Boitier d'apprentissage du BRAILLE

Ce projet a pour but de faciliter l'apprentissage du braille à travers un boîtier interactif contrôlé par reconnaissance vocale, synthèse vocale . Il utilise des servomoteurs pour afficher les caractères en braille, et propose deux modes d'apprentissage différents. Il a été réalisé dans le cadre du projet de terminale STI2D.

Difficulté **Moyen** | Durée **72 heure(s)** | Catégories **Électronique, Bien-être & Santé, Robotique**

Coût **180 EUR (€)**

Sommaire

Introduction

Ce projet consiste en la réalisation d'un boîtier interactif permettant l'apprentissage du braille, à destination des personnes non-voyantes ou malvoyantes ou toutes personnes voulant apprendre le braille. Conçu dans le cadre d'un projet de Terminale STI2D, ce dispositif utilise des servomoteurs, un Raspberry Pi, et un système de reconnaissance vocale pour traduire des lettres en braille physique.
Le boîtier propose deux modes d'apprentissage :
-Un mode où l'utilisateur prononce une lettre, et le système la répète à voix haute avant de l'afficher en braille tactile.
-Un mode aléatoire où le boîtier choisit une lettre, la prononce, et laisse l'utilisateur deviner.
Le projet a été imaginé pour favoriser l'autonomie des personnes déficientes visuelles tout en rendant l'apprentissage du braille plus ludique et accessible. Il mêle électronique, programmation Python, impression 3D et design inclusif.

Matériaux

-1 raspberry pi 3
-6 servo moteurs
-1 shield Grove raspberry pi (Module Grove Base Hat 103030275)
-6 câbles Groves
-1 haut parleur (Module haut-parleur SKU00101)
-2 Câbles femelle femelle
-1 microphone (Microphone dongle USB ADA3367)
-1 Cordon Jack CA35M

Outils

Étape 1 - Installer le module pour créer un environnement virtuel

Si ce n'est pas déjà fait, installez le package **python3-venv** qui permet de créer un environnement virtuel Python :
sudo apt install python3-venv

Étape 2 - Créer et activer un environnement virtuel

Placez-vous dans le dossier où vous souhaitez créer votre projet (par exemple, **~/mon_projet_braille**) avec la commande :
cd ~/mon_projet_braille
Puis créez un environnement virtuel en remplaçant **nom_de_lenvironnement** par le nom que vous souhaitez donner à cet environnement :
python3 -m venv nom_de_lenvironnement
Activez ensuite cet environnement avec la commande :
source nom_de_lenvironnement/bin/activate

Étape 3 - Installer les bibliothèques Python nécessaires

Avec l'environnement virtuel activé, installez les bibliothèques suivantes :
pip install vosk pyaudio numpy

Étape 4 - Installer les dépendances système pour PyAudio

PyAudio nécessite des paquets système spécifiques pour fonctionner avec le microphone. Installez-les avec :
sudo apt install portaudio19-dev python3-pyaudio

Étape 5 - Installer le modèle Vosk pour la reconnaissance vocale

Pour que Vosk puisse effectuer la reconnaissance vocale en français, il faut télécharger et décompresser un modèle de langue.
Téléchargez le modèle ici :
https://alphacephei.com/vosk/models/vosk-model-small-fr-0.22.zip
Puis décompressez-le avec la commande :
unzip vosk-model-small-fr-0.22.zip
Ensuite mettez le modèle dans le même dossier que votre script Python.

Étape 6 - Désactiver l'environnement virtuel (optionnel)

Lorsque vous avez fini d'installer les bibliothèques et de travailler dans l'environnement virtuel, vous pouvez le désactiver avec la commande :
deactivate

Étape 7 - Code

Ce programme Python tourne sur un Raspberry Pi et permet de piloter les 6 servomoteurs qui affichent les lettres en braille, selon les commandes vocales de l'utilisateur.
Le système fonctionne en deux modes :
-Un mode d'apprentissage : l'utilisateur dit une lettre, elle est reconnue, prononcée et affichée en braille.
-Un mode test : le système choisit aléatoirement des lettres à afficher pour entraîner l'utilisateur.
Le code gère :

