Talk-O-Matic

Tangibly Displaying Conversation

Iris Howley

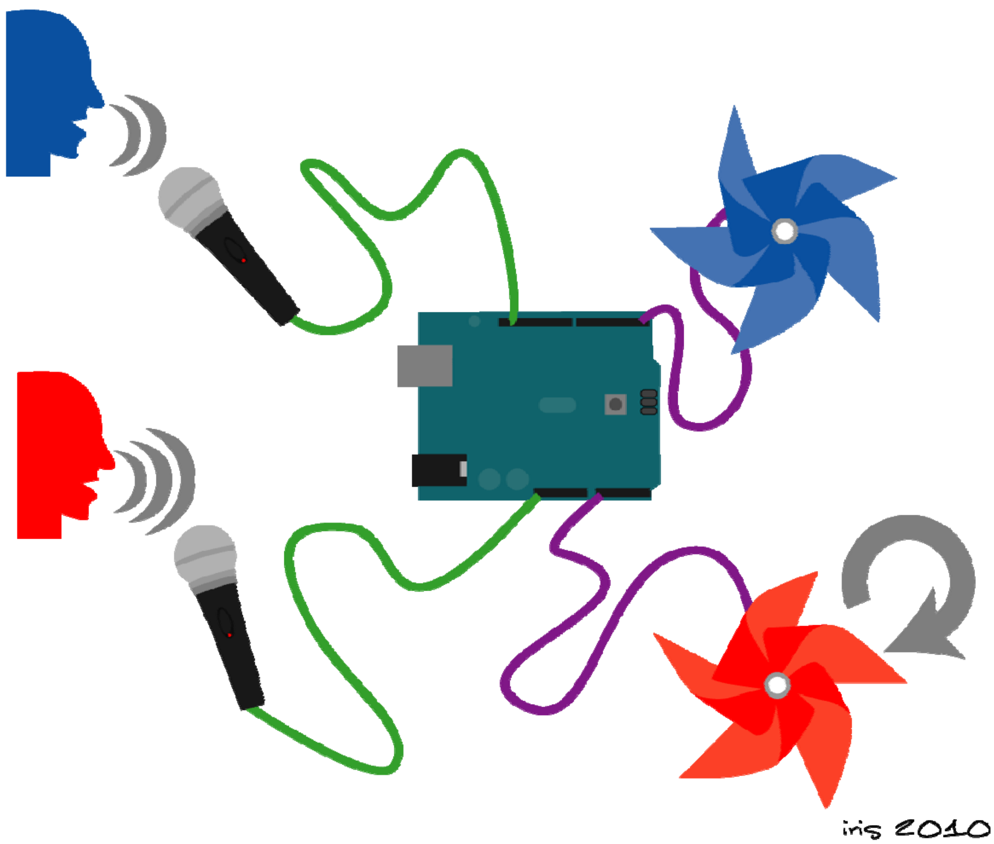
Table of Contents

Motivation

For the classroom teacher who wants to encourage discussion amongst students, the Talk-O-Matic provides a novel and interesting tangible display of conversation that cannot be found anywhere else.

For anyone who has ever been in a meeting where one person dominates the discussion, the Talk-O-Matic offers a visualization to subtly indicate the over-contribution to the dominating-talker.

The Talk-O-Matic takes input from two separate microphones, compares the audio input with an Arduino, and moves the pinwheel gear motor associated with the more active microphone. If input is approximately equal, both pinwheels will move. The Talk-O-Matic also has an on/off button that resets the system.



Challenges

While not having any background in electronics prototyping guaranteed I would have some issues, the greatest challenge I encountered while constructing the Talk-O-Matic was actually related to programming logic.

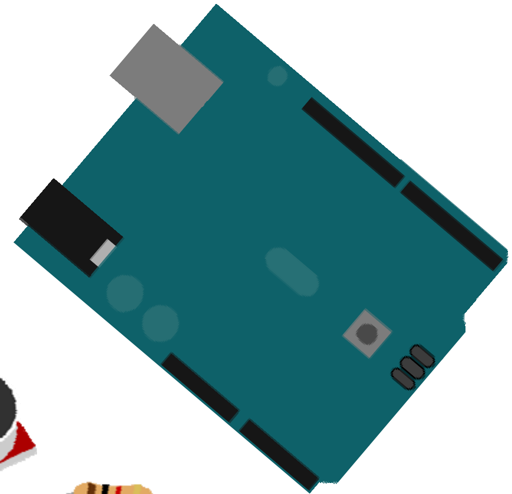
Determining exactly how to know when a microphone is experiencing conversation in a way that can be compared to another microphone proved especially difficult. The Electret Microphone Breakout Boards are rather poor quality and read in a number (typically around 500, but with a maximum of 1023) according to the volume of sound the sensor is experiencing, but with a considerable amount of noise.

