//\*keypad, keyboard and HX711 libraries start

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <Keypad.h>

#include "HX711.h"

//\*keypad, keyboard and HX711 libraries end

//\*stepper motor 1 pins start

#define PUL\_PIN1 10  // Pulse pin

#define DIR\_PIN1 11  // Direction pin

#define EN\_PIN1 12   // Enable pin

//\*stepper motor 1 pins end

//\*stepper motor 2 pins start

#define PUL\_PIN2 22  // Pulse pin

#define DIR\_PIN2 23  // Direction pin

#define EN\_PIN2 24   // Enable pin

//\*stepper motor 2 pins end

//\*stepper motor 3 pins start

#define PUL\_PIN3 25  // Pulse pin

#define DIR\_PIN3 26  // Direction pin

#define EN\_PIN3 27   // Enable pin

//\*stepper motor 3 pins end

//\*HX711 pins start

#define DOUT1 36

#define CLK1 37

#define DOUT2 38

#define CLK2 39

#define DOUT3 40

#define CLK3 41

//\*HX711 pins end

//\*Ultrasonic sensor start

const int trigPin1 = 30;

const int echoPin1 = 31;

const int trigPin2 = 32;

const int echoPin2 = 33;

const int trigPin3 = 34;

const int echoPin3 = 35;

//\*Ultrasonic sensor end

//\*relay module start

const int relay1 = 44;  // IN1 pin of the relay module

//\*relay module end

//\*display and keypad start

LiquidCrystal\_I2C lcd(0x27, 16, 2);  // Initialize LCD at address 0x27, 16x2

const byte ROWS = 4;  // Four rows

const byte COLS = 4;  // Three columns

char keys[ROWS][COLS] = {

  { '1', '2', '3', 'A' },

  { '4', '5', '6', 'B' },

  { '7', '8', '9', 'C' },

  { '\*', '0', '#', 'D' }

};

byte rowPins[ROWS] = { 5, 4, 3, 2 };

byte colPins[COLS] = { 9, 8, 7, 6 };

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

//\*display and keypad end

//\*HX711 start

HX711 scale;

float calibration\_factor1 = 250000;  // Adjust this value for calibration

float calibration\_factor2 = 250000;  // Adjust this value for calibration

float calibration\_factor3 = 250000;  // Adjust this value for calibration

//\*HX711 end

bool displayReady = 0;

bool interupted = 0;

void setup() {

  //\*display and keypad start

  lcd.init();          // Initialize the LCD

  lcd.backlight();     // Turn on backlight

  lcd.clear();         // Clear the LCD screen

  Serial.begin(9600);  // Start serial communication for debugging

  //\*display and keypad end

  //\*stepper motor 1 start

  pinMode(PUL\_PIN1, OUTPUT);

  pinMode(DIR\_PIN1, OUTPUT);

  pinMode(EN\_PIN1, OUTPUT);

  // Enable the driver

  digitalWrite(EN\_PIN1, LOW);  // LOW to enable

  // Set direction

  digitalWrite(DIR\_PIN1, HIGH);  // Set direction to HIGH for one direction

  //\*stepper motor 1 end

  //\*stepper motor 2 start

  pinMode(PUL\_PIN2, OUTPUT);

  pinMode(DIR\_PIN2, OUTPUT);

  pinMode(EN\_PIN2, OUTPUT);

  // Enable the driver

  digitalWrite(EN\_PIN2, LOW);  // LOW to enable

  // Set direction

  digitalWrite(DIR\_PIN2, HIGH);  // Set direction to HIGH for one direction

  //\*stepper motor 2 end

  //\*stepper motor 3 start

  pinMode(PUL\_PIN3, OUTPUT);

  pinMode(DIR\_PIN3, OUTPUT);

  pinMode(EN\_PIN3, OUTPUT);

  // Enable the driver

  digitalWrite(EN\_PIN3, LOW);  // LOW to enable

  // Set direction

  digitalWrite(DIR\_PIN3, HIGH);  // Set direction to HIGH for one direction

  //\*stepper motor 3 end

  //\*Ultrasonic sensor start

  pinMode(trigPin1, OUTPUT);

  pinMode(echoPin1, INPUT);

  pinMode(trigPin2, OUTPUT);

  pinMode(echoPin2, INPUT);

  pinMode(trigPin3, OUTPUT);

  pinMode(echoPin3, INPUT);