-la configuration des moteurs via les broches GPIO,
-la reconnaissance vocale (grâce au module Vosk),
-la synthèse vocale (avec espeak),
-la gestion des différents modes.
Ce script permet donc d'assurer à la fois l'interaction avec l'utilisateur, le traitement de la voix, et l'affichage mécanique du braille.

```
from vosk import Model, KaldiRecognizer
import pyaudio
import json
import RPi.GPIO as GPIO
import time
import subprocess
import random
import logging
import os
import sys
# Configuration du système de journalisation
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)
# Chemins de configuration - à ajuster selon votre environnement
MODEL_PATH = os.getenv("VOSK_MODEL_PATH", "/home/sin2025/models/vosk-model-small-fr-0.22")
AUDIO_PATH = os.getenv("AUDIO_PATH", "/home/sin2025/ressources")
# Configuration des GPIO pour les servomoteurs
servo_pins = [18, 26, 16, 24, 5, 22]
# Définition des positions des servos pour représenter les lettres en braille
definitions_braille = {
"A": [1, 0, 0, 0, 0, 0], "B": [1, 1, 0, 0, 0, 0],
"C": [1, 0, 0, 1, 0, 0], "D": [1, 0, 0, 1, 1, 0],
"E": [1, 0, 0, 0, 1, 0], "F": [1, 1, 0, 1, 0, 0],
"G": [1, 1, 0, 1, 1, 0], "H": [1, 1, 0, 0, 1, 0],
"I": [0, 1, 0, 1, 0, 0], "J": [0, 1, 0, 1, 1, 0],
"K": [1, 0, 1, 0, 0, 0], "L": [1, 1, 1, 0, 0, 0],
"M": [1, 0, 1, 1, 0, 0], "N": [1, 0, 1, 1, 1, 0],
"O": [1, 0, 1, 0, 1, 0], "P": [1, 1, 1, 1, 0, 0],
"Q": [1, 1, 1, 1, 1, 0], "R": [1, 1, 1, 0, 1, 0],
"S": [0, 1, 1, 1, 0, 0], "T": [0, 1, 1, 1, 1, 0],
"U": [1, 0, 1, 0, 0, 1], "V": [1, 1, 1, 0, 0, 1],
"W": [0, 1, 0, 1, 1, 1], "X": [1, 0, 1, 1, 0, 1],
"Y": [1, 0, 1, 1, 1, 1], "Z": [1, 0, 1, 0, 1, 1]
}
# Vocabulaire de reconnaissance vocale avec correspondances phonétiques
mots_vers_lettres = {
"ah": "A", "a": "A",
"mais": "B", "b": "B",
"c'est": "C", "c": "C",
"des": "D", "d": "D",
"euh": "E", "eux": "E", "e": "E",
"f": "F",
"j'ai": "G", "g": "G",
"hache": "H", "h": "H",
"i": "I", "aïe": "I", "hey": "I",
"j": "J",
"k": "K", "ca": "K",
"l": "L", "elle": "L",
"m": "M",
"n": "N",
"o": "O", "oh": "O", "eau": "O",
"p": "P", "pet": "P", "paix": "P",
"q": "Q", "cul": "Q",
"r": "R", "air": "R",
"s": "S",
"the": "T", "t": "T",
"u": "U", "hu": "U",
```

```
"v": "V", "vais": "V",
"w": "W", "double v": "W",
"x": "X",
"y": "Y", "ii grec": "Y", "les grecs": "Y", "grec": "Y", "le grec": "Y",
"z": "Z",
"mode un": "Mode 1",
"mode deux": "Mode 2",
"mode de": "Mode 2",
"fin": "Fin"
}
class BrailleReader:
def __init__(self):
# Initialiser les servomoteurs
self.initialize_servos()

# Initialiser l'audio et la reconnaissance vocale
self.initialize_audio()

# Mode actuel de l'application (0 = sélection de mode, 1 = Mode 1, 2 = Mode 2)
self.current_mode = 0

# Indicateur si l'application est en cours d'exécution
self.running = True

# Indicateur si le système est en train de parler
self.is_speaking = False

# Heure de fin de la dernière synthèse vocale (pour éviter l'auto-écoute)
self.last_speech_time = 0

# Délai minimum après synthèse vocale pour éviter l'auto-écoute (en secondes)
self.speech_cooldown = 1.0

logger.info("Lecteur Braille initialisé et prêt")
def initialize_servos(self):
"""Initialiser les GPIO et les servomoteurs"""
logger.info("Initialisation des GPIO et servomoteurs")

# Nettoyer toutes les configurations GPIO précédentes
GPIO.cleanup()

# Configurer le mode GPIO
GPIO.setmode(GPIO.BCM)

# Initialiser les servomoteurs
self.servos = []
for pin in servo_pins:
GPIO.setup(pin, GPIO.OUT)
servo = GPIO.PWM(pin, 50) # Fréquence PWM de 50Hz
servo.start(0)
self.servos.append(servo)

# Remettre tous les servos en position 0
self.reset_servos()

logger.info("Servomoteurs initialisés")
def initialize_audio(self):
"""Initialiser l'entrée audio et la reconnaissance vocale"""
logger.info("Initialisation de l'audio et de la reconnaissance vocale")