To deal with the noise in sensor readings, I created a low pass filter, which adjusts each value read, taking into account past values. This has the effect of smoothing the readings, so that if most of the readings are around '500', a reading of '153' might be readjusted to '350'.

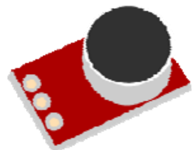
Now, to determine when a microphone is experiencing conversation, first we have to determine what the average values are for "silence." There is a calibration feature built into the code, so that we can determine the average values that are read in by both microphones when the area is at its default quietness. Once we have this average silence value, then we can run the program with its normal features. Every time we read in a value from the microphones, we check to see if it is a certain number of times larger than the average silence value. This way, we can check for abnormally large values being read in, and we count those as "interaction."

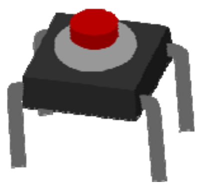
The program keeps a running count of these interaction moments, and then spins the motor for which microphone has more of these moments. When the counts are within 5 of each other, both motors spin.

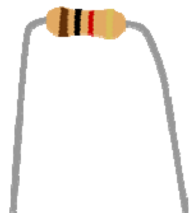
Materials

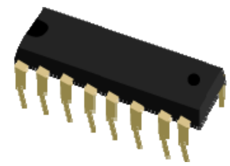


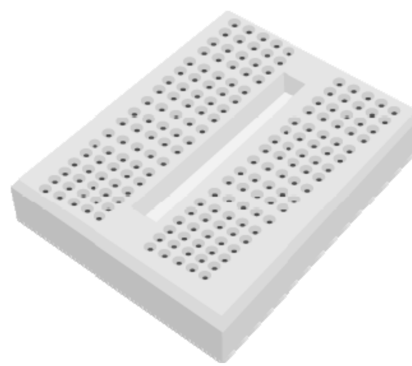
- Arduino Duemilanove
- Electret Microphone Breakout Board (2)
- Pushbutton
- H-bridge (SN754410NE)
- 1k Resistor(3)
- Gear Motor (2)
- Gear Wheels (2)
- Hookup Wire
- Mini Breadboard
- Power Source (Battery Holder, AC Adapter, etc)
- Scrap of Poster Board
- Small Box
- 6-inch Square of Scrap Book Paper (2)
- Craft Wire
- Glue
- Tape
- Scissors
- Wire strippers
- Needle Nose Pliers