  //\*Ultrasonic sensor end

  //\*relay module start

  // Set the relay pin as an output

  pinMode(relay1, OUTPUT);

  // Initialize the relay to be off

  digitalWrite(relay1, HIGH);

  //\*relay module end

}

void loop() {

  upper\_part();

  lower\_part();

}

//\*function to run motor start

void runMotor(int pulPin) {

  digitalWrite(pulPin, HIGH);

  delayMicroseconds(833);  // Adjust delay for desired speed

  digitalWrite(pulPin, LOW);

  delayMicroseconds(833);  // Adjust delay for desired speed

}

//\*function to run motor end

//\*function for ultrasonic sensor start

long readUltrasonicDistance(int trigPin, int echoPin) {

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);

  long distance = duration \* 0.034 / 2;

  return distance;

}

//\*function for ultrasonic sensor end

void lower\_part() {

  static unsigned long invalidMillis = millis();

  static unsigned long processMillis = millis();

  static unsigned long weightMillis = millis();

  static unsigned long motorMillis = millis();

  enum class lower\_part\_state : uint8\_t {

    IDLE,            // defaults to 0

    INTERUPT,        // defaults to 1

    SIZE\_SELECT,     // defaults to 2

    INVALID,         // defaults to 3

    SIZE\_ENTER,      // defaults to 4

    PROCESSING,      // defaults to 5

    SIZE\_DELETE,     // defaults to 6

    WEIGHT\_SELECT,   // defaults to 7

    WEIGHT\_ENTER,    // defaults to 8

    WEIGHT\_DELETE,   // defaults to 9

    CONFIRMATION,    // defaults to 10

    LOAD\_CELL\_1\_ON,  // defaults to 11

    LOAD\_CELL\_2\_ON,  // defaults to 12

    LOAD\_CELL\_3\_ON,  // defaults to 13

    MOTOR\_1\_ON,      // defaults to 14

    MOTOR\_2\_ON,      // defaults to 15

    MOTOR\_3\_ON,      // defaults to 16

    MOTOR\_1\_OFF,     // defaults to 17

    MOTOR\_2\_OFF,     // defaults to 18

    MOTOR\_3\_OFF,     // defaults to 19

    PROCESS\_END,     // defaults to 20

  };

  // Keep track of the current State

  static lower\_part\_state currState = lower\_part\_state ::IDLE;

  // Keep track of the previous State

  static lower\_part\_state prevState = lower\_part\_state ::IDLE;

  // Variables for user inputs and weight values

  static String inputText = "";    // String to store user input

  static String motor = "";        // String to store input motor number

  static float weightValue = 0.0;  // float to store input weight value

  static float weight = 0.0;       //float to store current weight

  static long reading = 0;

  static char key = "";

  if (currState == lower\_part\_state ::IDLE) {

    displayReady = 1;

  } else {

    displayReady = 0;

  }

  // Process according to our State Diagram

  switch (currState) {

      // Initial state (or final returned state)

    case lower\_part\_state ::IDLE:

      if (interupted == 1) {

        currState = lower\_part\_state ::INTERUPT;

        break;

      }

      // Waiting for size input

      lcd.setCursor(0, 0);

      lcd.print("Enter size(1/2/3): ");

      lcd.setCursor(0, 1);

      key = keypad.getKey();  // Read keypad input

      if (key != NO\_KEY) {

        currState = lower\_part\_state ::SIZE\_SELECT;

        Serial.println("size key pressed");

      }

      break;

    case lower\_part\_state ::INTERUPT:

      key = keypad.getKey();  // Read keypad input

      interupted = 0;

      if (key != NO\_KEY) {

        switch (key) {

          case '1':

          case '2':

          case '3':

          case '4':

          case '5':

          case '6':

          case '7':

          case '8':

          case '9':

          case '0':

          case '\*':

          case '#':

          case 'A':

          case 'B':

          case 'C':

            break;

          case 'D':

            lcd.clear();  // Clear the entire display

            currState = lower\_part\_state ::IDLE;

            break;

        }

      }

      break;

    case lower\_part\_state ::SIZE\_SELECT:

      switch (key) {

        case '1':

        case '2':

        case '3':

