    adc_key_in = analogRead(0);    // read the value from the sensor
    key = get_key(adc_key_in);    // convert into key press
    if (key != oldkey)
    {
      oldkey = key;
      if (key >= 0)
      {
        switch (key) {

          case 1:    // up key
            break;

          case 0:  //right key
            break;

          case 2:  //down key
            //Turn ON/OFF system debug state
            systemState.debug = !systemState.debug;
            Serial.print("System Debug: ");
            Serial.println(systemState.debug);
            break;

          case 3: // left key
            //Reset Scale
            scale.tare();
            scale1.tare();
            Serial.println("Scale Reset");
            break;

          case 4: // enter key
            // Turn sorting machine ON/OFF
            systemState.systemON = !systemState.systemON;

            if (systemState.systemON)
            {
              digitalWrite(BELTMOTOR, HIGH);
              digitalWrite(DISPENSERMOTOR, HIGH);
```

```
              systemState.conveyorON  = true;
              systemState.dispenserON = true;

              Serial.print("System State: ");
              Serial.println((String)systemState.systemON);
            }
            break;

          default:
            Serial.println("Key not implemented");
            break;

        }
        Serial.println(msgs[key]);
      }
    }

    delay(60);
  }
}


// Convert ADC value to key number
int get_key(unsigned int input)
{
  int k;
  for (k = 0; k < NUM_KEYS; k++)
  {
    if (input < adc_key_val[k])
    {
      return k;
    }
  }
  if (k >= NUM_KEYS)k = -1;  // No valid key pressed
  return k;
}


/* Color sensor setup fuction */
void colorSetup()
{
  pinMode(s0, OUTPUT);
  pinMode(s1, OUTPUT);
  pinMode(s2, OUTPUT);
  pinMode(s3, OUTPUT);
  pinMode(LED, OUTPUT);

  //Color calibration values
  colorCal.red = 27;       // 29, 21, 20, 40
  colorCal.green = 21;     // 23, 16, 27
  colorCal.blue = 22;      // 25, 17, 28
  //white 24, 16, 22
}


/* Color counter function */
void ISR_INTO()
{
  counter++;
}


/* Color timer function */
void timer2_init(void)
{
  TCCR2A = 0x00;
  TCCR2B = 0x07; //the clock frequency source 1024 points
  TCNT2 = 100;   //10 ms overflow again
  TIMSK2 = 0x01; //allow interrupt
}


/* Color capture function*/
void getColor()
{
  digitalWrite(s1, HIGH);
  digitalWrite(s0, LOW); //LOW
```

```
  flag = 0;
  attachInterrupt(0, ISR_INTO, CHANGE);
  timer2_init();
}

int i = 0;


/* Color sensor Interupt Service Routine */
ISR(TIMER2_OVF_vect)
{ //the timer 2, 10ms interrupt overflow again.
  //Internal overflow interrupt executive function

  TCNT2 = 100;
  flag++;
  if (flag == 1)
  {
    counter = 0;

  }
  else if (flag == 2)
  { //Red sensor
    digitalWrite(LED, HIGH);
    digitalWrite(s2, LOW);
    digitalWrite(s3, LOW);
    countR = counter;
    digitalWrite(s2, HIGH);
    digitalWrite(s3, HIGH);
  }
  else if (flag == 3)
  { //Green sensor
    countG = counter;
    digitalWrite(s2, LOW);
    digitalWrite(s3, HIGH);
  }
  else if (flag == 4)
  { //Blue sensor
    countB = counter;
    digitalWrite(s2, HIGH);
    digitalWrite(s3, HIGH);
  }
  else
  { //White sensor
    countW = counter;
    digitalWrite(s2, LOW);
    digitalWrite(s3, LOW);
    flag = 0;
    TIMSK2 = 0x00;
  }
  counter = 0;
  delay(2);
}


/* Serial Color printing function */
void colorSerialOut()
{
  delay(100);
  getColor();
  caliberateColor();

  Serial.print("Color:,");
  Serial.print(color.red);
  Serial.print(',');
  Serial.print(color.green);
  Serial.print(',');
  Serial.println(color.blue);
}


/* Color values caliberation function */
void caliberateColor()
{
  if (systemState.debug)
  {
    Serial.print("Red: ");
    Serial.println(countR);
```

```
    Serial.print("Green: ");
    Serial.println(countG);
    Serial.print("Blue: ");
    Serial.println(countB);
    //Serial.print("White: ");
    //Serial.println(countW);
  }

  color.red = constrain(map(countR, 6, colorCal.red, 0, 255), 0, 255);
  color.green = constrain(map(countG, 4, colorCal.green, 0, 255), 0, 255);
  color.blue = constrain(map(countB, 4, colorCal.blue, 0, 255), 0, 255);

}

//
// File: config.h
//
/* System Configuration and Global variables */

//Mega pin assignment
#define DOUT  53
#define CLK  52
#define DOUT1 50
#define CLK1 51
#define TRIGGER 35
#define ECHO 36

#define BELTMOTOR 37
#define DISPENSERMOTOR 38
#define SERVO 39

#define BUZZER 40

/* Custom type definition */
struct pepper_t {
  int quantity = 0, weight = 0,
       curQuantity = 0, curWeight = 0;
};

struct color_t {
  int white = 0, red = 0,
      blue = 0, green = 0,
      selection;
};

struct systemState_t {
  bool conveyorON = false,
       dispenserON = false,
       debug = false,
       systemON = false,
       displayFlip =  false;

} systemState;

struct timer_t {
  unsigned long t1 = 0, t2 = 0, t3 = 0;
} timer;
/*--------------*/


// custom pepper type
pepper_t greenPepper;
pepper_t ripePepper;

// custom color type
color_t color;
color_t colorCal;

//Noise filter
SignalFilter Filter;
SignalFilter Filter1;

//Load cell control board
HX711 scale(DOUT, CLK);
HX711 scale1(DOUT1, CLK1);

float calibration_factor = -12339;
```

```
Servo selectionServo;
Servo conveyor;

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};

/*--------- Color Sensor variables ---------------*/
const int s0 = 30;
const int s1 = 31;

const int taosOutPin = 2;
const int s2 = 32;
const int s3 = 33;
const int LED = 34;

int flag = 0;
int counter = 0;
int countR = 0, countG = 0, countB = 0, countW = 0;

/*---------- LCD Key variables --------------*/
char msgs[5][16] = {
  'Right Key OK ',
  'Up Key OK     ',
  'Down Key OK   ',
  'Left Key OK   ',
  'Select Key OK'
};

int adc_key_val[5] = {
  50, 200, 400, 600, 800
};

int NUM_KEYS = 5;
int adc_key_in;
int key = -1;
int oldkey = -1;


/*---------- Keypad variables ----------- */
char inputValue[3];
byte multipler[3] = {100, 10, 1};
char menuValue = '.';
int pos = 0;
const byte ROWS = 4; // Four rows
const byte COLS = 4; // Four columns

// Define the Keymap
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

// keypad ROW0, ROW1, ROW2 and ROW3 Arduino pins configuration.
byte rowPins[ROWS] = { 22, 23, 24, 25 };

// keypad COL0, COL1, COL2 and COL33 Arduino pins configuration.
byte colPins[COLS] = { 26, 27, 28, 29 };

Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
/*-------------------------------------*/

// LCD Arduino pins configuration
LiquidCrystal lcd(8, 13, 9, 4, 5, 6, 7);
```