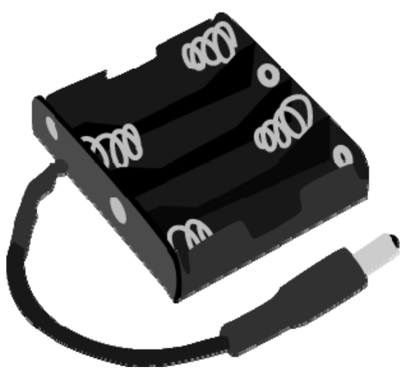


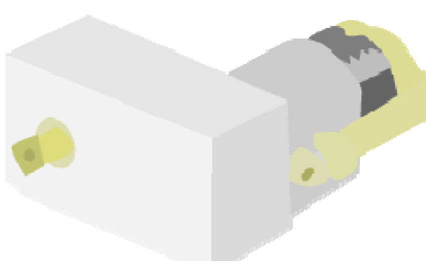


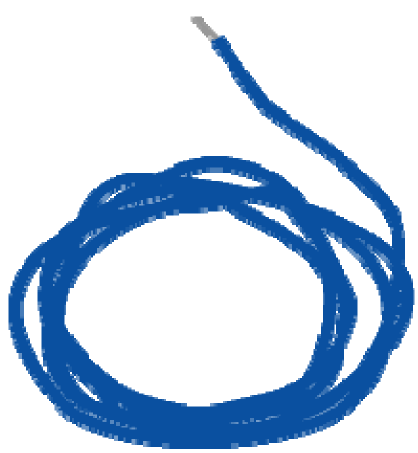




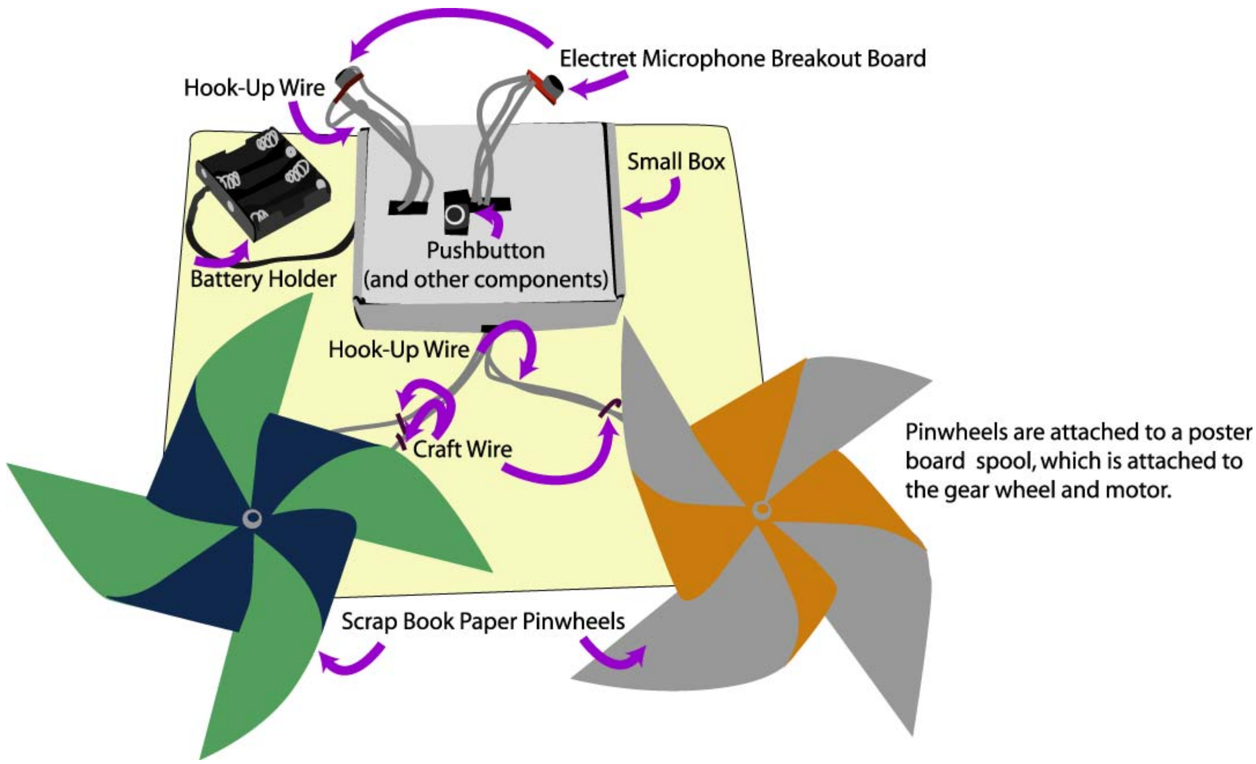




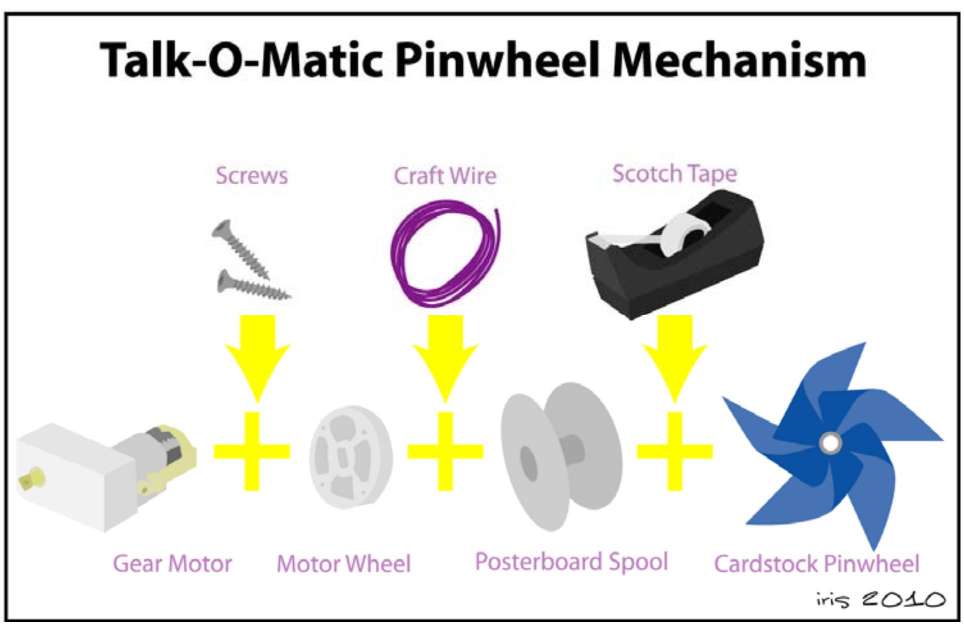




