

[arduino](#), [audio](#), [séquenceur](#), [optique](#), [em](#)

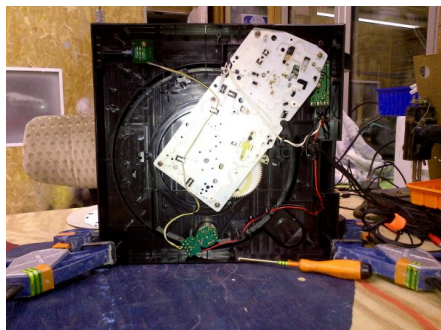
Page créée le 9 septembre 2020

Platine séquenceur

Transformation d'une platine disque en séquenceur optique : des capteurs posés le long du bras de la platine mesurent la lumière qu'ils reçoivent. Le disque est en carton/papier sur lequel sont tracées des formes au feutre.

Platine disque

La platine est une Pioneer PL-X11Z. Elle est conçu pour être alimentée en étant connectée à la chaine hifi par un mini-jack, en 12V. Elle fonctionne mais il manque la courroie. Dans un premier temps, on remplace la courroie manquante par un élastique assez grand, à section carrée.



Petits calculs

Un tour complet du plateau s'effectue en 1333.33 millisecondes (en position 45 tours/minute) et 1818 millisecondes en position 33 tours/minute Avec un disque de 30 cm, la circonférence extérieure est de 94 cm ($2 * \pi * r$)

En 33 tours / minute :

Si on divise le disque en 4 parties égales, chacune occupe 454 millisecondes soit un tempo de 132 BPM ($60 / 0.454\text{ms}$), en deux parties égales : BPM 66, etc.

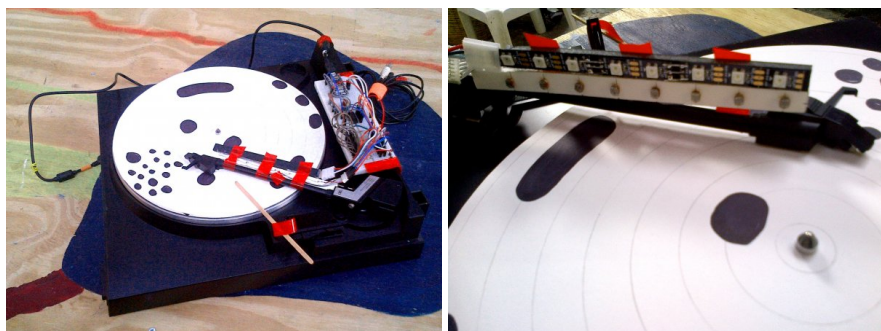
En 45 tours minute :

4 parties : chacune 333.33 ms soit un BPM de $180 / 2 \text{ parties} = \text{BPM de } 90$

Système

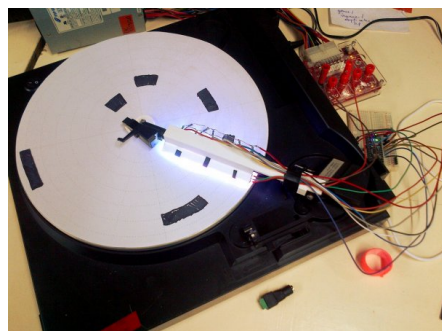
bras de la platine avec capteur → multiplexeur → arduino -(usb-série)→ ordi avec patch pure data

Premier prototype

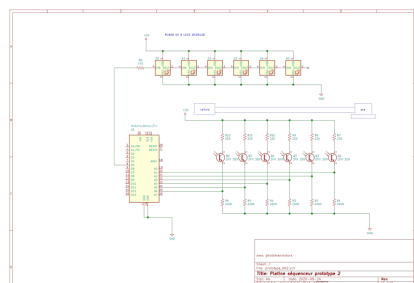


Problème : les photorésistances sont un peu lentes

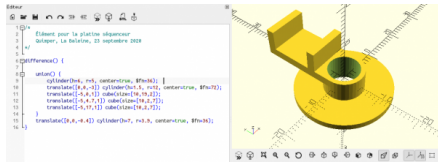
A l'origine la platine est alimentée en 12V par la chaîne hifi, après modification nous avons installé une alimentation directe par bloc secteur / transfo 12V et un interrupteur de marche-arrêt.



