The Ultimate Cat Feeder

By Mahsood Ebrahim

Description

My siblings and I are responsible for feeding our cat, Pumpkin. At times, in addition to my lethargy, feeding him becomes a nuisance. To relieve this plight and tap into my laziness, I have decided to create a cat feeder. While one is able to purchase this type of device from the many available products that exist within the market today, I wanted the flexibility and customizability of creating a product that fits my needs, and more. I designed the system so that I am able to dispense food from any location as long as I am connected to data/wifi. Also, for my own entertainment, the device plays music while dispensing food and is capable of driving around. All of the functionality built into the cat - feeder is accessible from a smartphone.

Figure 1. Cat Feeder

Electromechanical Details

Interfacing with the Cat Feeder

Food is placed into the green funnel which travels downward and is stopped by a spiral propeller hidden inside the system. The stepper motor located on the rear-end of the device rotates the propeller. While driving, the front and rear range sensors will automatically stop the

system if the device is within 10 cm of an object. The range sensor, located at the top of the funnel, is used to determine the remaining percentage of food stored.

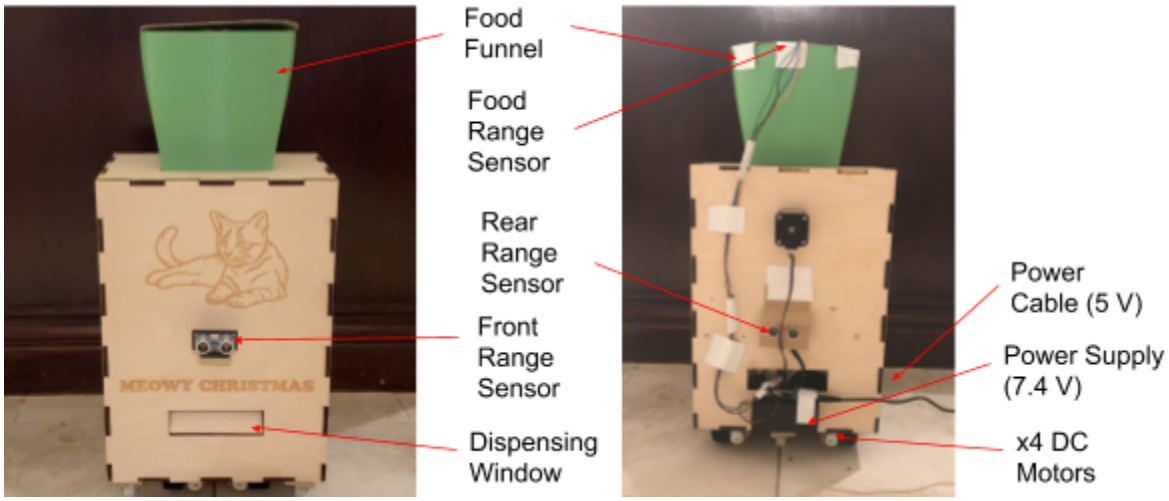

Figure 2. Cat Feeder Interface

Control

The control mechanism is simply a smartphone and connection to data/wifi. Using the Telegram app, a message is sent to a Bot on the Telegram servers. Thereafter, the bot sends the message from the user to the esp32 (connected to wifi). There are two commands that the esp32 will respond to: '/drive' and '/feed'. Both commands will respond with inline buttons that the user is able to depress to direct the system (displayed below). The 'DISPENSE FOOD' command will distribute food from the front window while the 'REMAINING FOOD STATUS' command checks and reports the percentage of food within the funnel. The 'FRONT', 'STOP', 'LEFT', 'RIGHT', and 'BACK' commands move the device in those directions, respectively.

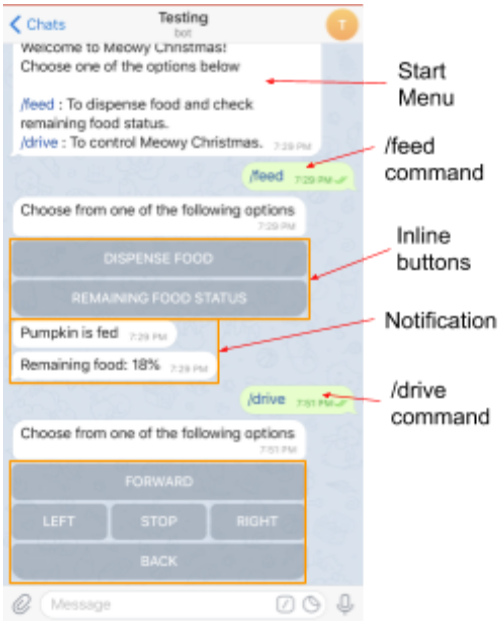

Figure 3. Control Mechanism

Circuit

The difficulty of the project was determining how to control the stepper motor and ensure the microcontrollers communicated with each other. The project uses three microcontrollers: one parent ((1) ESP32) and two children ((2) Arduino Uno & (3) ESP32). Communication between the devices was achieved using the UART protocol.