          Serial.println("1/2/3 pressed");

          if (inputText.length() < 1) {  // Limit to 1 characters

            inputText += key;

            lcd.print(inputText);  // Display the entered digits

            currState = lower\_part\_state ::IDLE;

            Serial.println("1/2/3 printed");

          } else {

            lcd.print("not valid");

            Serial.println("not valid printed");

            invalidMillis = millis();

            currState = lower\_part\_state ::INVALID;

            prevState = lower\_part\_state ::SIZE\_SELECT;

          }

          break;

        case '4':

        case '5':

        case '6':

        case '7':

        case '8':

        case '9':

        case '0':

        case 'A':

        case '\*':

        case '#':

          lcd.print("not valid");

          invalidMillis = millis();

          currState = lower\_part\_state ::INVALID;

          prevState = lower\_part\_state ::SIZE\_SELECT;

          break;

        case 'D':

          if (inputText != "") {

            currState = lower\_part\_state ::SIZE\_ENTER;

            Serial.println("enter pressed");

          } else {

            lcd.print("not valid");

            invalidMillis = millis();

            currState = lower\_part\_state ::INVALID;

            prevState = lower\_part\_state ::SIZE\_SELECT;

          }

          break;

        case 'C':

          currState = lower\_part\_state ::SIZE\_DELETE;

          break;

        case 'B':

          currState = lower\_part\_state ::SIZE\_DELETE;

          break;

        default:

          break;

      }

      break;

    case lower\_part\_state ::INVALID:

      if (millis() - invalidMillis >= 1000) {

        lcd.clear();     // Clear the entire display

        inputText = "";  // Clear the input text

        lcd.setCursor(0, 0);

        if (prevState == lower\_part\_state ::SIZE\_SELECT) {

          currState = lower\_part\_state ::IDLE;

        } else if (prevState == lower\_part\_state ::WEIGHT\_SELECT) {

          currState = lower\_part\_state ::WEIGHT\_SELECT;

        } else {

          currState = lower\_part\_state ::CONFIRMATION;

        }

      }

      break;

    case lower\_part\_state ::SIZE\_ENTER:

      motor = inputText;

      lcd.clear();     // Clear the entire display

      inputText = "";  // Clear the input text

      lcd.setCursor(0, 0);

      lcd.print("Processing...");

      processMillis = millis();

      currState = lower\_part\_state ::PROCESSING;

      prevState = lower\_part\_state ::SIZE\_ENTER;

      break;

    case lower\_part\_state ::PROCESSING:

      if (millis() - processMillis >= 1000) {

        lcd.clear();  // Clear the entire display

        lcd.setCursor(0, 0);

        if (prevState == lower\_part\_state ::SIZE\_ENTER) {

          currState = lower\_part\_state ::WEIGHT\_SELECT;

        } else if (prevState == lower\_part\_state ::WEIGHT\_ENTER) {

          currState = lower\_part\_state ::CONFIRMATION;

        } else {

          if (motor == "1") {

            currState = lower\_part\_state ::LOAD\_CELL\_1\_ON;

          } else if (motor == "2") {

            currState = lower\_part\_state ::LOAD\_CELL\_2\_ON;

          } else {

            currState = lower\_part\_state ::LOAD\_CELL\_3\_ON;

          }

        }

      }

      break;

    case lower\_part\_state ::SIZE\_DELETE:

      lcd.clear();     // Clear the entire display

      inputText = "";  // Clear the input text

      lcd.setCursor(0, 0);

      currState = lower\_part\_state ::IDLE;

      break;

    case lower\_part\_state ::WEIGHT\_SELECT:

      // Waiting for weight input

      lcd.setCursor(0, 0);

      lcd.print("Enter weight(kg): ");

      lcd.setCursor(0, 1);

      key = keypad.getKey();  // Read keypad input

      if (key != NO\_KEY) {

        switch (key) {

          case '1':

          case '2':

          case '3':

          case '4':

            if (inputText.length() < 1) {  // Limit to 1 characters

              inputText += key;

              lcd.print(inputText);  // Display the entered digits

            } else {

              lcd.print("not valid");

              invalidMillis = millis();

              currState = lower\_part\_state ::INVALID;

              prevState = lower\_part\_state ::WEIGHT\_SELECT;

