Let's Code Blacksburg! “Talking Skull” Workshop, Part-1 – Build-n-Code Doc

All - https://github.com/LetsCodeBlacksburg/arduino-talking-skull v2017-10-26_tweeks (CC)(BY)(SA)
Part-2 Hware-Hacking-Guide (the skull)

This 4-6hr hands on workshop steps you through building and programming a proximity activated “Talking Skull” that requires:

1 - Mini USB cable (blue)
1 - Funduino Arduino board (red, with extra pin headers)
1 - 9v Alkaline Battery w/attached clip
1 - DFPlayer mini, serial controllable MP3 player
1 - 4GiB µSD card (w/ten scary skull sounds preloaded)
1 - Speaker (8ohm, 3W)
1 - Ultrasonic PING distance sensor (HC-SR04, 4pins)
2 - Red LEDs
2 - 100Ω resistors (for LEDs)
2 - 1kΩ resistors (for MP3 serial lines)
4 - Female / Female connecting wires
12 - Male / Female connector wires
1 - SG90 micro 9g servo (for moving the mouth)
1 - popsicle stick (for reinforcing the servo mount)
(opt) - breadboard (if building other things vs making skull electronics permanent)
(opt) - foam board and velcro (unmounting & use arduino/mp3 in other projects)
-use of a dremel cutter and hot glue

1) Use First Cookbook Recipe - Blink

Before you build anything or do any real programming, you need to first open your the Files / Examples / 01. Basics / Blink, click the compile/upload icon _➔_, and verify that blink is actually working (blinking your pin 13 LED indicator). If that works, then try changing the blink speed of the LED. This will verify that your computer and the installed arduino IDE software (from www.arduino.cc) is properly configured to talk to your arduino. Get TA sign-off before proceeding:

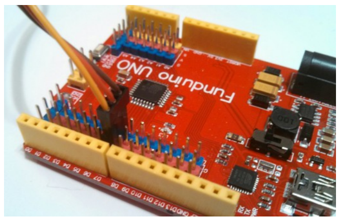
TA SIGN-OFF: __________

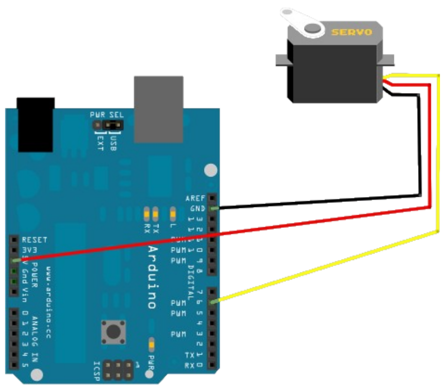
2) Rough Calibration of Your Mouth Servo (get approx closed vs open servo values)

This next step is purely to check to make sure your servo is working, and that you get the approximate values you think you will use to open and close your skull's mouth.

Before we actually attach the servo to the skull, we need to just hook the servo up to pin 6 (see right/below), load the sample **`LCBB_Talking-Skull_2_servo-test.ino`** code from the github repository, and test the servo movement starting with open/closed values of ~90 and ~110.

NOTE: If connecting to a "Funduino" or a "sensor shield", the servo connector (brown, red, orange) can be connected directly to 3 pin sensor connector for pin6 which includes 5v and ground.





Servo shown here connected to standard Arduino.

NOTE: We are not hooking the servo to the skull at this point.
This step is merely for testing your servo and verifying it's working well.

Example code snippet from full LCBB_servo-test.ino on guthub:

```
...
// adjust the values as needed once mounted in the skull.
int mouthClosed=125 ;    // These values needto be discovered for your config.
int mouthOpened=80 ;     // These values needto be discovered for your config.
int mouthDelay=300;      // Time to allow one open or closed movement (1/2 cycle)
...

// *************** MAIN LOOP *******************
void loop() {
  myServo.write(mouthOpened);
  delay(mouthDelay);

  myServo.write(mouthClosed);
  delay(mouthDelay);

  delay(5000);     // pause so you know which value is open vs closed
}
```

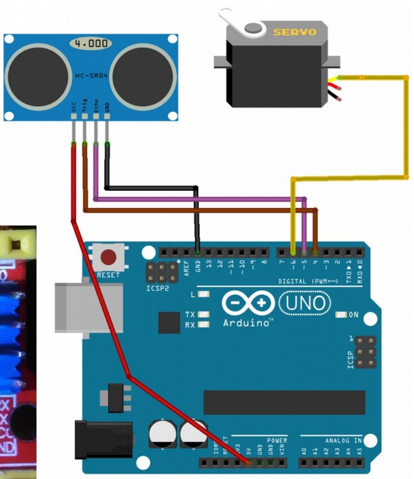
On you get your servo working, write down the experimental values you think you want to try in your skull's jaw (if servo is mouted in the skull's right jaw joint (left looking at the skull).

