impression 3D, animatronique, marionnette, raspberry-pi, electronique, em

Tête de marionnette animatronique

(**Cette tête fait partie du projet Barbichette**)

Fabrication d'une tête animatronique. Le projet original est diffusé sous licence libre par Rolf Jethon et se compose d'une partie hardware : mécanique (pièces à imprimer, visserie, etc), électronique (raspberry pi ou orange pi, contrôleur de servomoteurs, servomoteurs, etc.) et d'une partie software développée par l'auteur (en perl!), les README donne des infos sur l'articulation du système logiciel.

Le système permet de synchroniser des mouvements de servomoteurs pré-enregistrés avec des fichiers audio MP3 et de les rejouer. 16 servomoteurs sont controlables grâce au driver PCA9685, l'audio est joué sur la sortie audio du Raspberry Pi. Les mouvements de servo sont enregistrés sur une base de temps de 50ms.

- Site du projet : https://bechele.de/?page_id=70
- Bouche et sourcils : https://www.thingiverse.com/thing:2863069
- ~~Paire d'yeux indépendants :~~ https://www.thingiverse.com/thing:2781756
- Paire d'yeux améliorée : https://www.thingiverse.com/thing:4058084
- Software : https://bechele.de/?page_id=73
- README du software : https://bechele.de/?page_id=188

Le projet est aussi présenté en détail sur ce blog https://zappedmyself.com/animatronics/bechele2-info/

Réglages particuliers

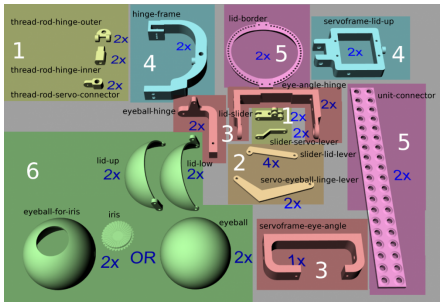
Avec le filament PLA Unite blanc, quelques pièces ont eu des difficultés d'impression : couche trop fragile au niveau des trous horizontaux. Quelques réglages d'impression adaptés : se baser sur le profil **fine 0.1mm**, changer : **couche d'impression 0.12mm, vitesse d'impression 40mm, flow 110%**

Yeux

Les fichiers sont fournis en stl sur thingiverse, je les ai réunis en 6 lots pour faciliter l'impression.
Le montage est expliqué, étape par étape dans cette vidéo : https://www.youtube.com/watch?v=U1c4R2EB83A

Impression

Différentes pièces :



Les fichiers sont slicés avec Cura pour Ender 3 avec une épaisseur de couches de 0.12mm

Durée d'impression

- lot 1 : 40 minutes
- lot 2 : 20 minutes
- lot 3 : 2h20

- lot 4 : 2h50
- lot 5 : 2h10
- lot 6 : 3h05 (avec eyeball-for-iris, mais pas eyeball)

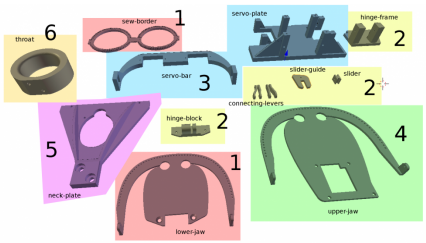
Un peu de casse au montage nécessite de nouveaux lots à imprimer

- lot 7 : 1h00 (eyeball hinge et hinge frame, 1 de chaque)

Tête

Les fichiers pour la tête sont disponibles sur thingiverse, en revanche pas de vidéo cette fois pour aider au montage...

Impression

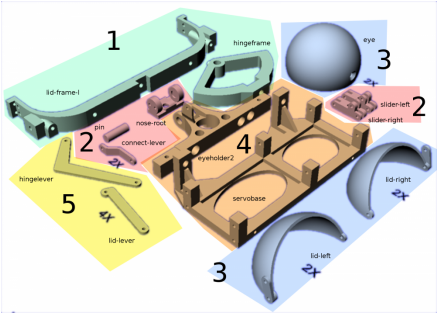


Durée d'impression

- lot 1 : 2h21
- lot 2 : 1h10
- lot 3 : 3h23
- lot 4 : 1h55
- lot 5 : 1h46
- lot 6 : 1h39
- lot 7 : 0h09 (oubli... connecting-levers)

Yeux 2

Ce sont les yeux animés qui vont avec la tête / fichiers fournis en stl sur thingiverse, réunis en 4 lots.



