Diese Seite dokumentiert das Projekt Audio-Guide, welches von Annika Just im Sommersemester 2020 durchgeführt wird.

Projektidee

Ich möchte im Rahmen dieses Projektes einen Audio-Guide entwerfen und bauen. Auf Knopfdruck werden Sprachaufnahmen abgespielt, die beispielsweise zusätzliche Informationen zu Inhalten einer Ausstellung geben. Das heißt, ein spezifischer Knopfdruck spielt einen spezifischen Sound. Eine Idee wäre, dass der Audio-Guide beispielsweise sich selbst mit Hilfe eines Plakates erklärt. Damit entwickele ich ein Tool, welches zur Wissenschaftskommunikation eingesetzt werden kann und erlerne währenddessen neue wissenschaftliche Skills anhand eines konkreten Projekts, beispielsweise Mikro-Controller Programmierung und Löten.

Projektplan

Hier kannst du dir den Projektplan herunterladen, welchen ich zu Semesterbeginn erstellt habe.

audio_guide_projektplan.pdf

Dokumentation

Projektziele

Anhand der Frage: “Was muss ein Audio-Guide können?” lassen sich die folgenden Projektziele ableiten:

- Mobilität des Geräts, zB über Powerbank, Akkus
- Einfache Bedienung: jeder Knopf spielt ein spezifisches Soundfile ab
- Anschluss an Kopfhörer oder Lautsprecher möglich, zB über Audio-Klinke

Umsetzung

Erster Versuch mit dem dfPlayer + Steckbrett: Gibt es Sound?

Um die Hardware + Ansteuerung zu überprüfen wurde der dfPlayer an eine gefundene Audio-Buchse angelötet und ein simples Set-Up direkt aus der DFMiniMp3.h library entnommen. Schaltplan siehe hier:

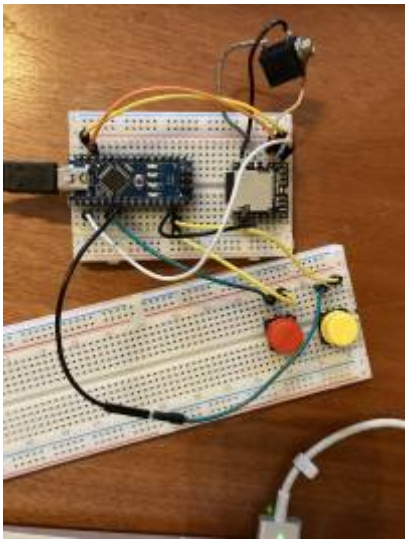
lab_prepare.pdf

Die Ansteuerung des dfPayers klappt, der Code auch, es werden die Sounds in Dauerschleife hintereinander abgespielt.

ABER: die Audio-Buchse spielt nur Mono obwohl das Stereo-Signal verfügbar ist (das weiß ich durch Nachmessen) - es wird eine neue bestellt.

Zweiter Versuch mit dem dfPlayer + Steckbrett: Sound auf Knopfdruck?

Im nächsten Schritt wurde das Set-Up sowie der Code erweitert, um Sound auf Knopfdruck zu erhalten. Dazu wurden auf dem Steckbrett zwei Pushbuttons angeschlossen. Desweiteren wurde ein Steck-Potentiometer zur Volumenkontrolle eingesetzt.

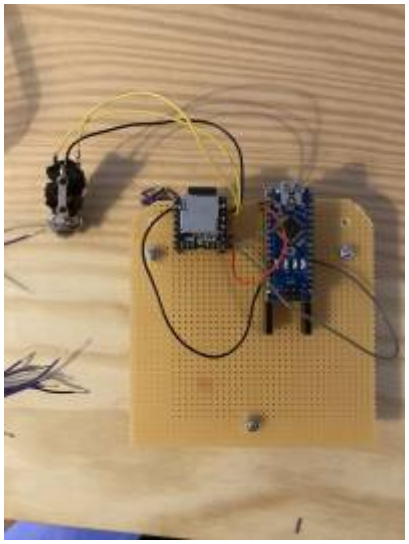
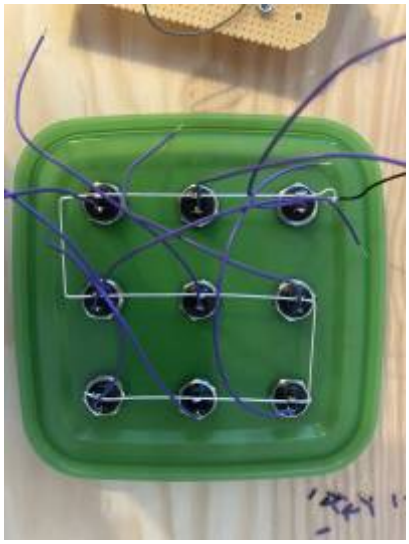
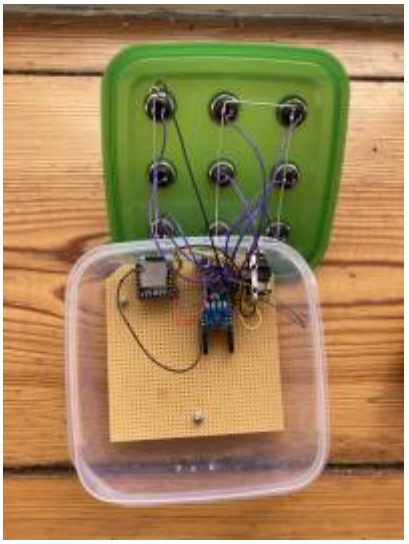