Dans cette version, 6 phototransistors sont utilisés, sans multiplexeur. A chaque phototransistor est associé deux résistances (en reprenant les valeurs définies dans le projet de Yunchi Luo et Mengliang Yu de l'université Cornell, voir sources en bas de page)



Repose-bras (ou quelque chose comme ça)



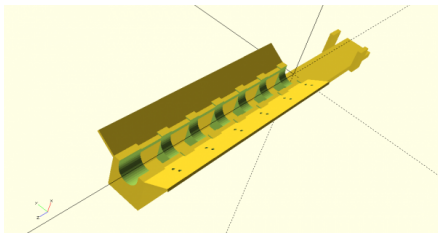
platine_squenceur_porte_bras.stl

platine_squenceur_porte_bras.scad (cliquer pour afficher le code)

[platine_squenceur_porte_bras.scad](#)

```
/*
  Élément pour la platine séquenceur
  Quimper, La Baleine, 23 septembre 2020
  OpenSCAD version 2019.05 @ kirin / Debian 9.5
*/
difference() {
    union() {
        cylinder(h=6, r=5, center=true, $fn=36);
        translate([0,0,-3]) cylinder(h=1.5, r=12, center=true, $fn=72);
        translate([-5,0,1]) cube(size=[10,19,2]);
        translate([-5,4.7,1]) cube(size=[10,2,7]);
        translate([-5,17,1]) cube(size=[10,2,7]);
    }
    translate([0,0,-0.4]) cylinder(h=7, r=3.9, center=true, $fn=36);
}
```

Adaptateur pour les capteurs



platine_squenceur_bras_porte_capteur.stl

platine_squenceur_bras_porte_capteur.scad (cliquer pour afficher le code)

[platine_squenceur_bras_porte_capteur.scad](#)

```
/*
  Élément pour la platine séquenceur
  bras porte capteur
  Quimper, La Baleine, 23 septembre 2020
  OpenSCAD version 2019.05 @ kirin / Debian 9.5
*/
difference() {
    color("Yellow") {
        difference() {
            translate([-10,-7.5,0]) cube(size=[12,15,100]);

            #union() {
                translate([0,0,10]) cube(size=[30,30,10], center=true);
                translate([0,0,25]) cube(size=[30,30,10], center=true);
                translate([0,0,40]) cube(size=[30,30,10], center=true);
                translate([0,0,55]) cube(size=[30,30,10], center=true);
                translate([0,0,70]) cube(size=[30,30,10], center=true);
                translate([0,0,85]) cube(size=[30,30,10], center=true);
            }
        }
        translate([-12,-7.5,-70]) cube(size=[3,15,170]);
        color("Lime") translate([-12,-7.5,0]) cube(size=[13.5,3,100]);
        color("Lime") translate([-12,4.5,0]) cube(size=[13.5,3,100]);
    }
    # color("Cyan") {
        translate([0,0,-1]) cylinder(h=102, r=4.5, center=false, $fn=36);
        translate([0,-4.5,-1]) cube(size=[9,9,102]);
    }
}
```

```

}
color("Blue") translate([-12,-9.49,-56]) cube(size=[12,2,8]);
color("Blue") translate([-12,7.49,-56]) cube(size=[12,2,8]);

* color("Green") translate([0,4.5,0]) rotate([0, 0, -45]) translate([0, 1, 0]) cube(size=[1,10,80]);
* color("Green") translate([0.5,-3.5,0]) rotate([0, 0, -135]) cube(size=[1,13,100]);

/* barre sans trou
color("Red") translate([3.25,-10.4,50]) rotate([0, 0, 30]) cube(size=[1,13,100],center=true);
*/

color("Red") translate([6.5,-16.85,0]) rotate([0, 0, 30]) plaque_phototransistor();

color("Red") translate([3.25,10.4,50]) rotate([0, 0, 150]) cube(size=[1,13,100],center=true);

module plaque_phototransistor() {
  difference() {
    cube(size=[1,13,100],center=false);
    #union() {
      rotate([0,90,0]) translate([-6,5,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
      rotate([0,90,0]) translate([-6,8,0]) cylinder(h=3, r=0.6, center=true, $fn=8);

      rotate([0,90,0]) translate([-22.5,5,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
      rotate([0,90,0]) translate([-22.5,8,0]) cylinder(h=3, r=0.6, center=true, $fn=8);

      rotate([0,90,0]) translate([-39,5,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
      rotate([0,90,0]) translate([-39,8,0]) cylinder(h=3, r=0.6, center=true, $fn=8);

      rotate([0,90,0]) translate([-55.5,5,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
      rotate([0,90,0]) translate([-55.5,8,0]) cylinder(h=3, r=0.6, center=true, $fn=8);

      rotate([0,90,0]) translate([-72,5,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
      rotate([0,90,0]) translate([-72,8,0]) cylinder(h=3, r=0.6, center=true, $fn=8);

      rotate([0,90,0]) translate([-89,5,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
      rotate([0,90,0]) translate([-89,8,0]) cylinder(h=3, r=0.6, center=true, $fn=8);
    }
  }
}
}