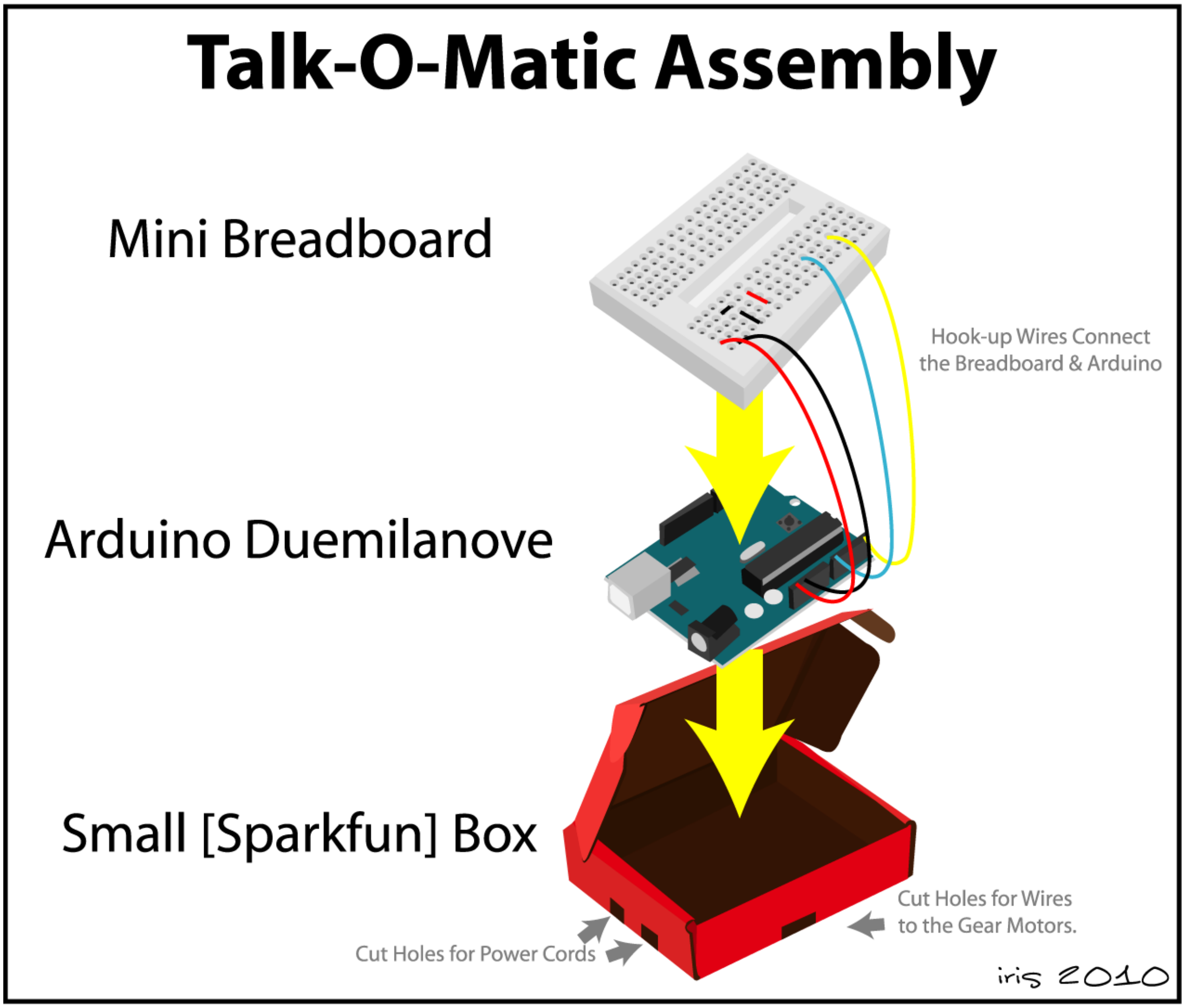
Physical Construction



The above illustration shows the exterior of the Talk-O-Matic, and how the components fit together. The pinwheel mechanisms are made of several parts as well, as shown below.