1. Parent ESP32: The head microcontroller is connected to wifi and is the device that processes messages from the Telegram Bot. It is also responsible for food range sensor and activating the stepper motor through the L298n motor driver. It is powered from an external 5 volt power supply that connects one end to the USB pin on the controller, and the other pin to ground. The motor driver is powered from a 7.4 volt (2200mah) battery, while the range sensors are powered by the 5v pin from the Arduino uno.
2. Child Arduino Uno: This microcontroller is responsible for controlling the four DC motors (using the L298n motor driver) connected to the wheels of the device, and the front and rear range sensors for obstacle avoidance. The Arduino board receives one - way messages via serial data from the parent ESP32. It is powered by the 7.4 volt battery through the Vin and Gnd pins on the board. Given the Arduino pins output 5 volts, a voltage divider was used after the Rx pin (Arduino) and before the Tx pin (on the ESP32 parent) such that the ESP32 received 3.3 volts.
3. Child ESP32: The child ESP32 is solely responsible for playing music while the food is dispensing. The ESP32 is used for playing music due to its large storage capacity; it is capable of playing approximately 8 seconds of a song without the need of external storage. Music is produced from this microcontroller using the PAM8302 audio amplifier and a speaker (8 ohm, 1 watt); the speaker is powered from the ESP32's 3.3 volt output pin. Similar to the Uno, this board receives one - way messages via serial data from the parent ESP32. This controller is powered in the same manner as the parent ESP32.

Finite State Machine

There are four states within the finite state machine: idling, checking food status, engaging DC motors, and playing music and engaging stepper motor. The system is primarily within the idling state as this is where the parent ESP32 waits until the Telegram bot sends a message from the user. Depending on what message the parent microcontroller receives dictates which state the system transitions to next. Notice, however, despite what state the system is in it quickly returns to the idling state awaiting the next input.

Additional figures of the device are displaying in Appendix A. For the complete Arduino IDE code, see Appendix B.