Impression

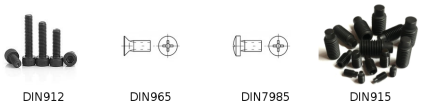
Durée d'impression
Sur la carte uSD : série de fichiers bechele_oeyes_X.gcode

- lot 1 : 1h31
- lot 2 : 0h30
- lot 3 : 3h02
- lot 4 : 2h12
- lot 5 : 0h11

Durée d'impression avec les réglages qui compensent le problème d'impression (série b)
Sur la carte uSD : série de fichiers bechele_oeyes_Xb.gcode

- 1 : 3h29
- 2 : 0h38
- 3 : 4h20
- 4 : 3h56
- 5 : 0h16

Composants et visserie



- un guide bien utile pour se repérer dans le monde merveilleux de la visserie : https://micro-modele.fr/img/cms/MICRO_VISSERIE/visserie_doc.pdf

Électronique

- Raspberry Pi 3 ou 4
- module électronique driver 16 canaux pour servo, à base de PCA9685 (différents possibles, exemple : HW-170 ou MotoPi)
- yeux améliorés : 7 servomoteurs miniatures, on utilise des DMS-MG90-A de DFRobot, avec une amplitude de 270°
- yeux d'origine : 5 servomoteurs
- tête : 5 servomoteurs
- alimentation 5V à x ampères : récupération d'une alim de PC

Quincaillerie

Test des servomoteurs avec arduino

Test de 4 servomoteurs avec du matériel grove et une alimentation de 550 mA (ancien chargeur de téléphone). L'alimentation est largement sous-dimensionnée pour utiliser 5 servomoteurs. En lisant la datasheet du servomoteur, on voit que son courant de décrochage (*stall currrent*) est de 800±30 mA à 4.8V et 1100±30 mA à 6V, on peut donc dimensionner 900mA pour alimenter chaque moteur (source)... Une alim de PC de récupération pourrait fournir largement ce qu'il faut.

test_servo.ino (cliquer pour afficher le code)

test_servo.ino

```
/* Test servomoteurs avec seeeduino lotus + module grove PCA9685 16-Chan I2C PWM driver

    arduino 1.8.5 @ Kirin, pierre@lesporteslogiques.net / 23 nov. 2022
      + lib. Seeed PCA9685 library, https://github.com/Seeed-Studio/Seeed_PCA9685

    Grove PCA9685 : https://wiki.seeedstudio.com/Grove-16-Channel_PWM_Driver-PCA9685
    Servo MG90 à 270° (DMS-MG90-A) https://www.dfrobot.com/product-1970.html
*/

#include "PCA9685.h"
#include <Wire.h>

ServoDriver servo;

void setup() {
  Wire.begin();  // join I2C bus
  Serial.begin(9600);
  servo.init(0x7f);
}

void loop() {
  // Test avec 4 servos
  for (int i = 1; i < 5; i++) {
    servo.setAngle(i, 0);
    delay(1000);
    servo.setAngle(i, 90);
    delay(1000);
  }
}
```

Le sketch ci-dessous est utile pour mettre les servo en position centrale avant de les inclure dans le montage :

raz_servo.ino (cliquer pour afficher le code)

raz_servo.ino

```
/*  Test / Remise à zéro de servomoteurs

    servomoteurs avec seeeduino lotus + module grove PCA9685 16-Chan I2C PWM driver
    pour programmer la lotus, choisir arduino uno dans l'IDE

    arduino 1.8.5 @ Kirin, pierre@lesporteslogiques.net / 7 déc. 2022
      + lib. Seeed PCA9685 library, https://github.com/Seeed-Studio/Seeed_PCA9685

    Grove PCA9685 : https://wiki.seeedstudio.com/Grove-16-Channel_PWM_Driver-PCA9685
    Servo MG90 à 270° (DMS-MG90-A) https://www.dfrobot.com/product-1970.html

    On peut relancer la procédure en faisant un reset de la carte
*/

#include "PCA9685.h"
#include <Wire.h>

ServoDriver servo;

boolean centerdone = false;

void setup() {
  Wire.begin();  // join I2C bus
  Serial.begin(9600);
  servo.init(0x7f);
}

void loop() {
  // Remettre les servos au centre
  if (!centerdone) {
    for (int i = 1; i < 6; i++) {
      servo.setAngle(i, 45);
      delay(2000);
      servo.setAngle(i, 135);
      delay(3000);
      servo.setAngle(i, 90);
      delay(2000);
    }
    centerdone = true;
  } else {
    delay(100);
  }
}
```

À noter : les servos ont une amplitude de 270°, la fonction servo.write() d'arduino prend en argument des valeurs entre 0 et 180. Dans le fichier Servo.h de la librairie servo, on peut trouver les valeurs extrêmes utilisées (ci-dessous), dont dans notre cas 0 correspond à -135° et 180 correspond à +135°... (Ce modèle de servo fonctionne entre 500 et 2500, mais je garde les valeurs prédéfinies, les servos n'auront pas besoin de parcourir toute leur amplitude)

```
#define MIN_PULSE_WIDTH       544     // the shortest pulse sent to a servo
#define MAX_PULSE_WIDTH      2400     // the longest pulse sent to a servo
#define DEFAULT_PULSE_WIDTH  1500     // default pulse width when servo is attached
```

Joystick

Pour enregistrer les mouvements des servos

Possible d'utiliser un de ces modèles en ajoutant les boutons

- https://www.thingiverse.com/thing:3250017
- https://www.thingiverse.com/thing:1276108
- https://www.thingiverse.com/thing:700346

On utilise une boite de dérivation électrique avec un joystick analogique (voir photo en bas de page)