Code-Erweiterung und finaler Zusammenbau Prototyp

Um den Prototypen fertig zu stellen, wurden die folgenden Schritte getan:

- Anlöten neuer Buchse
- Aufbringen von Arduino / dfPlayer auf Platine
- Verlöten der Knöpfe (Entscheidung für 9 Knöpfe, da schöne Matrix möglich 3×3)
- Entscheidung für lustiges Gehäuse (Brotbüchse), Bohren von Löchern

- Anpassung Code (für den finalen Code bitte hier klicken: Code Sound-Box)

Ergebnis

Der Prototyp der Soundbox funktioniert soweit wie geplant. Allerdings gibt es noch einige Verbesserungsmöglichkeiten, welche in Zukunft noch umgesetzt werden:

- Hinzufügen weiterer Sounddecks - diese werden per Knopfdruckkombination gewechselt, LED zeigt an in welchem Deck man sich befindet
- Einstellung der Lautstärke möglich machen (Code ist schon da, wurde auch schon ausprobiert, nur Einschraub-Poti ist nicht vorhanden)
- Verbesserung des Gehäuses (ggf. Kiste aus Holz, 3D-Druck?)



Reflektion

Die Umsetzung des Projektes war für mich sehr zeitintensiv, da ich mich in viele Themen erst einarbeiten musste. Allerdings habe ich dabei auch sehr viel gelernt und das Problemelösen hat mir viel Spaß gemacht. Vor allem die Kombination von Hardware und Software war für mich neu - also

dass ich mit meinem Code tatsächlich Bauteile ansteuere, und diese dementsprechend auch verkabeln und anschließen muss. Mit Elektronik kannte ich mich vorher gar nicht aus - in der Programmierung habe ich schon deutlich mehr Erfahrung. So musste ich immer wieder doch noch Kleinigkeiten besorgen.

Um die Soundbox als Audio-Guide nutzen zu können, fehlen noch die Sprachaufnahmen. Diese waren ursprünglich im Projektplan vorgesehen, werden aber aus zeitlichen Gründen nicht mehr in diesem Semester umgesetzt - insofern fehlt dem Projekt etwas der Aspekt der praktischen Wissenschaftskommunikation. Eventuell werde ich diese im Rahmen von lab:present umsetzen. Solange bleibt die Soundbox einfach ein Soundboard mit dem man lustige Geräusche abspielen kann.

Literatur

Hier liste ich die Ressourcen auf, die mir bei der Durchführung dieses Projekt geholfen haben bzw. noch helfen werden.

Sound-Box mit Arduino

Dokumentation zum MP3 Player

https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299

Ansteuerung Player Beispiele

https://wolles-elektronikkiste.de/dfplayer-mini-ansteuerung-mit-dem-arduino

https://www.instructables.com/id/MP3-Player-With-Arduino-Using-DF-Player-Mini/

https://gist.github.com/hdo/c97a4d7c866c61db902fefb7ab28a57f

https://gist.github.com/hdo/974da4330ac8bf7456ae2c7fa4cd315f

https://oneguyoneblog.com/2019/10/29/arduino-halloween-mp3-soundboard/

Videos

https://www.youtube.com/watch?v=UodfePdNfg8

https://www.youtube.com/watch?v=t6pdVZBE2Wg

Audio-Guides im Museum

Hier fehlt eine gründliche Literatursuche zum Nutzen von Audioguides in der Wissensvermittlung, und wie man diese ideal einsetzt. Wie sollte ein Audio-Guide gestaltet sein? Kann man sich durch Audio-Guides Inhalte tatsächlich besser merken? Ggf. können diese Fragen im Rahmen von Lab:present geklärt und umgesetzt werden.

de Teffé, C., & Müller-Hagedorn, L. (2008). Zur Wirkung von emotional und sachlich gestalteten Audio-

Guides in Museen. In Kulturmanagement der Zukunft (pp. 219-247). VS Verlag für Sozialwissenschaften.

Pitterle, M. (2014). Gedächtnisunterstützung mittels Audio im Museumskontext (Master's thesis).