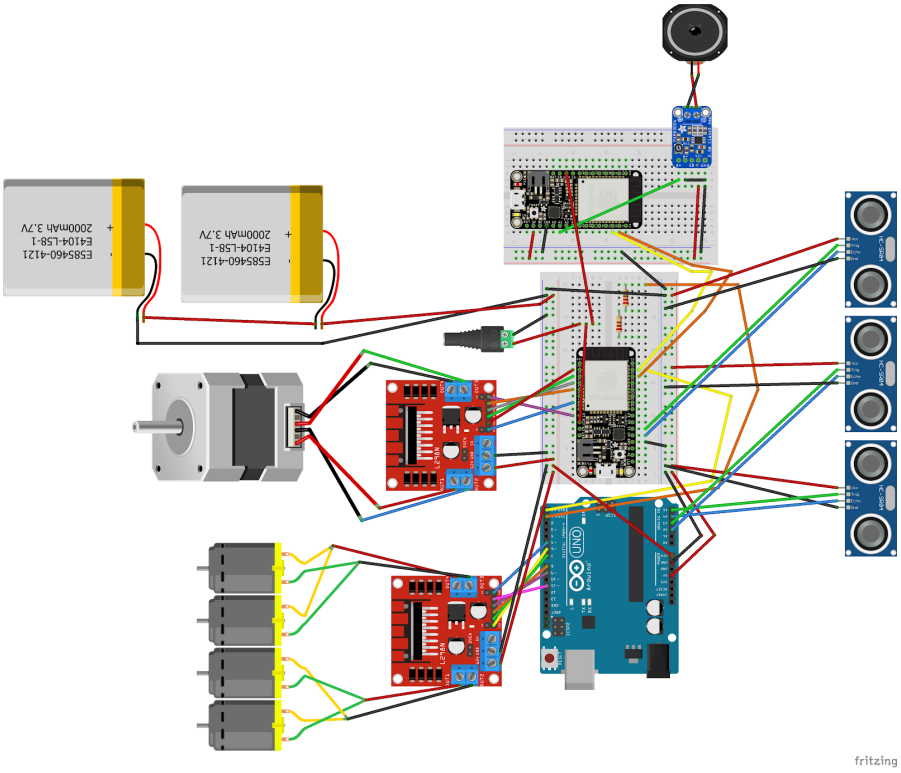

Figure 4. Circuit Diagram

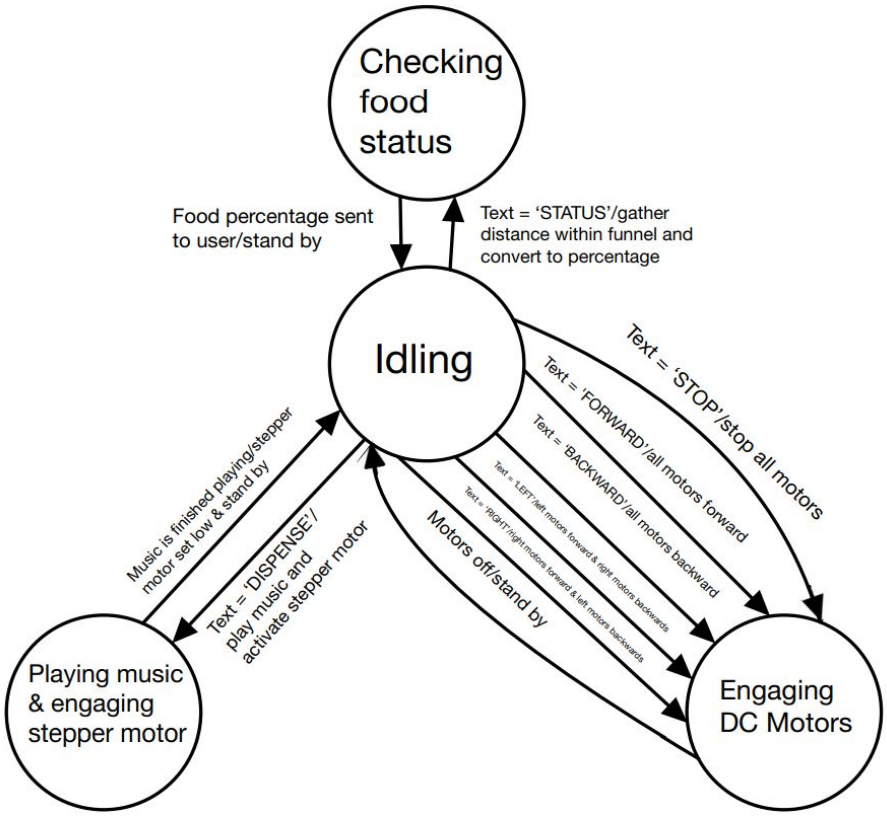

Figure 5. Finite State Machine

Appendix A

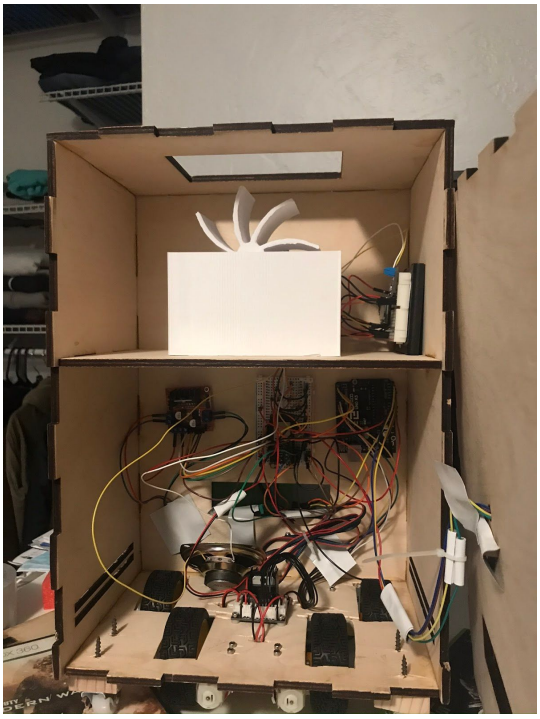

Figure 6. Interior Wiring

Figure 7. Interior Assembly

Appendix B

ESP32 Parent

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>

// Initialize Wifi connection to the router
char ssid[] = "Frontier5328";     // your network SSID (name)
char password[] = "7975320169";   // your network key