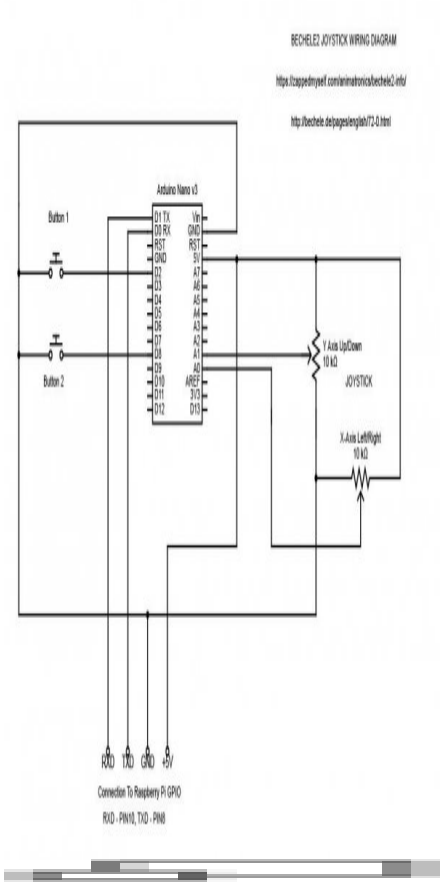


Schéma de PacketBob

code_arduino_joystick.ino (cliquer pour afficher le code)

code_arduino_joystick.ino

```
// Bechele2 Joystick Code V2.1 Sept 2021
//
// Arduino code for the joystick used for programming servo movements in Bechele2 animatronic software:
// http://bechele.de/pages/english/72-0.html
//
// Based on original code written by Rolf Jethon:
// http://bechele.de/pages/english/77-0.html
//
// For more info on building this joystck:
// https://zappedmyself.com/animatronics/bechele2-info/
//
// This code sends the Joystick X & Y values and the button status to the Raspberry Pi
// Each time the number 4 (ASCII value 52) is received from Raspberry Pi the data is sent
//
// Can run on any Arduino variant (I used a NANO clone)
// - Cleaned up comments and naming
// - Added code internal pullups to simplify wiring
// - Changed the joystick averaging code to get better range of ADC values


//***PIN ASSIGNMENTS***//
#define Y_PIN A0        // Analog input pin that the X axis pot is connected to
#define X_PIN A1        // Analog input pin that the Y axis pot is connected to
#define BUTTON1_PIN 2   // Digital input pin the START button is connected to
#define BUTTON2_PIN 3   // Digital input pin the STOP button is connected to


//***CUSTOMIZE VALUES***//
#define BAUD_RATE 19200       // Baud rate for serial port
#define ALPHA_VALUE 0.9       // Averaging factor (0.1 - 1.0) higher value = faster averaging


//***VARIABLE DECLARATION***//
int xAxisValue = 496;             // set X value to middle of possible range
int yAxisValue = 496;             // set Y value to middle of possible range
int xAxisMax = 1022;
int yAxisMax = 1022;
int xVal;
int yVal;
int inByte = 0;                   // incoming serial byte
float alphaFactor = ALPHA_VALUE;     // set to defined value
```

```
//***MICROCONTROLLER CONFIGURATION***//
void setup() {
  pinMode(BUTTON1_PIN, INPUT_PULLUP);
  pinMode(BUTTON2_PIN, INPUT_PULLUP);
  Serial.begin(BAUD_RATE);                    // Setup serial port speed
}


//***START OF MAIN LOOP***//
void loop() {
  xAxisValue = alphaFactor * analogRead(X_PIN) + (1 - alphaFactor) * xAxisValue;    // Get Xaxis value and average it
  delay(3);
  yAxisValue = alphaFactor * analogRead(Y_PIN) + (1 - alphaFactor) * yAxisValue;    // Get Yaxis value and average it
  int button1State = digitalRead(BUTTON1_PIN);                                      // Get Button 1 state
  int button2State = digitalRead(BUTTON2_PIN);                                      // Get Button 2 state

  xVal = xAxisMax - xAxisValue;
  yVal = yAxisMax - yAxisValue;

  // Test
  /*
  Serial.print(xVal);
  Serial.print(" ");
  Serial.print(yVal);
  Serial.print(" ");
  Serial.print(button1State);
  Serial.print(" ");
  Serial.println(button2State);

  */
  // Test 2 (graphique)
  Serial.print(xVal);
  Serial.print(",");
  Serial.print(yVal);
  Serial.print(",");
  Serial.print(button1State * 1000);
  Serial.print(",");
  Serial.println(button2State * 1000);

  delay(50);
  /*
    if (Serial.available() > 0) {                          // Check to see if serial data request has been received
    inByte = Serial.read();                                // Store serial data
    if (inByte == 52) {                                    // Send data values if ASCII '4' is received
      Serial.print(xAxisValue);
      Serial.print(" ");
      Serial.print(yAxisValue);
      Serial.print(" ");
      Serial.print(button1State);
      Serial.print(" ");
      Serial.println(button2State);
    }
    inByte = 0;                                            // Clear inByte value for next command
    }*/
}
```

Utilisation du module MotoPi RB-Moto3

Module pour Raspberry Pi : documentation / lien fabricant

En fait, je ne sais pas si ça peut marcher avec le montage, ce module fonctionne avec une librairie python, se connecte en I2C alors que le code du projet «Bechele» est en perl…

Arduino + PCA9685

Un module multiplexeur à base de PCA9685 permet de commander jusqu'à 16 servomoteurs, on peut chaîner plusieurs

module pour commander encore plus de servomoteurs. La communication avec arduino se fait en I2C.