```

Code d'envoi

Le code arduino comprend : une phase de calibration, la mesure des capteurs et l'envoi des données vers l'ordinateur, en série

platine_sequenceur_003.ino (cliquer pour afficher le code)

[platine_sequenceur_003.ino](#)

```

/*
 * Platine sequenceur / prototype 002
 * http://lesporteslogiques.net/wiki/openatelier/projet/platine_sequenceur
 * Quimper, La baleine, 24 sept 2020
 * Debian 9.5 @ kirin / arduino 1.8.5
 * + library Adafruit NeoPixel 1.1.3 https://github.com/adafruit/Adafruit_NeoPixel
 *
 * CIRCUIT
 * - 6 phototransistors reliés aux entrées analogiques
 * - ruban de 8 leds RGB
 *
 * MODES DE FONCTIONNEMENT (à régler dans le code)
 * Mode de réception des données (les données envoyées ne sont pas formatées de la même manière)
 * 0 pour le mode de test (traceur serie de l'arduino IDE)
 * 1 pour le mode de réception dans pure data
 *
 * VERSIONS
 * 001 : (prototype 001, photorésistances) envoi de valeurs brutes en série
 * 002 : (prototype 001, photorésistances) ajout des leds + calibration
 * 003 : (prototype 002, phototransistors)
 *
 * TODO
 * ajouter un bouton de calibration
 * ajouter un switch de mode 0 ou 1
 * ajouter une led (clignote = calibration, éteinte mode 0, allumée mode 1)
 *
 * RESSOURCES
 * - traitements de lissage des données : https://www.openprocessing.org/sketch/686436
 */

int MODE = 0;

// Inclure les bibliothèques de fonction (libraries) nécessaires
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
#include <avr/power.h>
#endif

```

```

#define BROCHE_PT1  A1    // Broche reliée au phototransistor 1
#define BROCHE_PT2  A2    // Broche reliée au phototransistor 2
#define BROCHE_PT3  A3    // Broche reliée au phototransistor 3
#define BROCHE_PT4  A4    // Broche reliée au phototransistor 4
#define BROCHE_PT5  A5    // Broche reliée au phototransistor 5
#define BROCHE_PT6  A6    // Broche reliée au phototransistor 6

#define BROCHE_LED   5     // A quelle broche est relié le ruban de LEDs ?
#define NUMPIXELS    6     // Combien de LEDs sur le ruban ?

// Créer l'objet correspondant au ruban de LEDs
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, BROCHE_LED, NEO_RGB + NEO_KHZ800);
int luminosite
    = 255;

int v1b, v2b, v3b, v4b, v5b, v6b; // valeurs brutes
int v1l, v2l, v3l, v4l, v5l, v6l; // valeurs lissées
int v1c, v2c, v3c, v4c, v5c, v6c; // valeurs calibrées

boolean CALIBRATION = true;
long v1s, v2s, v3s, v4s, v5s, v6s; // sommes utilisées pour la calibration
int v1i, v2i, v3i, v4i, v5i, v6i; // valeurs d'initialisation définies pendant la phase de calibration
int calibration_start;           // démarrage de la calibration à cette milliseconde!
int calibration_compteur = 0;    // utilisé pour le calcul réactualisé des moyennes

void setup() {

    pixels.begin(); // Initialiser l'objet du ruban de leds

    Serial.begin(57600);

    // Fixer la luminosité pour l'ensemble du ruban
    pixels.setBrightness(luminosite);
    // Définir une couleur identique pour chaque LED, la LED 0 est la plus proche des broches
    for (int i = 0; i < 6; i++) {
        pixels.setPixelColor(i, pixels.Color( 255, 255, 255 ));
    }
    pixels.show();

    delay(500);
    calibration_start = millis();
}

void loop () {

    if (CALIBRATION) {
        calibration_compteur ++;
        if (millis() - calibration_start > 3000) { // L'étape de calibration dure 3 secondes
            CALIBRATION = false;
            v1i = (int)(v1s / (calibration_compteur - 1) );
            v2i = (int)(v2s / (calibration_compteur - 1) );
            v3i = (int)(v3s / (calibration_compteur - 1) );
            v4i = (int)(v4s / (calibration_compteur - 1) );
            v5i = (int)(v5s / (calibration_compteur - 1) );
            v6i = (int)(v6s / (calibration_compteur - 1) );
            v1l = v1i;
            v2l = v2i;
            v3l = v3i;
            v4l = v4i;
            v5l = v5i;
            v6l = v6i;

        } else {
            v1s += analogRead(BROCHE_PT1);
            delayMicroseconds(3);
            v2s += analogRead(BROCHE_PT2);
            delayMicroseconds(3);
            v3s += analogRead(BROCHE_PT3);
            delayMicroseconds(3);
            v4s += analogRead(BROCHE_PT4);
            delayMicroseconds(3);
            v5s += analogRead(BROCHE_PT5);
            delayMicroseconds(3);
            v6s += analogRead(BROCHE_PT6);
            delayMicroseconds(3);
        }
    }

    if (!CALIBRATION) {

        // Récupérer les valeurs actuelles
        v1b = analogRead(BROCHE_PT1);
        delayMicroseconds(3);
        v2b = analogRead(BROCHE_PT2);
        delayMicroseconds(3);
        v3b = analogRead(BROCHE_PT3);
        delayMicroseconds(3);
        v4b = analogRead(BROCHE_PT4);
        delayMicroseconds(3);
        v5b = analogRead(BROCHE_PT5);
        delayMicroseconds(3);
    }
}