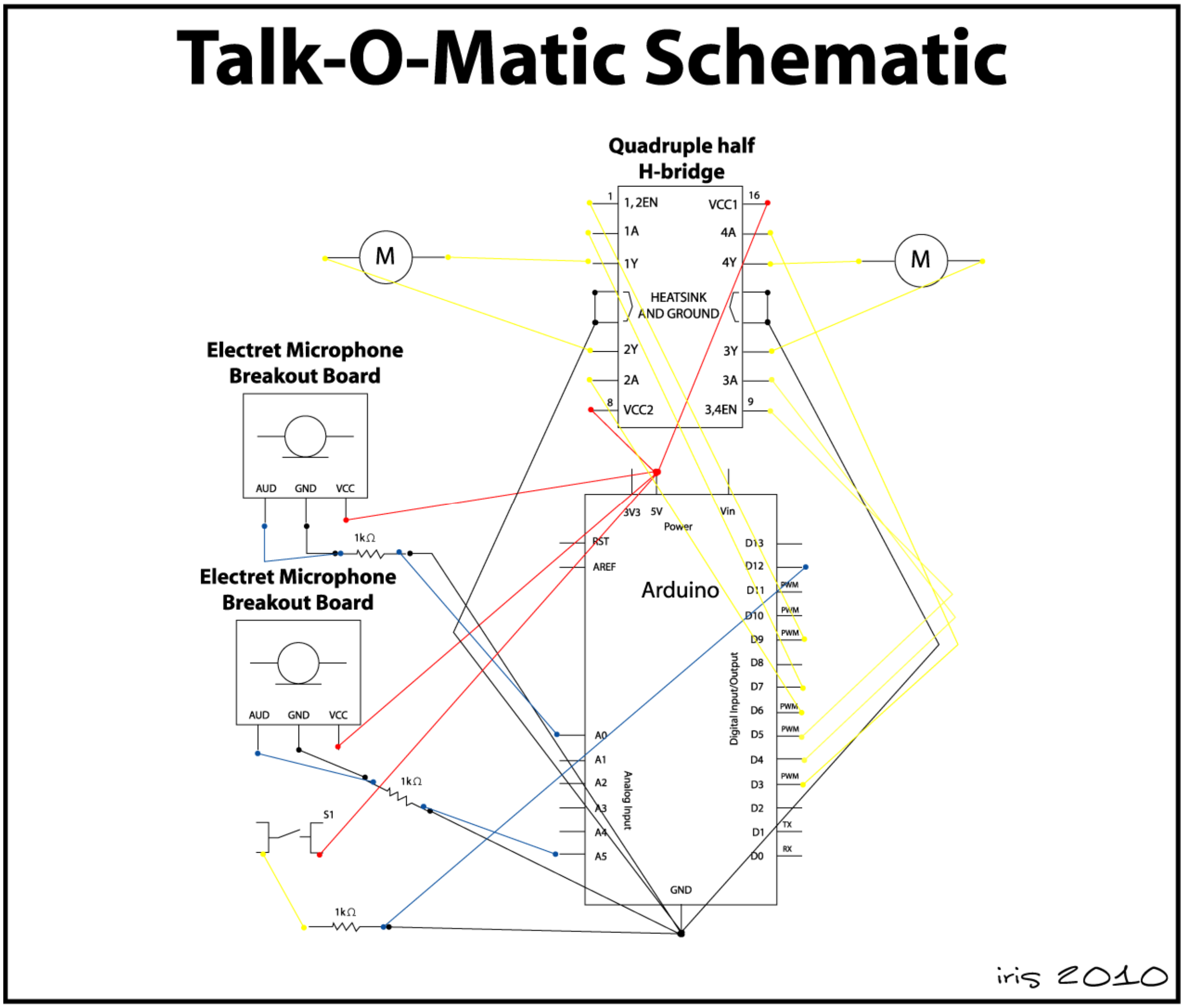


The interior of the "small box" that contains the bulk of the Talk-O-Matics electronics includes a mini breadboard, Arduino Duemilanove, and all the smaller electronics components. The layout of the interior of the box is displayed below.