Le condensateur électrochimique du module PCA9685 est adapté au nombre de servomoteurs utilisés : compter 100 µF par moteur

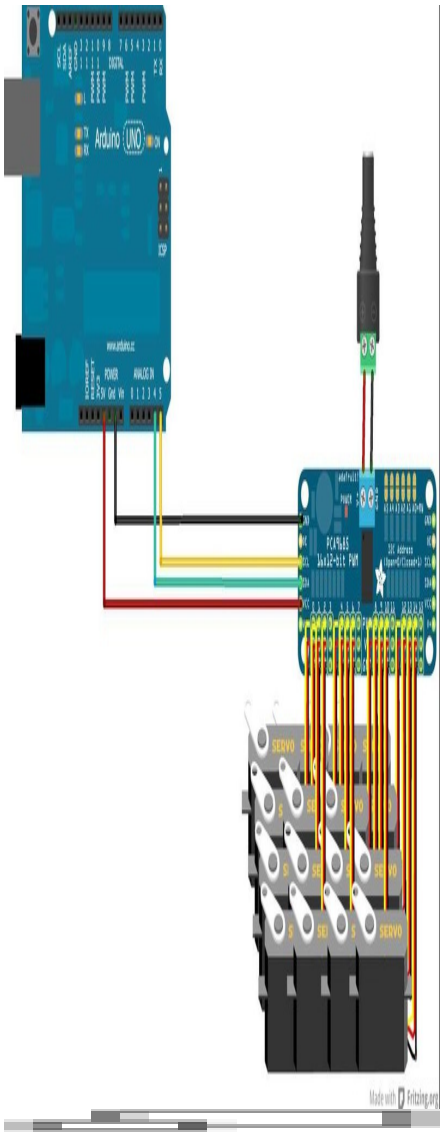


Photo Adafruit

Code d'exemple avec la lib. Adafruit PWM Servo:

arduino_servo_pca9685.ino (cliquer pour afficher le code)

arduino_servo_pca9685.ino

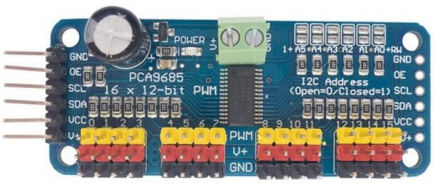
```
/*  Test servo

    arduino 1.8.5 @ Kirin, pierre@lesporteslogiques.net / 27 mars 2023
      + lib. Adafruit PWM Servo https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

    Réglages de la carte PCA9685
    Testé avec Grove Beginner Kit (vu Comme Arduino Genuino/Uno)

    Sans réglage, ça fonctionne corectement à l'adresse I2C : 0x40
 */

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

// Depending on your servo make, the pulse width min and max may vary, you
// want these to be as small/large as possible without hitting the hard stop
// for max range. You'll have to tweak them as necessary to match the servos you
// have!
```

```
#define SERVOMIN  150 // This is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  600 // This is the 'maximum' pulse length count (out of 4096)
#define USMIN  600 // This is the rounded 'minimum' microsecond length based on the minimum pulse of 150
#define USMAX  2400 // This is the rounded 'maximum' microsecond length based on the maximum pulse of 600
#define SERVO_FREQ 50 // Analog servos run at ~50 Hz updates

// our servo # counter
uint8_t servonum = 0;
uint8_t servonum_max = 1;

void setup() {
  Serial.begin(9600);
  Serial.println("8 channel Servo test!");

  pwm.begin();
  /*
   * In theory the internal oscillator (clock) is 25MHz but it really isn't
   * that precise. You can 'calibrate' this by tweaking this number until
   * you get the PWM update frequency you're expecting!
   * The int.osc. for the PCA9685 chip is a range between about 23-27MHz and
   * is used for calculating things like writeMicroseconds()
   * Analog servos run at ~50 Hz updates, It is importaint to use an
   * oscilloscope in setting the int.osc frequency for the I2C PCA9685 chip.
   * 1) Attach the oscilloscope to one of the PWM signal pins and ground on
   *    the I2C PCA9685 chip you are setting the value for.
   * 2) Adjust setOscillatorFrequency() until the PWM update frequency is the
   *    expected value (50Hz for most ESCs)
   * Setting the value here is specific to each individual I2C PCA9685 chip and
   * affects the calculations for the PWM update frequency.
   * Failure to correctly set the int.osc value will cause unexpected PWM results
   */
  pwm.setOscillatorFrequency(27000000);
  pwm.setPWMFreq(SERVO_FREQ);  // Analog servos run at ~50 Hz updates

  delay(10);
}

// You can use this function if you'd like to set the pulse length in seconds
// e.g. setServoPulse(0, 0.001) is a ~1 millisecond pulse width. It's not precise!
void setServoPulse(uint8_t n, double pulse) {
  double pulselength;

  pulselength = 1000000;   // 1,000,000 us per second
  pulselength /= SERVO_FREQ;   // Analog servos run at ~60 Hz updates
  Serial.print(pulselength); Serial.println(" us per period");
  pulselength /= 4096;  // 12 bits of resolution
  Serial.print(pulselength); Serial.println(" us per bit");
  pulse *= 1000000;  // convert input seconds to us
  pulse /= pulselength;
  Serial.println(pulse);
  pwm.setPWM(n, 0, pulse);
}

void loop() {
  // Drive each servo one at a time using setPWM()
  Serial.println(servonum);
  for (uint16_t pulselen = SERVOMIN; pulselen < SERVOMAX; pulselen++) {
    pwm.setPWM(servonum, 0, pulselen);
  }

  delay(500);
  for (uint16_t pulselen = SERVOMAX; pulselen > SERVOMIN; pulselen--) {
    pwm.setPWM(servonum, 0, pulselen);
  }

  delay(500);

  // Drive each servo one at a time using writeMicroseconds(), it's not precise due to calculation rounding!
  // The writeMicroseconds() function is used to mimic the Arduino Servo library writeMicroseconds() behavior.
  for (uint16_t microsec = USMIN; microsec < USMAX; microsec++) {
    pwm.writeMicroseconds(servonum, microsec);
  }

  delay(500);
  for (uint16_t microsec = USMAX; microsec > USMIN; microsec--) {
    pwm.writeMicroseconds(servonum, microsec);
  }

  delay(500);

  servonum++;
  if (servonum > servonum_max) servonum = 0; // Testing the first 8 servo channels
}
```