try:
# Initialiser le modèle Vosk
self.model = Model(MODEL_PATH)
```

```
# Créer la chaîne de vocabulaire à partir du dictionnaire mots_vers_lettres
vocabulaire = "[ \"" + "\", \"".join(mots_vers_lettres.keys()) + "\" ]"

# Initialiser le reconnaisseur avec le vocabulaire
self.rec = KaldiRecognizer(self.model, 44100, vocabulaire)

# Initialiser PyAudio
self.p = pyaudio.PyAudio()
self.stream = self.p.open(
format=pyaudio.paInt16,
channels=1,
rate=44100,
input=True,
frames_per_buffer=8000
)
self.stream.start_stream()

logger.info("Audio et reconnaissance vocale initialisés")
except Exception as e:
logger.error(f"Erreur lors de l'initialisation audio: {e}")
sys.exit(1)
def set_servo_angle(self, servo, angle):
"""Définir l'angle du servomoteur"""
duty_cycle = (angle / 18) + 2
servo.ChangeDutyCycle(duty_cycle)
time.sleep(0.3)
servo.ChangeDutyCycle(0)
def reset_servos(self):
"""Remettre tous les servos en position 0"""
for servo in self.servos:
self.set_servo_angle(servo, 0)
def afficher_braille(self, lettre):
"""Afficher une lettre en Braille en utilisant les servomoteurs"""
if lettre in definitions_braille:
for i in range(6):
self.set_servo_angle(
self.servos[i],
90 if definitions_braille[lettre][i] else 0
)
return True
return False
def speak_text(self, text):
"""Utiliser espeak pour convertir le texte en parole"""
try:
# Marquer que nous commençons à parler
self.is_speaking = True

# Mettre en pause la reconnaissance pour éviter l'auto-écoute
self.toggle_recognition_stream(pause=True)

subprocess.run(
["espeak", "-v", "fr", "-s", "200", "-p", "100", "-a", "1000", text],
stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL
)

# Enregistrer quand nous avons fini de parler
self.last_speech_time = time.time()

# Marquer que nous avons fini de parler
self.is_speaking = False
```

```
# Attendre un court délai avant de reprendre la reconnaissance
time.sleep(0.5) # Courte attente pour s'assurer qu'espeak est complètement terminé

# Reprendre la reconnaissance
self.toggle_recognition_stream(pause=False)

# Ajouter un délai pour éviter la reconnaissance immédiate (auto-écoute)
time.sleep(0.5)

except Exception as e:
logger.error(f"Erreur avec la synthèse vocale: {e}")
self.is_speaking = False
self.toggle_recognition_stream(pause=False)
def play_audio(self, file_name):
"""Lire un fichier audio en utilisant ffmpeg"""
try:
# Marquer que nous commençons à parler
self.is_speaking = True

# Mettre en pause la reconnaissance pour éviter l'auto-écoute
self.toggle_recognition_stream(pause=True)

file_path = os.path.join(AUDIO_PATH, file_name)
if os.path.exists(file_path):
subprocess.run(
["ffmpeg", "-i", file_path, "-f", "alsa", "default"],
stdout=subprocess.DEVNULL,
stderr=subprocess.DEVNULL
)
else:
logger.warning(f"Fichier audio introuvable: {file_path}")
# Utiliser espeak comme solution de repli si le fichier n'est pas trouvé
self.speak_text(file_name.replace(".mp3", ""))

# Enregistrer quand nous avons fini de parler
self.last_speech_time = time.time()

# Marquer que nous avons fini de parler
self.is_speaking = False

# Attendre un court délai avant de reprendre la reconnaissance
time.sleep(0.5)

# Reprendre la reconnaissance
self.toggle_recognition_stream(pause=False)

except Exception as e:
logger.error(f"Erreur lors de la lecture audio: {e}")
self.is_speaking = False
self.toggle_recognition_stream(pause=False)
def toggle_recognition_stream(self, pause=True):
"""Mettre en pause ou reprendre le flux de reconnaissance audio"""
if pause:
self.stream.stop_stream()
else:
self.stream.start_stream()
def recognize_speech(self, timeout=5):
"""Écouter la parole et retourner le texte reconnu"""
# Si nous sommes en train de parler ou venons de finir, ne pas écouter encore
if self.is_speaking:
return None
```

```