// Initialize Telegram BOT
#define BOTtoken "1254992782:AAE-LH5HB51QA87jUJ03h3iPGY6JpOKRKMY"  // your Bot Token (Get from Botfather)

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);

int Bot_mtbs = 1000; //mean time between scan messages
long Bot_lasttime;   //last time messages' scan has been done
bool Start = false;

////This was for the LED
//const int ledPin = 13;
//int ledStatus = 0;

//// %%%% MOSFET %%%%
//int MosfetGatePin = 4;
//int period = 5000;
//int timeMark;

// %%%% UART COMMUNICATION %%%%
#define RXD2 16
#define TXD2 17

// %%%% RANGE SENSOR %%%%
int Echo = A0;
int Trig = A1;

//Ultrasonic distance measurement Sub function
int getDistance() {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
```

```
  digitalWrite(Trig, LOW);
  return (int)pulseIn(Echo, HIGH) / 58;
}

// %%%% STEPPER MOTOR %%%%
#include <Stepper.h>
int ENA = 12;
int ENB = 13;
int IN1 = 14;
int IN2 = 32;
int IN3 = 15;
int IN4 = 33;
const int stepsPerRevolution = 200;  // steps per revolution for the motor
// initialize the stepper library on pins 14, 32, 15, 33:
Stepper myStepper(stepsPerRevolution, IN1, IN2, IN3, IN4);

void enableMotors() {
  digitalWrite(ENA, HIGH);
  digitalWrite(ENB, HIGH);
}

void disableMotors() {
  digitalWrite(ENA, LOW);
  digitalWrite(ENB, LOW);
}

void stepperMotorOff() { // turn off the stepper motors
  disableMotors();
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
}

void activateStepperMotor() {
  enableMotors();
  myStepper.step(-stepsPerRevolution);
  stepperMotorOff();
}

int cmToPercentage() {
  int foodRemainingPercentage = (-16 * getDistance()) + 176; // y = mx + b
  return foodRemainingPercentage;
}


void handleNewMessages(int numNewMessages) {
  Serial.println("handleNewMessages");
  Serial.println(String(numNewMessages));
```

```
91
92    for (int i = 0; i < numNewMessages; i++) {
93
94      // Inline buttons with callbacks when pressed will raise a callback_query message
95      if (bot.messages[i].type == "callback_query")
96      {
97        String text = bot.messages[i].text;
98        //      Serial.print("Call back button pressed with text: ");
99        //      Serial.println(text);
100       String chat_id = String(bot.messages[i].chat_id);
101
102       if (text == "FORWARD") {
103         Serial2.print("f");
104       }
105       else if (text == "STOP") {
106         Serial2.print("s");
107       }
108       else if (text == "LEFT") {
109         Serial2.print("l");
110       }
111       else if (text == "RIGHT") {
112         Serial2.print("r");
113       }
114       else if (text == "BACK") {
115         Serial2.print("b");
116       }
117       else if (text == "DISPENSE") {
118         //        playSong();
119         Serial2.print("z");
120         activateStepperMotor();
121         bot.sendMessage(chat_id, "Pumpkin is fed", "");
122       }
123       else if (text == "STATUS") {
124         //        String remainFood = "Remaining food: " + String(cmToPercentage()) + "%";
125         String remainFood = "Remaining food: 18%";
126         bot.sendMessage(chat_id, remainFood, "");
127       }
128
129       //      bot.sendMessage(bot.messages[i].from_id, bot.messages[i].text, "");
130     }
131     else {
132
133       String chat_id = String(bot.messages[i].chat_id);
134       String text = bot.messages[i].text;
135
136       String from_name = bot.messages[i].from_name;
137       if (from_name == "") from_name = "Guest";
138
139       if (text == "/drive")
```