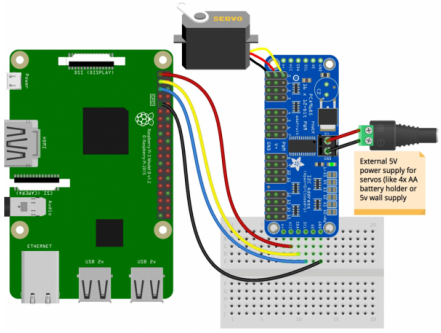
Rpi + Python + PCA9685

Le montage se compose d'un Raspberry Pi qui communique, via I2C sur ses broches GPIO, à un contrôleur de 16 servos basé sur la puce PCA9685. Les servosmoteurs sont reliés à ce circuit.

Multiplexeur PCA9685



Montage



Source du schéma : https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/hooking-it-up

Pour le brochage du Raspberry Pi 4, voir :
https://lesporteslogiques.net/wiki/materiel/raspberry_pi/start#raspberry_pi_4_model_b

Préparation du Raspberry Pi pour I2C

Installation de paquets

```
sudo apt-get install -y python-smbus  # support d'I2C dans python
sudo apt-get install -y i2c-tools     # entre autre pour scanner le port I2C
```

Configurer le support par le kernel

```
sudo raspi-config # choisir interface options/I2C/activer l'interface
```

Puis redémarrer

```
sudo reboot
```

Ensuite on peut tester que le module PCA9685 est bien branché

```
sudo i2cdetect -y 1  # Le port I2C numéro 1 est utilisé sur les rpi > 512 MB de RAM
```

Ce qui devrait afficher (les adresses des ports utilisés sont indiquées)

```
                         pi@tinycheck: ~ 80x24
pi@tinycheck:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
pi@tinycheck:~ $
```

Circuit complet

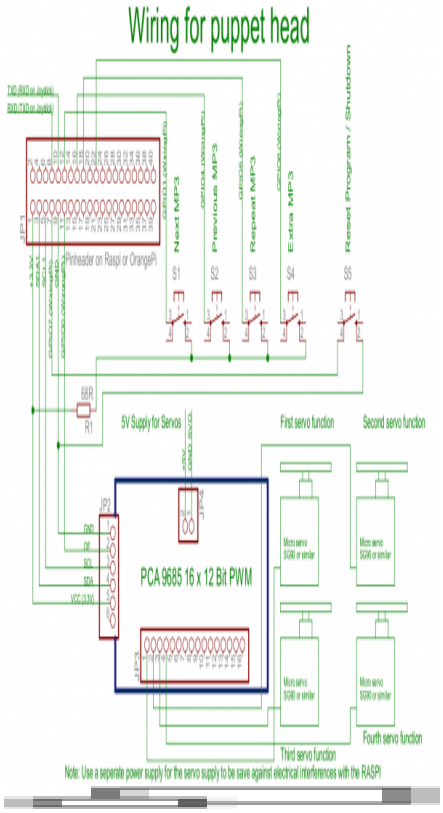


Schéma de Rolf Jethon

Communication I2C vers les servomoteurs

En utilisant Adafruit Servokit Library : https://docs.circuitpython.org/projects/servokit/en/latest/

- https://github.com/adafruit/Adafruit_CircuitPython_Bundle
- https://github.com/adafruit/Adafruit_CircuitPython_ServoKit

Installation

```
sudo pip3 install adafruit-circuitpython-servokit
python3.7 -m pip install adafruit-circuitpython-servokit
```

(Dans l'IDE geany, penser à changer les chemins vers l'exécutable de python, menu “construire” > “définir les outils de construction”)

test_servo_pca9685.py (cliquer pour afficher le code)

test_servo_pca9685.py

```
import time
from adafruit_servokit import ServoKit

# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=16)

kit.servo[0].angle = 180
#kit.continuous_servo[1].throttle = 1
time.sleep(1)
#kit.continuous_servo[1].throttle = -1
time.sleep(1)
kit.servo[0].angle = 0
#kit.continuous_servo[1].throttle = 0
```