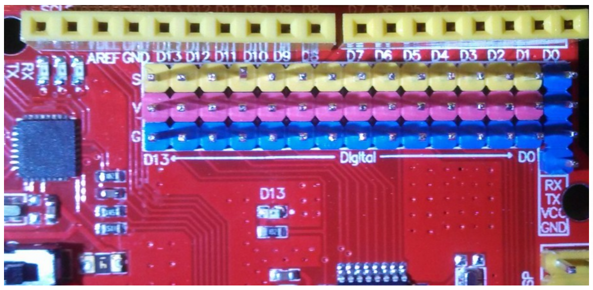
Once working, save your code and call it `Talking-Skull_servo-test`

mouthOpened =   mouthClosed =   mouthDelay =   TA SIGN-OFF: ________

Challenge:
Write a function called `laughCount(n)` that will take in integer “n” and laugh that many times!

3) Ultrasonic “ping” Distance Sensor:
To read up on the theory of the Ultrasonic distance sensor, look up the *INPUT: Ultrasonic “Eyes” Range Sensor* section of the “LCBB Arduino Cookbook” here. We're going to hook up our US-100 or HC-SR04 distance sensor to Ground and VCC (outer two pins) and pins 4 & 5 for Trigger and Echo on the sensor respectively.





NOTE: If using the red Funduino board, to keep from having to connect to limited GND and 5v connections as shown in the right schematic, try making use of the ample G(round) and V (5v) connector headers by using a male/female connector wire.

For this workshop, instead of using the cookbook recipe, we recommend entering the code bnelow (or if you're having problems getting it working, then copy the ping-test code from our github arduino-talking-skull code repository):

```
//Ping sensor
const   int trgPin = 4;     // pin we're sending the trigger/ping signal on
const   int echoPin = 5;    // pin we're reading back the echo on
const   long motionDist=6;  // distance that triggers a action
long    distance=5000;      // distance to be recorded by ping sensor

//********************** SETUP BLOLCK **************************
void setup() {
  Serial.begin(9600); // for sending serial text back to your computer

  // Set Up US-100 or HC-SR04 PING SENSOR I/O PINS
  pinMode(trgPin, OUTPUT);
  digitalWrite(trgPin, LOW);    // this pin sends out the ping signal
  pinMode(echoPin, OUTPUT);    // just to make sure
  digitalWrite(echoPin, LOW);  // we clear any previous settings
  pinMode(echoPin, INPUT);     // and then use it as INPUT
  delay(500);
}

//************************* MAIN LOOP ********************************
void loop() {
  delay(100);
  distance=getDist();       // samples distance from ping sensor
  delay(100);

  ////// Loop here until someone comes closer than motionDist value
  while(distance > motionDist ){
    distance=getDist();                    // sample distance
    delay(100);
    Serial.print("Distance = ");
    Serial.println(distance);
    }

  Serial.println("************** Someone is close! **************");
  Serial.print("Distance = ");
  Serial.println(distance);
  Serial.println("--move mouth, play sound--");
  delay(5000);
}

// Code from Arduino ping
// ***********************************************************
// ***** getDist() *******************************************
// ***********************************************************
long getDist()
{
  long duration, inches, cm;
  digitalWrite(trgPin, HIGH);          // start the outgoing ping
  delayMicroseconds(10);               // do the ping for 10uS
  digitalWrite(trgPin, LOW);           // stop doing the ping
  duration = pulseIn(echoPin, HIGH);  // grab the delay of return echo
  inches = microsecondsToInches(duration);    // convert echo time to inches
  //cm = microsecondsToCentimeters(duration); // use for cm
  //Serial.print(inches);
  //Serial.print("in, ");
  //Serial.print(cm);
  //Serial.print("cm");
  //Serial.println();
  return (inches);
}

// Original code from the ping sensor library
long microsecondsToCentimeters(long microseconds){
  // The speed of sound is 340 m/s or 29 microseconds per centimeter.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}

long microsecondsToInches(long microseconds){
  // According to Parallax's datasheet for the PING)))
  // 73.746 microseconds per inch
  // See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf
  return microseconds / 74 / 2;
}
```

Get this code to compile and upload , and turn click your arduino serial monitor icon_ _ to see if you're recording the approximate distance to a large flat object like a book or jacket. If you get an error when clicking on the serial monitor, check your arduino software's tools / port setting and make sure the correct serial device is selected.