```
      {
        String keyboardJson = "[[{ \"text\" : \"FORWARD\", \"callback_data\" : \"FORWARD\" }],[{ \"text\" : \"LEFT\", \"callback_data\" : \"LEFT\" }";
        keyboardJson += ", { \"text\" : \"STOP\", \"callback_data\" : \"STOP\" }, { \"text\" : \"RIGHT\", \"callback_data\" : \"RIGHT\" }]";
        keyboardJson += ",[{ \"text\" : \"BACK\", \"callback_data\" : \"BACK\" }]]";
        bot.sendMessageWithInlineKeyboard(chat_id, "Choose from one of the following options", "", keyboardJson);
      }

      if (text == "/feed")
      {
        String keyboardJson = "[[{ \"text\" : \"DISPENSE FOOD\", \"callback_data\" : \"DISPENSE\" }],[{ \"text\" : \"REMAINING FOOD STATUS\", \"callback_data\" : \"STATUS\" }]]";
        bot.sendMessageWithInlineKeyboard(chat_id, "Choose from one of the following options", "", keyboardJson);
      }

      if (text == "/start") {
        String welcome = "Welcome to Meowy Christmas!\n";
        welcome += "Choose one of the options below\n\n";
        welcome += "/feed : To dispense food and check remaining food status.\n";
        welcome += "/drive : To control Meowy Christmas.\n";
        //        welcome += "/status : Returns current status of LED\n";
        bot.sendMessage(chat_id, welcome, "Markdown");
      }
    }
  }
}


void setup() {
  Serial.begin(115200);

  // Attempt to connect to Wifi network:
  Serial.print("Connecting Wifi: ");
  Serial.println(ssid);

  // Set WiFi to station mode and disconnect from an AP if it was Previously
  // connected
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
```

```
186     Serial.println(WiFi.localIP());
187
188     //  pinMode(ledPin, OUTPUT); // initialize digital ledPin as an output.
189     //  delay(10);
190     //  digitalWrite(ledPin, LOW); // initialize pin as off
191
192     // %%%% STEPPER MOTOR %%%%
193     pinMode(ENB, OUTPUT);
194     pinMode(IN1, OUTPUT);
195     pinMode(IN2, OUTPUT);
196     pinMode(IN3, OUTPUT);
197     pinMode(IN4, OUTPUT);
198     pinMode(ENA, OUTPUT);
199
200     stepperMotorOff(); // turn off stepper motor so that no current runs through motor driver
201
202     // set the speed at 40 rpm:
203     myStepper.setSpeed(40);
204
205     //  // %%%% MOSFET %%%%
206     //  pinMode(MosfetGatePin, OUTPUT);
207     //  digitalWrite(MosfetGatePin, LOW);
208
209     // %%%% RANGE SENSOR %%%%
210     pinMode(Echo, INPUT);
211     pinMode(Trig, OUTPUT);
212
213     // %%%%%%%%%%%%%% SERIAL COMMUNICATION TX & RX %%%%%%%%%%%%%%
214     Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
215   }
216
217   void loop() {
218     if (millis() > Bot_lasttime + Bot_mtbs)  {
219       int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
220
221       while (numNewMessages) {
222         Serial.println("got response");
223         handleNewMessages(numNewMessages);
224         numNewMessages = bot.getUpdates(bot.last_message_received + 1);
225       }
226
227       Bot_lasttime = millis();
228     }
229   }
```

Child Arduino Uno

```
// %%%% RANGE SENSOR %%%%
#define EchoFront A2
#define TrigFront A3
#define EchoRear  A4
#define TrigRear  A5

// %%%% L298N MOTOR DRIVER %%%%
#define ENB 5
#define IN1 7
#define IN2 8
#define IN3 9
#define IN4 11
#define ENA 6
#define carSpeed 200
bool motorForwardFlag = false; // flag for checking distance
bool motorBackwardFlag = false; // flag for checking distance
int distanceToStopAt = 9; // cm
int currentDistance;

// %%%% FORWARD - MOTORS %%%%
// The motors will push the cart forward.
void forward() {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  motorForwardFlag = true;
  //  Serial.println("Forward");
}

// %%%% BACKWARDS - MOTORS %%%%
// The motors will push the cart backwards.
void back() {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  motorBackwardFlag = true;
  //  Serial.println("Back");
}