Test des servomoteurs

Définir sur quelles broches sont reliés les servomoteurs, fixer minimum et maximum pour chacun.

test_python_pca9685_servomoteurs.py (cliquer pour afficher le code)

test_python_pca9685_servomoteurs.py

```
import time
from adafruit_servokit import ServoKit

# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=16)

# Correspondances (gauche/droite pour la tête)
# commands du côté droit inversées
# 0 : sourcil gauche          (50, 135)
# 1 : sourcil droit   (inv)     (135, 50)
# 2 : machoire basse              (?, ?)
# 3 : oeil gauche             (50, 130)
# 4 : oeil droit                 (50, 130)
# 5 : paupière gauche         ()            PROBLEME
# 6 : paupière droite         ()           PROBLEME
# 7 : bouche gauche               (50, 110)   PROB : refaire mécanisme
# 8 : bouche droite       (inv)    (110, 50)   PROB : refaire mécanisme
# 9 : yeux                        (50, 130)

for boucle in range(0, 3) :
    kit.servo[0].angle = 135
    kit.servo[1].angle = 50
    kit.servo[2].angle = 50
    kit.servo[3].angle = 130
    kit.servo[4].angle = 130
    # ~ kit.servo[5].angle = 80
    # ~ kit.servo[6].angle = 80
    kit.servo[7].angle = 50
    kit.servo[8].angle = 110
    kit.servo[9].angle = 50
    time.sleep(1)

    kit.servo[0].angle = 50
    kit.servo[1].angle = 135
    kit.servo[2].angle = 120
    kit.servo[3].angle = 50
    kit.servo[4].angle = 50
    # ~ kit.servo[5].angle = 100
    # ~ kit.servo[6].angle = 100
    kit.servo[7].angle = 110
    kit.servo[8].angle = 50
    kit.servo[9].angle = 130
    time.sleep(1)

    kit.servo[0].angle = 90
    kit.servo[1].angle = 90
    kit.servo[2].angle = 90
    kit.servo[3].angle = 90
    kit.servo[4].angle = 90
    # ~ kit.servo[5].angle = 90
    # ~ kit.servo[6].angle = 90
    kit.servo[7].angle = 90
    kit.servo[8].angle = 90
    kit.servo[9].angle = 90
    time.sleep(2)
```

Expressions

Test de quelques expressions du visage animatronique

expressions.py (cliquer pour afficher le code)

expressions.py

```
import time
from adafruit_servokit import ServoKit

# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=16)

# Correspondances (gauche/droite pour la tête)
# commands du côté droit inversées
# 0 : sourcil gauche          (50, 135)
# 1 : sourcil droit   (inv)     (135, 50)
# 2 : machoire basse              (?, ?)
# 3 : oeil gauche             (50, 130)
# 4 : oeil droit                 (50, 130)
# 5 : paupière gauche         ()            PROBLEME
# 6 : paupière droite         ()           PROBLEME
```

```
# 7 : bouche gauche               (50, 110)   PROB : refaire mécanisme
# 8 : bouche droite       (inv)    (110, 50)   PROB : refaire mécanisme
# 9 : yeux                        (50, 130)

def moue():
    kit.servo[0].angle = 50
    kit.servo[1].angle = 135
    kit.servo[2].angle = 120
    kit.servo[3].angle = 70
    kit.servo[4].angle = 110
    # ~ kit.servo[5].angle = 100
    # ~ kit.servo[6].angle = 100
    kit.servo[7].angle = 110
    kit.servo[8].angle = 50
    kit.servo[9].angle = 130
    time.sleep(2)

def sleep():
    kit.servo[0].angle = 90
    kit.servo[1].angle = 90
    kit.servo[2].angle = 90
    kit.servo[3].angle = 90
    kit.servo[4].angle = 90
    # ~ kit.servo[5].angle = 90
    # ~ kit.servo[6].angle = 90
    kit.servo[7].angle = 90
    kit.servo[8].angle = 90
    kit.servo[9].angle = 90
    time.sleep(2)

def louche():
    kit.servo[0].angle = 135
    kit.servo[1].angle = 50
    kit.servo[2].angle = 120
    kit.servo[3].angle = 130
    kit.servo[4].angle = 50
    # ~ kit.servo[5].angle = 100
    # ~ kit.servo[6].angle = 100
    kit.servo[7].angle = 110
    kit.servo[8].angle = 50
    kit.servo[9].angle = 130
    time.sleep(2)

louche()
moue()
sleep()
louche()
sleep()
```

Communication série avec le joystick arduino

Attention aux niveaux de tension si connexion directe aux broches GPIO 3V3 != 5V)

Dans le joystick, un arduino envoie des informations sur le port USB-série vers le raspberry pi, à 19200 bps
On peut vérifier que le port série est bien reconnu avec `lsusb`
Et utiliser `ls /dev/tty*` pour voir si le port utilisable Penser aussi à ajouter l'utilisateur au groupe `dialout` :

```
sudo adduser pi dialout
python3 -m pip install pyserial       # installer les bibliothèques
```