You should be getting back data like this as you move objects away from and closer to your distance sensor:

```
Distance = 17
Distance = 17
Distance = 18
Distance = 19
Distance = 21
Distance = 20
Distance = 16
Distance = 12
Distance = 8
Distance = 5
************** Someone is close! **************
Distance = 5
--move mouth, play sound--
```

If your code is working.. SAVE IT with the meaningful name `Talking-Skull_3_ping-test` .

Q: Look at the variables at the top of your program. What is the distance variable that triggers the animation of the skull's sound and mouth?

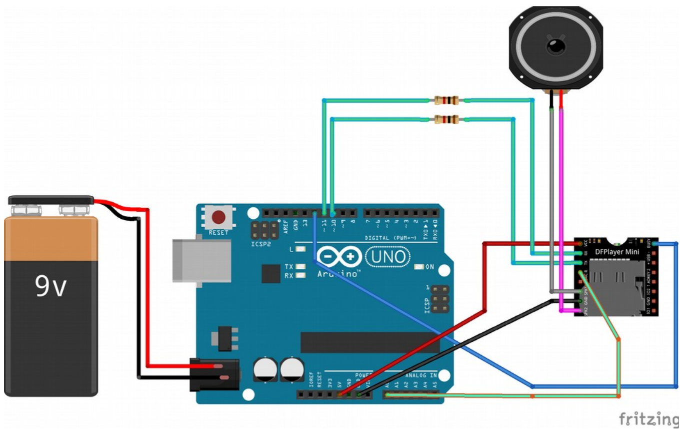
Variable name = ______________ Variable Setting= __________ TA SIGN-OFF: __________

Challenge:
Now when your distance sensor is triggered, instead of just printing "`--move mouth, play sound--`", instead, pull the needed variables, setup and loop code to actually move the skull's mouth servo opened and closed.

Show this to your TA and get this signed off, and save this code as `Talking-Skull_3_distance+servo`
__**Don't disconnect your servo and range finder for the next session.**__ Just ignore them until step 5.

TA SIGN-OFF: ____________

__**4) Implement the DFPlayer Mini, serial controlled MP3 player:**__
The DFPlayer is a 3.3v input/output device and __it is very sensitive and will die if it is not hooked up exactly right__. When hooking up this circuit, leave it unplugged from USB and the battery and get TA-helper sign-off before you plug in USB or power it up. Hooking this MP3 player wrong can destroy the serial control lines, making it useless for this project.

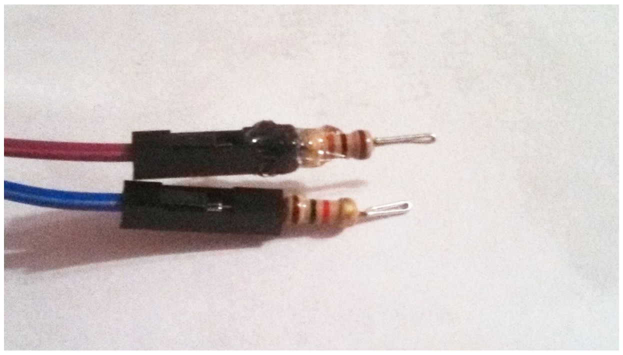


WARNING: Do not hook up USB cable or 9V battery before getting TA signoff on this section. Not being extra careful at this stage can easily blow the DFPlayer's serial I/O control lines. Ask me how I know… I inadvertently blew two DFPlayers while writing this workshop. Learn from my mistakes.

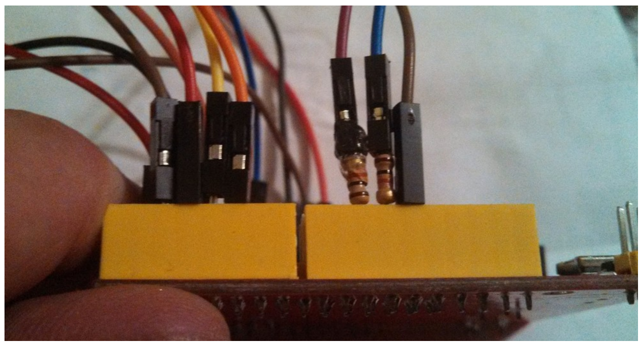
__Notice that the DFPlayer is connected to the 5volts pin for power, but that its serial TX (transmit) and RXC (receive) lines are buffered (made safe) by going through two 1k ohm resistors__. These resistors just drop a couple of volts off the 5v on the arduino I/O pins so they're safe to use with the DFPlayer's I/O pins. Not doing this, will blow the serial control lines of the DFPlayer.