// %%%% LEFT - MOTORS %%%%
```

```
47  // The motors will turn the cart left.
48  void left() {
49    analogWrite(ENA, carSpeed);
50    analogWrite(ENB, carSpeed);
51    digitalWrite(IN1, LOW);
52    digitalWrite(IN2, HIGH);
53    digitalWrite(IN3, LOW);
54    digitalWrite(IN4, HIGH);
55    //  Serial.println("Left");
56    delay(1500);
57    stop();
58  }
59
60  // %%%% RIGHT - MOTORS %%%%
61  // The motors will turn the cart right.
62  void right() {
63    analogWrite(ENA, carSpeed);
64    analogWrite(ENB, carSpeed);
65    digitalWrite(IN1, HIGH);
66    digitalWrite(IN2, LOW);
67    digitalWrite(IN3, HIGH);
68    digitalWrite(IN4, LOW);
69    delay(1500);
70    stop();
71    //  Serial.println("Right");
72  }
73
74  // %%%% STOP - MOTORS %%%%
75  // The motors will stop the cart.
76  void stop() {
77    digitalWrite(ENA, LOW);
78    digitalWrite(ENB, LOW);
79    motorForwardFlag = false;
80    motorBackwardFlag = false;
81    //    Serial.println("Stop!");
82  }
83
84  // %%%% SERIAL DATA %%%%
85  // This function reads incoming serial data and controls the cart.
86  void getSerialData() {
87    if (Serial.available()) {
88      Serial.println("here");
89      switch (Serial.read()) {
90        case 'f': forward();  break;
91        case 'b': back();     break;
92        case 'l': left();     break;
93        case 'r': right();    break;
94        case 's': stop();     break;
95        default:  break;
```

```
    }
  }
}

// %%%% READING FRONT DISTANCE %%%%
//Ultrasonic distance measurement Sub function
int getFrontDistance() {
  digitalWrite(TrigFront, LOW);
  delayMicroseconds(2);
  digitalWrite(TrigFront, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigFront, LOW);
  return (int)pulseIn(EchoFront, HIGH) / 58;
}

// %%%% READING REAR DISTANCE %%%%
//Ultrasonic distance measurement Sub function
int getRearDistance() {
  digitalWrite(TrigRear, LOW);
  delayMicroseconds(2);
  digitalWrite(TrigRear, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigRear, LOW);
  return (int)pulseIn(EchoRear, HIGH) / 58;
}

// %%%% DISTANCE CHECK %%%%
// This function will stop cart if too close to objects.

void checkDistance() {
//  // If the cart is moving forward and distance is less than 9 cm, stop.
//  if (motorForwardFlag) {
//    currentDistance = getFrontDistance();
//    currentDistance = abs(currentDistance);
//    if (currentDistance < distanceToStopAt) {
//      stop();
//    }
//  }

  // If the cart is moving backward and distance is less than 9 cm, stop.
  if (motorBackwardFlag) {
    currentDistance = getRearDistance();
    currentDistance = abs(currentDistance);
    if (currentDistance < distanceToStopAt) {
      stop();
    }
  }
//Serial.print("Front: "+ String(getFrontDistance()));
//Serial.print(" Rear: " + String(getRearDistance()) + "\n");
```

```
}

void setup() {
  pinMode(EchoFront, INPUT);
  pinMode(TrigFront, OUTPUT);
  pinMode(EchoRear, INPUT);
  pinMode(TrigRear, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  stop();
  Serial.begin(9600);
}

void loop() {
  getSerialData();
  checkDistance();
}
```

Child ESP32

```
#include "ChristmasSong.h"
#include "XT_DAC_Audio.h"

#define RXD2 16
#define TXD2 17
const int period = 1000; // 1 second
int timeMark;            // keep track of reference time

XT_Wav_Class christmasSong(wonderfulTimeSong);     // create an object of type XT_Wav_Class that is used by
// the dac audio class (below), passing wav data as parameter.

XT_DAC_Audio_Class DacAudio(25, 0);   // Create the main player class object.
// Use GPIO 25, one of the 2 DAC pins and timer 0

void setup() {
  Serial.begin(115200);
  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
}

void loop() {
  if (Serial2.available()) {
    Serial.println("here");
    //    Serial.println(char(Serial2.read()));
    switch (char(Serial2.read())) {

      case 'z':
        Serial.println("inside");
        //        Serial.println("z");
        timeMark = millis();              // Keeping track of time allows for entry into the while loop below.
        // Had the issue where -----.TimeLeft was equal to zero at the start,
        // and a do-while loop didn't work either. The solution was the enter the
        // while loop for 1 second before the short-circuit or takes over for the
        // remainder of the song.

        while (christmasSong.TimeLeft != 0 || ((millis() - timeMark) < period)) {
          DacAudio.FillBuffer();          // Fill the sound buffer with data
          if (christmasSong.Playing == false)    // if not playing,
            DacAudio.Play(&christmasSong);       // play the song
        }
        break;
      default:
        break;
    }
  }
}
```