Ensuite ce sketch arduino :

python_serial_read.py (cliquer pour afficher le code)

python_serial_read.py

```
#!/usr/bin/env python3

# source : https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/
# arduino relié à /dev/ttyUSB0 (pour trouver le port : ls /dev/tty*)
# baud rate à 19200, correspond à celui défini dans arduino
# timeout : durée allouée à la lecture série
# readline() : lit jusqu'au caractère de fin de ligne
# decode('utf- 8') : transforme les bytes réçues dans le type souhaité
# rstrip() : retire les caractères de fin de ligne
import serial
if __name__ == '__main__':
    ser = serial.Serial('/dev/ttyUSB0', 19200, timeout=1)
    # vider le buffer série en début de communication
    ser.reset_input_buffer()
    while True:
        # y a t'il des données en attente ?
        if ser.in_waiting > 0:
```

```
line = ser.readline().decode('utf-8').rstrip()
print(line)
```

Voir aussi : https://www.aranacorp.com/fr/communication-serie-entre-raspberry-pi-et-arduino/

Pour adapter les niveaux logiques, voir :

- https://www.okdo.com/project/level-shifting/?ok_ts=1680009581943
- https://raspberrypi.stackexchange.com/questions/77176/raspberry-pi-gpio-input-voltage-limit

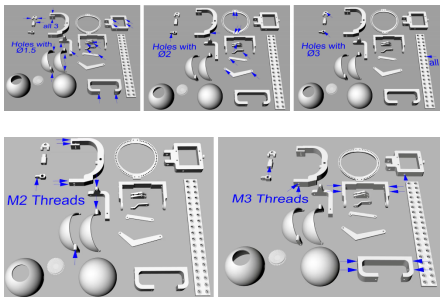
Alimentation

A base d'une alimentation d'ordinateur ATX à 24 broches : https://en.wikipedia.org/wiki/ATX#Power_supply

Montage

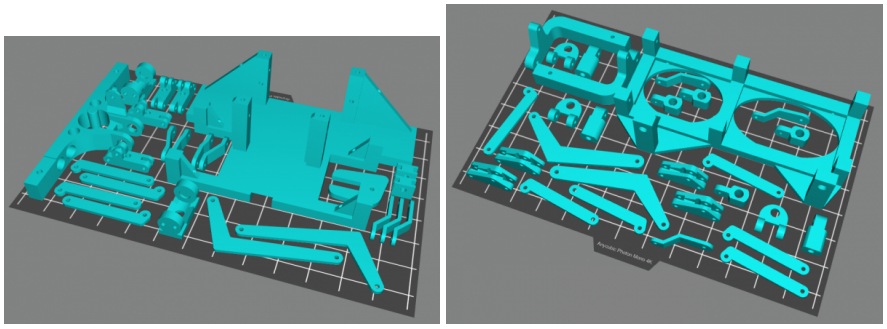
Pour le montage du mécanisme des yeux, se referer à cette vidéo : https://www.youtube.com/watch?v=U1c4R2EB83A

Il est nécessaire de percer et tarauder les pièces avant le montage :

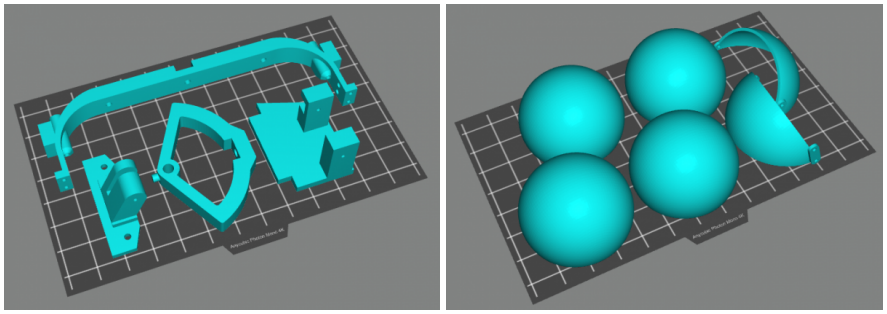


Impression résine

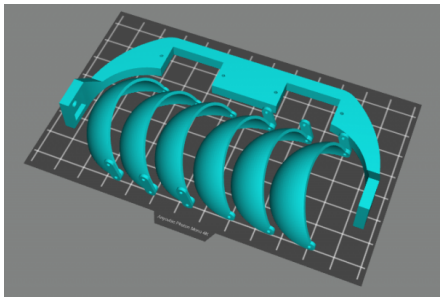
On imprime des pièces en résine avec la anycubic photon mono 4K (Ces pièces sont trop fragiles en impression 3D PLA). Deux plateaux sont préparés pour l'impression.



Et comme ça marche plutôt bien, on en fait 2 autres!