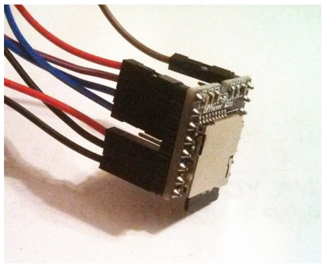
Since we are not using a breadboard for this workshop, we're going to be hooking these resistors directly in to two M/F connection wires, and pushing them directly into pins 10 & 11 of the arduino, something like this:

Two 1k ohm resistors on TX/RX Lines



Arduino TX(11)→MP3's RX(1)
Arduino RX(10)←MP3's TX(2)





Just like we're hooking the serial lines, here's what the whole DFPlayer looks like all using the direct connect method. Pin 1 (5v) is the upper left pin. Upper right is the unit's pin 16 (_BUSY). This is how we are going to tell when the DFPlayer is busy playing a sound (which is when we want to be moving the mouth).

This code requires that you install the <DFRobotDFPlayerMini.h> library. Do this from the arduino menu **Sketch / Include Library / Manage Libraries** and search for the `DFPlayer_Mini_MP3` library. Install it if not already installed.

Here's the code for implementing a basic, sequential sound file MP3 player:

```
// Hardware wiring of DFPlay
//
// DFPlay mini  --Vcc(1)--RX(2)--TX(3)--DACR(4)--GND(7)  +(6) -(8)  ...  _BUSY
//                 |       |      |       |       |       |    |           |
//     Arduino    5v      11*    10*     A0      GND      |    |          12*
//                                                       \Spkr/
//                              * requires 1k resisitor

#include "Arduino.h"
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>

/////////// HARDWARE SETTINGS/HOOKUPS ////////
//DFPlayer mp3 player
const int ardRX=10;       // The arduino software Receive lins (goes to DFPlayer TX)
const int ardTX=11;       // The arduino software Transmit line (goes to DFPlayer RX)
const int dfBusy=12;      // From DFPlayer pin 16, active low (HIGH != playing sound)
int sndFile=1;            // Sound file pointer.
int fileCount=9;          // Either set the max # of files here, or load it from
myDFPlayer.readFileCounts()
  // NOTE: readFileCounts() also sees deleted files.
  // May need to reformat card to load new sounds.
int setVol=25;            //Set volume value (0~30)

// Software Serial Pins To DFPlayer
SoftwareSerial mySoftwareSerial(10,11); // RX, TX

DFRobotDFPlayerMini myDFPlayer;
void printDetail(uint8_t type, int value);

/////////////////////////////////////////////////////////////////////
//////////////////////// LARGE SETUP BLOCK //////////////////////////
/////////////////////////////////////////////////////////////////////
void setup(){

//  Set up DFPlay mini
  delay(250);
  pinMode(ardRX, INPUT);
  pinMode(ardTX, OUTPUT);
  pinMode(dfBusy, INPUT);
  delay(100);
  mySoftwareSerial.begin(9600);
  delay(100);
  Serial.begin(9600);
  Serial.println(F("DFRobot DFPlayer Setup"));
  Serial.println(F("Initializing DFPlayer ... (May take 3~5 seconds)"));
  // Check for DFPlay initialization via softserial
  if (!myDFPlayer.begin(mySoftwareSerial)) {
    // if it did not work
    delay(20);
    Serial.println(F("Unable to begin:"));
    Serial.println(F("1.Please recheck the connection!"));
    Serial.println(F("2.Please insert the SD card!"));
    // T-SHOOTING
    Serial.print("INFO: player state / file counts: ");
    Serial.print(myDFPlayer.readState()); //read mp3 state
    Serial.print(" / ");
    Serial.println(myDFPlayer.readFileCounts()); //read all file counts in SD card
    while(true);      // Hang forever if error
  }

  // if serial setup worked
  Serial.println(F("DFPlayer Mini online."));
  // configure settings
  myDFPlayer.setTimeOut(500); //Set serial communictaion time out 500ms
  myDFPlayer.volume(setVol);  //Set volume value (0~30).
  myDFPlayer.EQ(DFPLAYER_EQ_NORMAL);
  myDFPlayer.outputDevice(DFPLAYER_DEVICE_SD);

///// Print Status:
  Serial.print("SETUP-INFO: player state= ");
  Serial.print(myDFPlayer.readState());         //read mp3 state
  Serial.print(" / volume setting(0-30)= ");
  Serial.print(myDFPlayer.readVolume());        //read current volume
  Serial.print(" / EQ setting= ");
  Serial.print(myDFPlayer.readEQ());          //read EQ setting
  Serial.print(" / fileCount= ");
  Serial.println(myDFPlayer.readFileCounts());  //read all file counts in SD card
}


// ***********************************************************
// ********************** MAIN LOOP ************************
// ***********************************************************
void loop()
{

  ////// If at the last sound file, then loop back to the top (1)
  if (sndFile == (fileCount+1) ) {
    sndFile = 1;
  }

  ////// Begin playing the next sound file...
  Serial.println("************** PLAYING ****************");
  myDFPlayer.play(sndFile);   //Play the next mp3
  delay(100);                 // wait to start


  ////// While DFPlayer _BUSY line is active(low), use preferred laugh function.
  while(!digitalRead(dfBusy) == true ){
    Serial.println("still playing...");
    Serial.println("Good place to flash eyes & move mouth... ;)");
    delay(300);
  }

  delay(200);
  sndFile++;     // Incriment to the next sound file
```