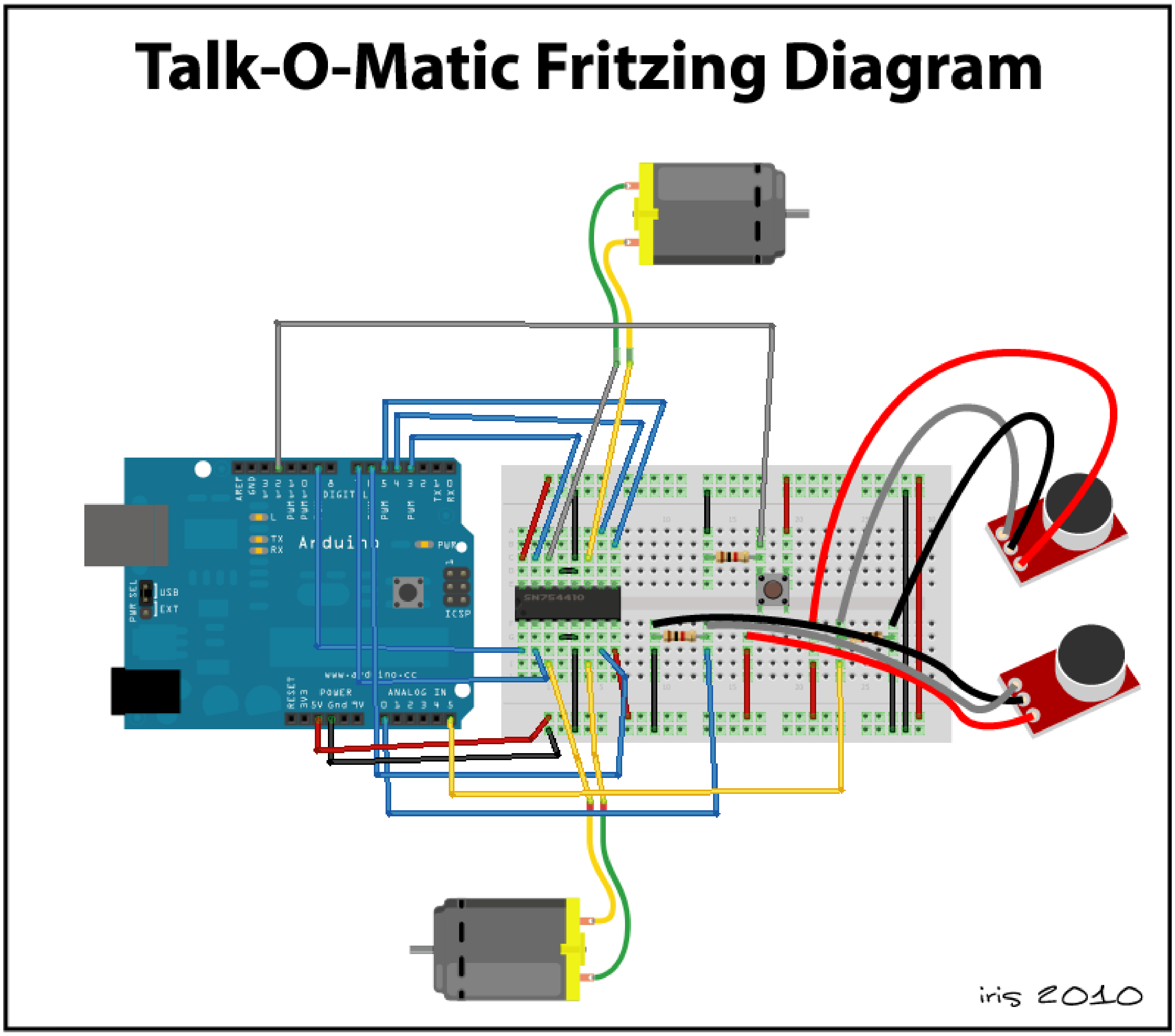


Electronic Details

When it comes to electronic components, the Talk-O-Matic system is basically two motors connected to an H-bridge. The H-bridge is connected to the Arduino, as well as the two microphone breakout boards and the pushbutton switch. The schematic below shows the particular details of the electronics involved.



Below is a Fritzing Diagram of the Talk-O-Matic electronics components. The microphone breakout boards and the gear motors are attached to the breadboard, but are on the exterior of the Talk-O-Matic's small cardboard box. The rest of the electronics components attached to the breadboard and Arduino are stored inside the small box.



Programming Code

There is a considerable amount of Processing code involved to make the Talk-O-Matic function. The figure below describes the basic logic of the code (the bold line, determining if the values are peaks is where there is additional complexity, as described in the “Challenges” section).

Talk-O-Matic Programming Logic

(1) Check if On/Off Button is PRESSED.
(2) If PRESSED, then:
  (a) set system status OFF if on.
  (b) set system status ON & reset() if off.
(3) If ON, then:
  (a) Read microphone 1 and 2.
  (b) **If either value is a peak**, then:
    (i) Add '1' to that microphone's COUNT.
  (c) If mic 1 COUNT is greater than mic 2 COUNT, then:
    (i) Turn on mic 1's motor.
    (ii) Turn off mic 2's motor.
  (d) If mic 2 COUNT is greater than mic 1 COUNT, then:
    (i) Turn on mic 2's motor.
    (ii) Turn off mic 1's motor.
  (e) If mic 1 COUNT equals mic 2 COUNT, then:
    (i) Turn on both mics' motors.
(4) If OFF, then:
  (a) Turn off both mics' motors.
(5) Return to step (1).