Et un petit dernier. (nb: ça aurait été plus malin de grouper les pièces par catégorie sur chaque plateau)



Installation du raspberry pi

Le projet est fourni avec une image pour raspberry pi 3 mais vu que c'est difficile de se procurer ce genre de carte en ce moment, on part sur une installation manuelle sur un raspberry pi 4

Version du Pi 4 : Raspbian 10 buster (lsb_release -a)

téléchargement du fichier https://bechele.de/wp-content/uploads/2022/07/bechele2.tar.gz (depuis https://bechele.de/?page_id=80)

Installation des bibliothèques nécessaires

```
sudo apt update
sudo apt install cpanminus
sudo apt install wiringpi
sudo apt install alsa-utils
sudo apt install mpg123
sudo apt install i2c-tools
sudo apt install perl5 # déjà installé
sudo apt install perl-device-serialport # ne fonctionne pas
sudo apt install libdevice-serialport-perl
sudo cpanm strict
sudo cpanm Device::SerialPort
sudo apt install ncurses-base ncurses-bin
sudo apt install libncurses5-dev libncursesw5 libncursesw5-dev
sudo cpanm Curses::UI
sudo cpanm WiringPi::API
sudo cpanm File::Find::Rule
sudo cpanm Device::PWMGenerator::PCA9685
sudo cpanm Audio::Play::MPG123
sudo cpanm Time::HR
```

Installation des scripts

les fichiers téléchargés sont dans le dossier /home/pi/bechele, on les copie dans les bons dossiers de /usr avec sudo.

```
sudo cp /home/pi/bechele/usr/lib/systemd/system/runlive.service /usr/lib/systemd/system/
sudo cp -R /home/pi/bechele/usr/local/bin/* /usr/local/bin/
```

Les fichiers à copier dans /home sont copiés manuellement.

Configuration du Raspberry Pi

Il faut activer la communication I2C, pour cela

```
sudo raspi-config
# choisir dans le menu : interface, puis I2C
```

Test pour commander les moteurs en python

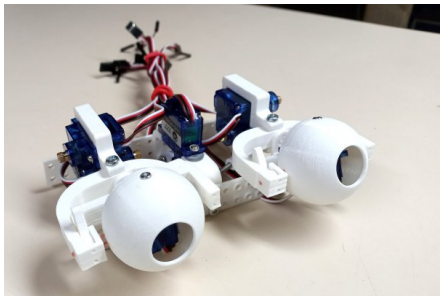
Une fois le PCA9685 relié à 2 moteurs et au rpi, le rpi configuré pour communiquer en I2C, un premier test en python

Journal

Journal partiel, pour poster quelques photos sur des étapes importantes

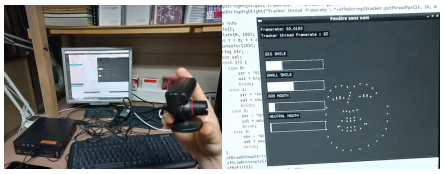
7 décembre 2022 : pour le software : A. teste les capacités du client léger pour voir s'il est capable de traiter de l'image vidéo en CV, plutôt oui! L'install. d'OpenFrameworks + CV n'est pas triviale mais à la fin ça fonctionne plutôt bien... pour le hardware : pas mal de petite quincaillerie pour laquelle il manque toujours une pièce, aujourd'hui c'était de tige filetée de

3mm, les moteurs sont installés, le prototype prend forme, ça mérite une photo.

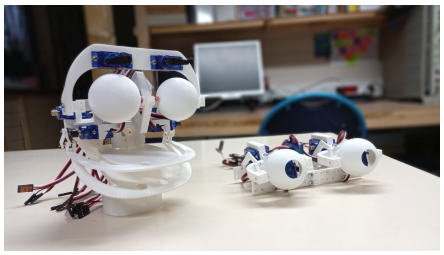


4 janvier 2023 : en continuant le montage, je casse une pièce fragile, à réimprimer donc… c'est compliqué de trouver des vis DIN915 mais on ne peut pas vraiment s'en passer (trouvées finalement sur la boutique en ligne d'un vendeur du marketplace d'amazon).

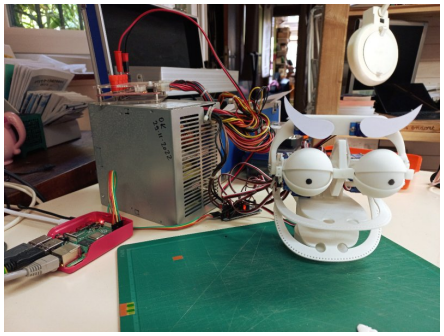
18 janvier 2023 : la reconnaissance de visage fonctionne bien sur le client léger qu'on envisageait! A. a écrit du code pour lisser les mouvements et différencier différents types de sourires. Montage du joystick pour piloter les yeux et enregistrer les séquences.



2 mars 2023 : **TODO** photos des prototypes d'animatronique, l'impression 3D en PLA est trop fragile, il va falloir penser à tout refaire en résine…



14 juin 2023 : électronique ok, prototype construit et actionné par script python depuis le raspberry, une vidéo des premiers essais d'expression



20230614_barbichette_prototype.mp4

Ressources

Sur l'animatronique : https://zappedmyself.com/animatronics/animatronic-projects/
Une autre paire d'yeux : https://www.instructables.com/DIY-Compact-3D-Printed-Animatronic-Eye-Mechanism/

Article extrait de : http://lesporteslogiques.net/wiki/ - **WIKI Les Portes Logiques**
Adresse : **http://lesporteslogiques.net/wiki/openatelier/projet/tete_animatronique?rev=1686759958**
Article mis à jour: **2023/06/14 18:25**