            }

            break;

          case '5':

          case '6':

          case '7':

          case '8':

          case '9':

          case '0':

          case 'A':

          case '\*':

          case '#':

            lcd.print("not valid");

            invalidMillis = millis();

            currState = lower\_part\_state ::INVALID;

            prevState = lower\_part\_state ::WEIGHT\_SELECT;

            break;

          case 'D':

            currState = lower\_part\_state ::WEIGHT\_ENTER;

            prevState = lower\_part\_state ::WEIGHT\_SELECT;

            break;

          case 'C':

            currState = lower\_part\_state ::WEIGHT\_DELETE;

            prevState = lower\_part\_state ::WEIGHT\_SELECT;

            break;

          case 'B':

            currState = lower\_part\_state ::SIZE\_DELETE;

            break;

          default:

            break;

        }

      }

      break;

    case lower\_part\_state ::WEIGHT\_ENTER:

      Serial.print("inputText: ");

      Serial.println(inputText);

      weightValue = inputText.toFloat() \* 100;

      Serial.print("weight value: ");

      Serial.print(weightValue);

      Serial.println();

      lcd.clear();     // Clear the entire display

      inputText = "";  // Clear the input text

      lcd.setCursor(0, 0);

      lcd.print("Processing...");

      processMillis = millis();

      currState = lower\_part\_state ::PROCESSING;

      prevState = lower\_part\_state ::WEIGHT\_ENTER;

      break;

    case lower\_part\_state ::WEIGHT\_DELETE:

      lcd.clear();     // Clear the entire display

      inputText = "";  // Clear the input text

      lcd.setCursor(0, 0);

      currState = lower\_part\_state ::WEIGHT\_SELECT;

      break;

    case lower\_part\_state ::CONFIRMATION:

      // Waiting for weight input

      lcd.setCursor(0, 0);

      lcd.print("size: ");

      lcd.print(motor);

      lcd.setCursor(0, 1);

      lcd.print("weight: ");

      lcd.print(weightValue);

      key = keypad.getKey();  // Read keypad input

      if (key != NO\_KEY) {

        switch (key) {

          case 'D':

            lcd.clear();     // Clear the entire display

            inputText = "";  // Clear the input text

            lcd.setCursor(0, 0);

            lcd.print("Processing...");

            processMillis = millis();

            currState = lower\_part\_state ::PROCESSING;

            prevState = lower\_part\_state ::CONFIRMATION;

            break;

          case 'B':

            currState = lower\_part\_state ::SIZE\_DELETE;

            break;

          case '1':

          case '2':

          case '3':

          case '4':

          case '5':

          case '6':

          case '7':

          case '8':

          case '9':

          case '0':

          case 'A':

          case 'c':

          case '\*':

          case '#':

            lcd.print("not valid");

            invalidMillis = millis();

            currState = lower\_part\_state ::INVALID;

            prevState = lower\_part\_state ::CONFIRMATION;

            break;

          default:

            break;

        }

      }

      break;

    case lower\_part\_state ::LOAD\_CELL\_1\_ON:

      //\*HX711 start

      Serial.println("HX711 calibration sketch");

      Serial.println("Remove all weight from scale");

      Serial.println("After readings begin, place known weight on scale");

      Serial.println("Press + or a to increase calibration factor");

      Serial.println("Press - or z to decrease calibration factor");

      scale.begin(DOUT1, CLK1);

      scale.set\_scale(calibration\_factor1);

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

      static long zero\_factor1 = scale.read\_average();  // Get a baseline reading

      // Set the scale to this zero factor

      scale.set\_offset(zero\_factor1);  // Zero out the scale

      Serial.print("Zero factor: ");   // This can be used to remove the need to tare the scale. Useful in permanent scale projects.