iris 2010

The remainder of this section is dedicated to the code used by the Talk-O-Matic.

```
/**
 * PROJECT PART 2
 * - by Iris Howley -
 *
 * This program keeps track of sound coming in through two
 * microphones and displays the input as a function of two
 * gear motors (and an h-bridge). There is also a pushbutton/switch
 * for turning the system on/off and resetting the global counts.
 **/

    //**********************
    //        MOTORS
    //**********************
    int hbridgeR1 = 6;    //declares the first pin on the right side of the hbridge for the motor
    int hbridgeR2 = 7;    //declares the second pin on the right side of the hbridge for the motor
    int motorRpmw = 9;     // this is the pmw that will set how much power the motor is getting (speed)

    int hbridgeL1 = 4;
    int hbridgeL2 = 3;
    int motorLpmw = 5;

    int DEFAULT_MOTOR_POWER = 255;

    //**********************
    //      MICROPHONES
    //**********************
    int mic1SensorPin = A0; // the first microphone, analog in
    int mic2SensorPin = A5; // the second microphone, analog in
```

```
    int mic1Count = 0; // the first mic's activity count
    int mic2Count = 0; // the second mic's activivity count

    double PEAK = 1.3; // what multiple of the  average will be considered a peak.
    int PEAK_THRESHOLD = 5; // how many more peaks one sensor must have than the other to be considered "more" talkative

    // Calibration
    boolean calibrateMic = false; // if we are running this to calibrate the microphones or not
    int average1 = 503; // the average of the smoothed mic1 values at "silence"
    int average2 = 646; // the average of the smoothed mic2 values at "silence"
    int count = 0; // how many sensor readings we've done so far

    //**********************
    //   LOW PASS FILTER
    //**********************
    float filterVal = 0.5;       // this determines smoothness: 0 is off (no smoothing) and .999 is max
    float smoothedVal1 = 0;     // this holds the last loop value for mic1
    float smoothedVal2 = 0;   // this holds the last loop value for mic2

    //**********************
    //    ON/OFF BUTTON
    //**********************
    int onOffPin = 12; // gray wire --> 12
```

```
    boolean onOffStatus = 0; //default status, system is off

    //**********************
    //    MAIN FUNCTIONS
    //**********************

    /**
     * setup() establishes defaults, declares pin modes, and other
     * start-up activities necessary before running the program.
     **/
    void setup() {
      Serial.begin(9600);
      pinMode(onOffPin, INPUT);
      //pinMode(mic1SensorPin, INPUT); // analog in
      //pinMode(mic2SensorPin, INPUT); // analog in

      // Declaring as outputs
      pinMode(hbridgeL1, OUTPUT);
      pinMode(hbridgeL2, OUTPUT);
      //pinMode(motorLpmw, OUTPUT); // analog out

      pinMode(hbridgeR1, OUTPUT);
      pinMode(hbridgeR2, OUTPUT);
      //pinMode(motorRpmw, OUTPUT); // analog out

      reset(); // establish original values of variables
      delay(1000); // wait a second so we ignore the first couple audio readings
    }

    /**
     * loop() runs infinitely on the Arduino.
     **/
```

```
void loop() {
  calibrateMic = false; // we don't want to calibrate
  run(); // the main function of this program
}

/**
 * run() runs the left and right motors in proportion to
 * how much interaction the microphones are experiencing.
 * Motors start 'off', but when the pushbutton turns on
 * the software runs and the motors turn on.
 **/
void run() {
  boolean switchState = digitalRead(onOffPin);

  if (switchState && !onOffStatus) { // the button is pressed, the status is off, TURN ON
    onOffStatus = true;
    Serial.println("Switch state to 'on'");
    delay(1000); // wait a second
  }
  else if (switchState && onOffStatus) { // the button is pressed, the status is on, TURN OFF
    onOffStatus = false;
    Serial.println("Switch state to 'off'");
    reset(); // reset original values
    delay(1000); // wait a second
  }

  if (!onOffStatus) {  // if we're off, turn motors off
    analogWrite(motorLpmw, 0); // turn off left motor
    analogWrite(motorRpmw, 0); // turn off right motor
```

```

    } else if (onOffStatus) { // If we're supposed to be on...then be on
      int mic1SensorValue = analogRead(mic1SensorPin);
      int mic2SensorValue = analogRead(mic2SensorPin);

      // Ignore if '0' is read (since that's due to loose connections)
      if (mic1SensorValue > 50 && mic2SensorValue > 50) {

        // Pass through smoothing (low pass) filter
        smoothedVal1 = smooth(mic1SensorValue, filterVal, smoothedVal1);
        smoothedVal2 = smooth(mic2SensorValue, filterVal, smoothedVal1);

        // Print smoothed and original audio values
        Serial.print(smoothedVal1); Serial.print(" ("); Serial.print(mic1SensorValue); Serial.print(")"); Serial.print("\t");
        Serial.print(smoothedVal2); Serial.print(" ("); Serial.print(mic2SensorValue); Serial.println(")");

        if (calibrateMic) { // if we're calibrating
          average1 += smoothedVal1;
          average2 += smoothedVal2;
          count++;
        }

        // Determine if these values are peaks