Code

```
// Soundbox
// V1.0

// Dies ist der Code zur Sound-Box für lab:prepare im Sommersemester 2020
// Die Box nutzt Arduino und den dfPlayer
// last edit 20.09.2020
// Annika Just

// Soundfiles müssen auf der SD Karte im Ordner mp3 liegen
// und müssen so nummeriert sein: 0001.mp3

// Libraries einbinden

#include <SoftwareSerial.h>
#include <DFMiniMp3.h>

// DEFINITIONS

// Für Debounce
const int debounceDelay = 15; //Debounce Time
unsigned long lastDebounceTime = 0; //Zeitstempel

//Buttons für die Songs
const uint8_t NUM_BUTTONS   = 9; //Anzahl der Buttons
const byte buttons[NUM_BUTTONS] = {4, 5, 6, 7, 8, 9, 10, 11, 12}; //Pins auf
denen die Button-Inputs jeweils liegen, anordnung nach Tonleiter
byte prevButtonStates[NUM_BUTTONS]; //bool?
byte currentButtonStates[NUM_BUTTONS]; //bool?

//Poti für Lautstärke
// edit: noch nicht Hardwaremässig implementiert (habe kein Einschraubpoti
da)
const int NUM_POTIS = 1; // Anzahl der Potis
const byte potis[NUM_POTIS] = {A3}; //Analog-Pins auf denen die Potis liegen
int prevPotiWert[NUM_POTIS];
int currentPotiWert[NUM_POTIS];
int volume;


// das hier wurde 1:1 so übernommen aus Bsp. Code der library:
// implement a notification class,
// its member methods will get called
```

```
//
class Mp3Notify
{
  public:
    static void PrintlnSourceAction(DfMp3_PlaySources source, const char*
action)
    {
      if (source & DfMp3_PlaySources_Sd)
      {
        Serial.print("SD Card, ");
      }
      if (source & DfMp3_PlaySources_Usb)
      {
        Serial.print("USB Disk, ");
      }
      if (source & DfMp3_PlaySources_Flash)
      {
        Serial.print("Flash, ");
      }
      Serial.println(action);
    }
    static void OnError(uint16_t errorCode)
    {
      // see DfMp3_Error for code meaning
      Serial.println();
      Serial.print("Com Error ");
      Serial.println(errorCode);
    }
    static void OnPlayFinished(DfMp3_PlaySources source, uint16_t track)
    {
      Serial.print("Play finished for #");
      Serial.println(track);
    }
    static void OnPlaySourceOnline(DfMp3_PlaySources source)
    {
      PrintlnSourceAction(source, "online");
    }
    static void OnPlaySourceInserted(DfMp3_PlaySources source)
    {
      PrintlnSourceAction(source, "inserted");
    }
    static void OnPlaySourceRemoved(DfMp3_PlaySources source)
    {
      PrintlnSourceAction(source, "removed");
    }
};

// instance a DFMiniMp3 object,
// defined with the above notification class and the hardware serial class
SoftwareSerial secondarySerial(3, 13); // RX, TX
DFMiniMp3<SoftwareSerial, Mp3Notify> mp3(secondarySerial);
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  Serial.println("initializing...");

  mp3.begin();

  uint16_t volume = mp3.getVolume();
  Serial.print("volume ");
  Serial.println(volume);
  mp3.setVolume(25);

  uint16_t count = mp3.getTotalTrackCount(DfMp3_PlaySource_Sd);
  Serial.print("files ");
  Serial.println(count);

  Serial.println("starting...");

  // initialize the pushbutton pin as an input:
  for (int i = 0; i < NUM_BUTTONS; ++i) {
    pinMode(buttons[i], INPUT_PULLUP);
  }

// initialize the poti as an input:
  for (int i = 0; i < NUM_POTIS; ++i) {
    pinMode(potis[i], INPUT);
  }

}

void waitMilliseconds(uint16_t msWait)
{
  uint32_t start = millis();

  while ((millis() - start) < msWait)
  {
    // calling mp3.loop() periodically allows for notifications
    // to be handled without interrupts
    mp3.loop();
    delay(1);
  }
}



void buttonUpdatePlay() {
  for (int i = 0; i < NUM_BUTTONS; ++i) {
    int reading = digitalRead(buttons[i]);

    // Debounce setings: Das Array mit den vorherigen Button States bekommt
ein Update
```

```
    if (reading != prevButtonStates[i]) {
      lastDebounceTime = millis();
      prevButtonStates[i] = reading;
    }
    if ((millis() - lastDebounceTime) > debounceDelay) { // wenn die
debounce zeit vergangen ist
      if (reading != currentButtonStates[i]) { //überprüfe, ob button state
konstant ist
        currentButtonStates[i] = reading; // wenn nicht, dann neues Update
        if (currentButtonStates[i] == LOW) {
          mp3.playMp3FolderTrack(i + 1);
          Serial.println(i);

        }
        //        if (currentButtonStates[i] == HIGH) {
        //        }
      }
    }
  }
}


// Poti Update
void potiUpdateVolume() {
  int newVolume;
  for (int i = 0; i < NUM_POTIS; ++i) {
    currentPotiWert[i] = analogRead(potis[i]);
    newVolume = map(currentPotiWert[i], 0, 1023, 0, 30);
    if (newVolume != volume) {
      mp3.setVolume(newVolume);
      Serial.println(newVolume);
      volume = newVolume;
    }
  }
}


void loop() {

  buttonUpdatePlay();
  potiUpdateVolume();

}
```