      Serial.println(zero\_factor1);

      weightMillis = millis();

      motorMillis = millis();

        lcd.setCursor(0, 0);

        lcd.print("    wait");

        lcd.setCursor(0, 1);

        lcd.print("weighing began");

        currState = lower\_part\_state ::MOTOR\_1\_ON;

        Serial.println("motor 1 selected");

      break;

      //\*HX711 end

    case lower\_part\_state ::LOAD\_CELL\_2\_ON:

      //\*HX711 start

      Serial.println("HX711 calibration sketch");

      Serial.println("Remove all weight from scale");

      Serial.println("After readings begin, place known weight on scale");

      Serial.println("Press + or a to increase calibration factor");

      Serial.println("Press - or z to decrease calibration factor");

      scale.begin(DOUT2, CLK2);

      scale.set\_scale(calibration\_factor2);

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

      static long zero\_factor2 = scale.read\_average();  // Get a baseline reading

      // Set the scale to this zero factor

      scale.set\_offset(zero\_factor2);  // Zero out the scale

      Serial.print("Zero factor: ");   // This can be used to remove the need to tare the scale. Useful in permanent scale projects.

      Serial.println(zero\_factor2);

      weightMillis = millis();

      motorMillis = millis();

        lcd.setCursor(0, 0);

        lcd.print("    wait");

        lcd.setCursor(0, 1);

        lcd.print("weighing began");

        currState = lower\_part\_state ::MOTOR\_2\_ON;

        Serial.println("motor 2 selected");

      break;

      //\*HX711 end

    case lower\_part\_state ::LOAD\_CELL\_3\_ON:

      //\*HX711 start

      Serial.println("HX711 calibration sketch");

      Serial.println("Remove all weight from scale");

      Serial.println("After readings begin, place known weight on scale");

      Serial.println("Press + or a to increase calibration factor");

      Serial.println("Press - or z to decrease calibration factor");

      scale.begin(DOUT3, CLK3);

      scale.set\_scale(calibration\_factor3);

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

      static long zero\_factor3 = scale.read\_average();  // Get a baseline reading

      // Set the scale to this zero factor

      scale.set\_offset(zero\_factor3);  // Zero out the scale

      Serial.print("Zero factor: ");   // This can be used to remove the need to tare the scale. Useful in permanent scale projects.

      Serial.println(zero\_factor3);

      weightMillis = millis();

      motorMillis = millis();

        lcd.setCursor(0, 0);

        lcd.print("    wait");

        lcd.setCursor(0, 1);

        lcd.print("weighing began");

        currState = lower\_part\_state ::MOTOR\_3\_ON;

        Serial.println("motor 3 selected");

      break;

      //\*HX711 end

    case lower\_part\_state ::MOTOR\_1\_ON:

      //\*HX711 start

      if (millis() - weightMillis >= 1000) {

        scale.set\_scale(calibration\_factor);  // Adjust to this calibration factor

        weight = scale.get\_units();

        if (weight < 0) {

          weight = 0.00;

        }

        Serial.print("Reading: ");

        Serial.print("Grams: ");

        weight = weight \* 1000;  // Convert kg to grams

        Serial.print(weight);

        Serial.print(" g");

        Serial.print(" calibration\_factor: ");

        Serial.print(calibration\_factor);

        reading = scale.read();

        Serial.print(" Raw reading: ");

        Serial.print(reading);

        Serial.println();

        weightMillis = millis();

      }

      //\*HX711 end

      //\*motor 1 start

      if (millis() - motorMillis >= 2000) {

        runMotor(PUL\_PIN1);

      }

      //\*motor 1 end

      if (millis() - motorMillis >= 2500) {

        if (weight >= weightValue) {

          lcd.clear();  // Clear the entire display

          currState = lower\_part\_state ::MOTOR\_1\_OFF;

          Serial.println("moving to motor 1 off");

        }

      }

      break;

    case lower\_part\_state ::MOTOR\_2\_ON:

      //\*HX711 start

      if (millis() - weightMillis >= 1000) {

        scale.set\_scale(calibration\_factor);  // Adjust to this calibration factor

        weight = scale.get\_units();

        if (weight < 0) {

          weight = 0.00;

        }

        Serial.print("Reading: ");

        Serial.print("Grams: ");

        weight = weight \* 1000;  // Convert kg to grams

        Serial.print(weight);

        Serial.print(" g");

        Serial.print(" calibration\_factor: ");

        Serial.print(calibration\_factor);

        Serial.print("weight value: ");

        Serial.print(weightValue);

        Serial.println();

        weightMillis = millis();

      }

      //\*HX711 end

      //\*motor 2 start

      if (millis() - motorMillis >= 2000) {

        runMotor(PUL\_PIN2);