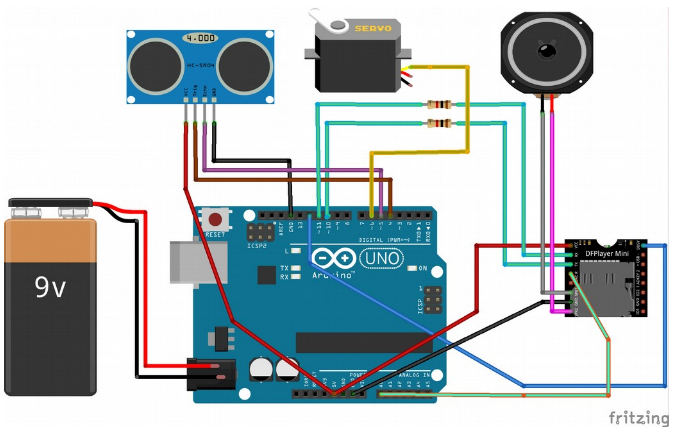
```
}

//////////////////// printDetail() //////////////////////
void printDetail(uint8_t type, int value){
  switch (type) {
    case TimeOut:
      Serial.println(F("Time Out!"));
      break;
    case WrongStack:
      Serial.println(F("Stack Wrong!"));
      break;
    case DFPlayerCardInserted:
      Serial.println(F("Card Inserted!"));
      break;
    case DFPlayerCardRemoved:
      Serial.println(F("Card Removed!"));
      break;
    case DFPlayerCardOnline:
      Serial.println(F("Card Online!"));
      break;
    case DFPlayerPlayFinished:
      Serial.print(F("Number:"));
      Serial.print(value);
      Serial.println(F(" Play Finished!"));
      break;
    case DFPlayerError:
      Serial.print(F("DFPlayerError:"));
      switch (value) {
        case Busy:
          Serial.println(F("Card not found"));
          break;
        case Sleeping:
          Serial.println(F("Sleeping"));
          break;
        case SerialWrongStack:
          Serial.println(F("Get Wrong Stack"));
          break;
        case CheckSumNotMatch:
          Serial.println(F("Check Sum Not Match"));
          break;
        case FileIndexOut:
          Serial.println(F("File Index Out of Bound"));
          break;
        case FileMismatch:
          Serial.println(F("Cannot Find File"));
          break;
        case Advertise:
          Serial.println(F("In Advertise"));
          break;
        default:
          break;
      }
      break;
    default:
      break;
  }
}
```

Save your working code as **`Talking-Skull_4_dfplayer-test`** . There is much more info, troubleshooting and DFPlayer functionality included in the **`LCBB_Talking-Skull_4_dfplayer-test`** code on our github repo here.

TA SIGN-OFF: ____________

5) Bringing It All Together!

Now, if you previously disconnected the servo or ultrasonic range finder, now go ahead and connect in each of the components as shown (making sure the servo has all three pins correctly connected).



If you got your DFPlayer working in step 4, then you should not need any "new code" except the code from steps 2, 3 and 4, combined with the laughCount(n) challenge in step 3, and carefully bring it all together.

If you need to see the form of what it all looks like together, take a peek out at LCBB_Talking-Skull_5_ping-servo-dfplayer_complete
and bring in whatever code you need.

After you have your code working, BE SURE TO SAVE IT with a meaningful name such as **`Talking-Skull_ping-servo-dfplayer`**. Once you have this all working, you're ready to load it all into the skull, hot glue your speaker into place and do your final testing and get TA sign-off!

TA SIGN-OFF: ____________

Follow the "Part-2" LCBB_Talking-Skull_Pt-2_Hware-Hacking-Guide.pdf out on our github site!