```

```

v6b = analogRead(BROCHE_PT6);
delayMicroseconds(3);

v1l = (0.85 * v1l) + (0.15 * v1b) ;
v2l = (0.85 * v2l) + (0.15 * v2b) ;
v3l = (0.85 * v3l) + (0.15 * v3b) ;
v4l = (0.85 * v4l) + (0.15 * v4b) ;
v5l = (0.85 * v5l) + (0.15 * v5b) ;
v6l = (0.85 * v6l) + (0.15 * v6b) ;

v1c = (v1l - v1i) * -1;
v2c = (v2l - v2i) * -1;
v3c = (v3l - v3i) * -1;
v4c = (v4l - v4i) * -1;
v5c = (v5l - v5i) * -1;
v6c = (v6l - v6i) * -1;

if (MODE == 0) {

  Serial.print(v1c);
  Serial.print(",");
  Serial.print(v2c);
  Serial.print(",");
  Serial.print(v3c);
  Serial.print(",");
  Serial.print(v4c);
  Serial.print(",");
  Serial.print(v5c);
  Serial.print(",");
  Serial.println(v6c);
  //Serial.println("");
}
if (MODE == 1) {
  Serial.print("photores ");

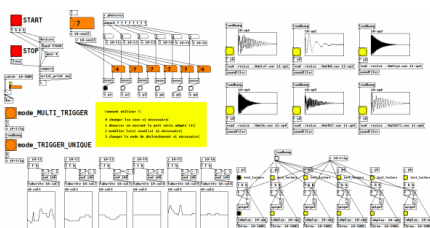
  Serial.print(v1c);
  Serial.print(" ");
  Serial.print(v2c);
  Serial.print(" ");
  Serial.print(v3c);
  Serial.print(" ");
  Serial.print(v4c);
  Serial.print(" ");
  Serial.print(v5c);
  Serial.print(" ");
  Serial.println(v6c);

}
delay(5);
}
}

```

Code réception

Le code pure data récupère les données série, et déclenche les sons en conséquence



platine_sequenceur_004_puredata.zip

Disque

Un modèle de disque à imprimer à l'échelle avec les pistes correspondant à la position des capteurs.

modele_disque_gradue_a3.pdf

Les rythmes sont à dessiner dans le sens inverse des aiguilles d'une montre (lévogyre!), le disque tournant dans le sens des aiguilles (dextrogyre)

Problèmes, améliorations, etc.

Le signal des phototransistors est très parasité

→ alimenter séparément les leds : **testé, et c'est beaucoup mieux**

→ utiliser la source de tension de référence 1.1V incluse dans l'arduino pour la capture analogique (plutôt que VCC) (pas testé)

→ traiter le signal (moyenne, etc) → envoyer des messages série plus courts

→ mesurer les temps pour trouver un timing précis

→ tous les phototransistors ne réagissent pas de la même manière : **réglé en ajoutant des seuils définissables dans le patch pure data**

Ajouter quelques composants complémentaires

- un bouton pour lancer une calibration à n'importe quel moment
- un switch pour basculer de mode "traceur série arduino" / "réception pure data"
- une led pour indiquer tout ça

Sources et ressources

Datasheet du phototransistor Osram Opto SFH 309 :

phototransistor_osram-opto_sfh309.pdf

Utilisation des phototransistors, un bon exemple :

https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2010/yl477_my288/yl477_my288/index.html

Groove Pizza : <https://apps.musedlab.org/groovepizza/?museid=ucEbu-1J6&>

Pocket Operations, a collection of drum patterns (pdf à télécharger sur <https://shittyrecording.studio>

rhythm patterns : <https://www.ethanhein.com/wp/2013/my-collection-of-transcribed-rhythm-patterns/>

Drum machine patterns : <http://808.pixll.de/>

Article extrait de : <http://www.lesporteslogiques.net/wiki/> - **WIKI Les Portes Logiques**

Adresse : http://www.lesporteslogiques.net/wiki/openatelier/projet/platine_sequenceur?rev=1601070155

Article mis à jour: **2020/09/25 23:42**