      }

      //\*motor 2 end

      if (millis() - motorMillis >= 2500) {

        if (weight >= weightValue) {

          lcd.clear();  // Clear the entire display

          currState = lower\_part\_state ::MOTOR\_2\_OFF;

          Serial.println("moving to motor 2 off");

        }

      }

      break;

    case lower\_part\_state ::MOTOR\_3\_ON:

      //\*HX711 start

      if (millis() - weightMillis >= 1000) {

        scale.set\_scale(calibration\_factor);  // Adjust to this calibration factor

        weight = scale.get\_units();

        if (weight < 0) {

          weight = 0.00;

        }

        Serial.print("Reading: ");

        Serial.print("Grams: ");

        weight = weight \* 1000;  // Convert kg to grams

        Serial.print(weight);

        Serial.print(" g");

        Serial.print(" calibration\_factor: ");

        Serial.print(calibration\_factor);

        Serial.print("weight value: ");

        Serial.print(weightValue);

        Serial.println();

        weightMillis = millis();

      }

      //\*HX711 end

      //\*motor 3 start

      if (millis() - motorMillis >= 2000) {

        runMotor(PUL\_PIN3);

      }

      //\*motor 3 end

      if (millis() - motorMillis >= 2500) {

        if (weight >= weightValue) {

          lcd.clear();  // Clear the entire display

          currState = lower\_part\_state ::MOTOR\_3\_OFF;

          Serial.println("moving to motor 3 off");

        }

      }

      break;

    case lower\_part\_state ::MOTOR\_1\_OFF:

      digitalWrite(EN\_PIN1, HIGH);

      currState = lower\_part\_state ::PROCESS\_END;

      Serial.println("moving to process end");

      break;

    case lower\_part\_state ::MOTOR\_2\_OFF:

      digitalWrite(EN\_PIN2, HIGH);

      currState = lower\_part\_state ::PROCESS\_END;

      break;

    case lower\_part\_state ::MOTOR\_3\_OFF:

      digitalWrite(EN\_PIN3, HIGH);

      currState = lower\_part\_state ::PROCESS\_END;

      break;

    case lower\_part\_state ::PROCESS\_END:

      if (millis() - motorMillis >= 1000) {

        Serial.println("precess end");

        motorMillis = millis();

      }

      lcd.setCursor(0, 0);

      lcd.print("  please remove");

      lcd.setCursor(0, 1);

      lcd.print("     weight");

      //\*HX711 start

      if (millis() - weightMillis >= 1000) {

        scale.set\_scale(calibration\_factor);  // Adjust to this calibration factor

        weight = scale.get\_units();

        if (weight < 0) {

          weight = 0.00;

        }

        Serial.print("Reading: ");

        Serial.print("Grams: ");

        weight = weight \* 1000;  // Convert kg to grams

        Serial.print(weight);

        Serial.print(" g");

        Serial.print(" calibration\_factor: ");

        Serial.print(calibration\_factor);

        Serial.print("weight value: ");

        Serial.print(weightValue);

        Serial.println();

        weightMillis = millis();

      }

      //\*HX711 end

      if (weight <= 10.0) {

        Serial.println("restart");

        lcd.clear();  // Clear the entire display

        currState = lower\_part\_state ::IDLE;

      }

      digitalWrite(EN\_PIN1, LOW);  // LOW to enable

      digitalWrite(EN\_PIN2, LOW);  // LOW to enable

      digitalWrite(EN\_PIN3, LOW);  // LOW to enable

      break;

    default:

      Serial.println("'Default' Switch Case reached - Error");

  }

}

void upper\_part() {

  static unsigned long sensor1Millis = millis();

  static unsigned long sensor2Millis = millis();

  static unsigned long sensor3Millis = millis();

  enum class upper\_part\_state : uint8\_t {

    NORMAL,             // defaults to 0

    CONTAINER\_1\_FULL,   // defaults to 1

    CONTAINER\_2\_FULL,   // defaults to 2

    CONTAINER\_3\_FULL,   // defaults to 3

    CONTAINER\_1\_EMPTY,  // defaults to 4

    CONTAINER\_2\_EMPTY,  // defaults to 5

    CONTAINER\_3\_EMPTY,  // defaults to 6

  };