        if (smoothedVal1 > average1*PEAK) { // it is a peak for mic1
          mic1Count++;
        }
```

```
      if (smoothedVal2 > average2*PEAK) { // it is a peak for mic2
        mic2Count++;
      }

      Serial.print(mic1Count); Serial.print(" vs "); Serial.println(mic2Count);
      if ((mic1Count - mic2Count) > PEAK_THRESHOLD) { // if mic1 has more than threshold peaks than mic2
        leftMotorForward(DEFAULT_MOTOR_POWER);
        analogWrite(motorRpmw, 0); // turn off right motor
      } else if ((mic2Count - mic1Count) > PEAK_THRESHOLD) { // if mic2 has more than threshold peaks than mic1
        rightMotorForward(DEFAULT_MOTOR_POWER);
        analogWrite(motorLpmw, 0); // turn off left motor
      } else { // equal (within 10 counts)! turn both motors on
        rightMotorForward(DEFAULT_MOTOR_POWER);
        leftMotorForward(DEFAULT_MOTOR_POWER);
      }

    } // end if mic1 > 50 && mic2 > 50

  } // end if onOffStatus
}

/**
 * reset() reestablishes the default values of variables
 * and resets everything to its original state.
 **/
void reset() {
```

```
        mic1Count = 0; // the first mic's activity count
        mic2Count = 0; // the second mic's activivity count

        smoothedVal1 = 0; // first mic's low pass filter value
        smoothedVal2 = 0; // second mic's low pass filter value

        if (calibrateMic) { // if we're calibrating
          average1 = average1/count;
          average2 = average2/count;
          Serial.print("Average1: "); Serial.print(average1); Serial.print("\tAverage 2: "); Serial.println(average2);
        }
    }

    //**********************
    //   LOW PASS FILTER
    //**********************
    /**
      * Smooths a given sensor reading using past smoothed values
      * and a given filtering/smoothing level.
      * This function was adapted from the Arduino Playground:
      *     http://www.arduino.cc/playground/Main/Smooth
      * @param data the sensor reading we're smoothing
      * @param filterVal the level of filtering/smoothing to apply
      * @param smoothedVal the running 'smoothed value' associated with the sensor
      * @return the smoothed sensor value
      **/
```

```
  int smooth(int data, float filterVal, float smoothedVal) {
    // Check to make sure param's are within range
    if (filterVal > 1) {
      filterVal = .99;
    } else if (filterVal <= 0) {
      filterVal = 0;
    }

    smoothedVal = (data * (1 - filterVal)) + (smoothedVal  *  filterVal);
    return (int)smoothedVal;
  }

  /**
   * Outputs the read-in sensor value, the value after smoothing,
   * and the value at which we're smoothing/filtering.
   * @param sensPin the analog pin we're smoothing.
   **/
  void testFilter(int sensPin) {
    int sensVal = analogRead(sensPin);
    smoothedVal1 =  smooth(sensVal, filterVal, smoothedVal1);

    Serial.print(sensVal);
    Serial.print("   ");
    Serial.print(smoothedVal1);
    Serial.print("      ");
    Serial.print("filterValue * 100 =  ");   // print doesn't work with floats
    Serial.println(filterVal * 100);
    delay(1000);
  }

  //**********************
```

```
//    MOTOR FUNCTIONS
//**********************
/**
 * Move the left motor forward.
 * @param power the power at which to move the motors
 **/
void leftMotorForward(int power) {
  // Keeps the left motor on
  analogWrite(motorLpmw, power);
  digitalWrite(hbridgeL1, LOW);    //turns the motors on
  digitalWrite(hbridgeL2, HIGH);
}

/**
 * Move the rightft motor forward.
 * @param power the power at which to move the motors
 **/
void rightMotorForward(int power) {
  // Keeps the right motor on
  analogWrite(motorRpmw, power);
  digitalWrite(hbridgeR1, LOW);    //turns the motors on
  digitalWrite(hbridgeR2, HIGH);
}
```

Future Work

The Talk-O-Matic performs the desired functions, as detailed in the beginning of this document, however, there is still more work that can be performed to improve its effectiveness:

- Improve the form-factor of Talk-O-Matic, perhaps by making the pinwheels wireless. Portable pinwheels could be attached to nametags, or moved around on a conference table to make the display more integrated. Making the microphones wireless as well would improve the flexibility of the Talk-O-Matic.
- There are also other displays, besides pinwheels, that could be used to display discussion. Pinwheels are a "comparative" display, but a tug-of-war could be considered a "competitive" display, and a "cooperative" display could be two avatars raising a tent together.
- The Talk-O-Matic should be studied more in depth, to see if it brings out the desired behaviors described earlier. User studies could tell us if the Talk-O-Matic actually increases or decreases participation in a discussion.
- Instead of showing only face-to-face conversation, the Talk-O-Matic could display online chat conversation. This is particularly relevant to the Computer-Supported Collaborative Learning research community.
- With a finer grained understanding of *what* is actually being said (rather than simply *if* something is being said or not), we could use the display to encourage or discourage specific chat behaviors (ones that lead to learning, disruptive contributions, bullying, etc).