  // Keep track of the current State

  static upper\_part\_state currState = upper\_part\_state ::NORMAL;

  long distance1 = readUltrasonicDistance(trigPin1, echoPin1);

  long distance2 = readUltrasonicDistance(trigPin2, echoPin2);

  long distance3 = readUltrasonicDistance(trigPin3, echoPin3);

  switch (currState) {

    case upper\_part\_state ::NORMAL:

      if (millis() - sensor1Millis >= 1000) {

        Serial.print("Sensor 1: ");

        Serial.print(distance1);

        Serial.print(" cm, ");

        sensor1Millis = millis();

      }

      if (millis() - sensor2Millis >= 1000) {

        Serial.print("Sensor 2: ");

        Serial.print(distance2);

        Serial.print(" cm, ");

        sensor2Millis = millis();

      }

      if (millis() - sensor3Millis >= 1000) {

        Serial.print("Sensor 3: ");

        Serial.print(distance3);

        Serial.println(" cm");

        sensor3Millis = millis();

      }

      // Check if sensor 1 reads below 10 cm

      if (distance1 < 10) {

        currState = upper\_part\_state ::CONTAINER\_1\_FULL;

      }

      // Check if sensor 2 reads below 10 cm

      if (distance2 < 10) {

        currState = upper\_part\_state ::CONTAINER\_2\_FULL;

      }

      // Check if sensor 3 reads below 10 cm

      if (distance3 < 10) {

        currState = upper\_part\_state ::CONTAINER\_3\_FULL;

      }

      break;

    case upper\_part\_state ::CONTAINER\_1\_FULL:

      // Stop the motor

      digitalWrite(relay1, LOW);

      Serial.println("Motor 1 stopped");

      interupted = 1;

      if (displayReady == 1) {

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("  container 1");

        lcd.setCursor(0, 1);

        lcd.print("    is full");

        currState = upper\_part\_state ::CONTAINER\_1\_EMPTY;

      }

      break;

    case upper\_part\_state ::CONTAINER\_2\_FULL:

      // Stop the motor

      digitalWrite(relay1, LOW);

      Serial.println("Motor 2 stopped");

      interupted = 1;

      if (displayReady == 1) {

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("  container 2");

        lcd.setCursor(0, 1);

        lcd.print("    is full");

        currState = upper\_part\_state ::CONTAINER\_2\_EMPTY;

      }

      break;

    case upper\_part\_state ::CONTAINER\_3\_FULL:

      // Stop the motor

      digitalWrite(relay1, LOW);

      Serial.println("Motor 3 stopped");

      interupted = 1;

      if (displayReady == 1) {

        lcd.clear();

        lcd.setCursor(0, 0);

        lcd.print("  container 3");

        lcd.setCursor(0, 1);

        lcd.print("    is full");

        currState = upper\_part\_state ::CONTAINER\_3\_EMPTY;

      }

      break;

    case upper\_part\_state ::CONTAINER\_1\_EMPTY:

      if (millis() - sensor1Millis >= 1000) {

        Serial.print("Sensor 1: ");

        Serial.print(distance1);

        Serial.println(" cm, ");

        sensor1Millis = millis();

      }

      // Check if sensor 1 reads below 10 cm

      if (distance1 > 25) {

        digitalWrite(relay1, HIGH);

        currState = upper\_part\_state ::NORMAL;

      }

      break;

    case upper\_part\_state ::CONTAINER\_2\_EMPTY:

      if (millis() - sensor2Millis >= 1000) {

        Serial.print("Sensor 2: ");

        Serial.print(distance2);

        Serial.println(" cm, ");

        sensor2Millis = millis();

      }

      // Check if sensor 1 reads below 10 cm

      if (distance2 > 25) {

        digitalWrite(relay1, HIGH);

        currState = upper\_part\_state ::NORMAL;

      }

      break;

    case upper\_part\_state ::CONTAINER\_3\_EMPTY:

      if (millis() - sensor3Millis >= 1000) {

        Serial.print("Sensor 3: ");

        Serial.print(distance3);

        Serial.println(" cm, ");

        sensor3Millis = millis();

      }

      // Check if sensor 1 reads below 10 cm

      if (distance3 > 25) {

        digitalWrite(relay1, HIGH);

        currState = upper\_part\_state ::NORMAL;

      }

      break;

    default:

      Serial.println("'Default' Switch Case reached - Error");

  }