# Vérifier si nous sommes encore dans la période de délai après avoir parlé
if time.time() - self.last_speech_time < self.speech_cooldown:
return None

start_time = time.time()
while time.time() - start_time < timeout:
data = self.stream.read(4000, exception_on_overflow=False)

if self.rec.AcceptWaveform(data):
result = json.loads(self.rec.Result())
if "text" in result and result["text"].strip():
recognized_text = result["text"].lower()
logger.debug(f"Recognized: {recognized_text}")
return recognized_text

return None
def mode_selection(self):
"""Interface de sélection du mode"""
logger.info("Entrée dans la sélection de mode")

# Remettre les servos à zéro
self.reset_servos()

# Message de bienvenue
self.speak_text("Bienvenue au système d'apprentissage de Braille")
self.speak_text("Dites Mode un ou Mode deux pour commencer")

while self.current_mode == 0 and self.running:
# Écouter pour la sélection de mode
recognized_text = self.recognize_speech()

if recognized_text:
if recognized_text in mots_vers_lettres:
command = mots_vers_lettres[recognized_text]

# Gérer la sélection de mode
if command == "Mode 1":
self.speak_text("Mode un sélectionné")
self.current_mode = 1
logger.info("Mode 1 sélectionné")
elif command == "Mode 2":
self.speak_text("Mode deux sélectionné")
self.current_mode = 2
logger.info("Mode 2 sélectionné")
elif command == "Fin":
self.speak_text("Au revoir")
self.running = False
else:
self.speak_text("Commande non reconnue")
else:
self.speak_text("Commande non reconnue")
def mode_1(self):
"""Mode 1: L'utilisateur dit une lettre, le système la prononce et l'affiche"""
logger.info("Démarrage du Mode 1")

# Introduction pour le Mode 1
self.speak_text("Mode un. Dites une lettre, je vais la prononcer et l'afficher en braille")
self.speak_text("Dites fin pour revenir au menu principal")

while self.current_mode == 1 and self.running:
# Attendre l'entrée d'une lettre
recognized_text = self.recognize_speech()
```

```
if recognized_text and not self.is_speaking:
if recognized_text in mots_vers_lettres:
command = mots_vers_lettres[recognized_text]

# Vérifier si l'utilisateur veut sortir
if command == "Fin":
self.speak_text("Fin du mode un")
self.current_mode = 0
logger.info("Sortie du Mode 1")
break

# Gérer l'affichage de la lettre
if len(command) == 1 and command in definitions_braille:
logger.info(f"Affichage de la lettre: {command}")

# Afficher d'abord la lettre en braille
self.afficher_braille(command)

# Puis prononcer la lettre avec reconnaissance désactivée pour éviter l'auto-écoute
self.speak_text(command)
else:
self.speak_text("Ce n'est pas une lettre valide")
else:
self.speak_text("Commande non reconnue")
def mode_2(self):
"""Mode 2: Le système affiche aléatoirement des lettres"""
logger.info("Démarrage du Mode 2")

# Introduction pour le Mode 2
self.speak_text("Mode deux. Je vais afficher des lettres aléatoires en braille")
self.speak_text("Dites fin pour revenir au menu principal")

while self.current_mode == 2 and self.running:
# Afficher une lettre aléatoire
random_letter = random.choice(list(definitions_braille.keys()))
logger.info(f"Affichage de la lettre aléatoire: {random_letter}")

# Prononcer et afficher la lettre
self.afficher_braille(random_letter)
self.speak_text(random_letter)

# Attendre un moment pour permettre à l'utilisateur de dire "fin" si nécessaire
time.sleep(2)

# Vérifier les commandes vocales (même pendant la séquence d'affichage)
recognized_text = self.recognize_speech(timeout=1)

if recognized_text and recognized_text in mots_vers_lettres:
command = mots_vers_lettres[recognized_text]

if command == "Fin":
self.speak_text("Fin du mode deux")
self.current_mode = 0
logger.info("Sortie du Mode 2")
return
def cleanup(self):
"""Nettoyer les ressources à la sortie"""
logger.info("Nettoyage des ressources")

# Remettre les servos à zéro
self.reset_servos()
```

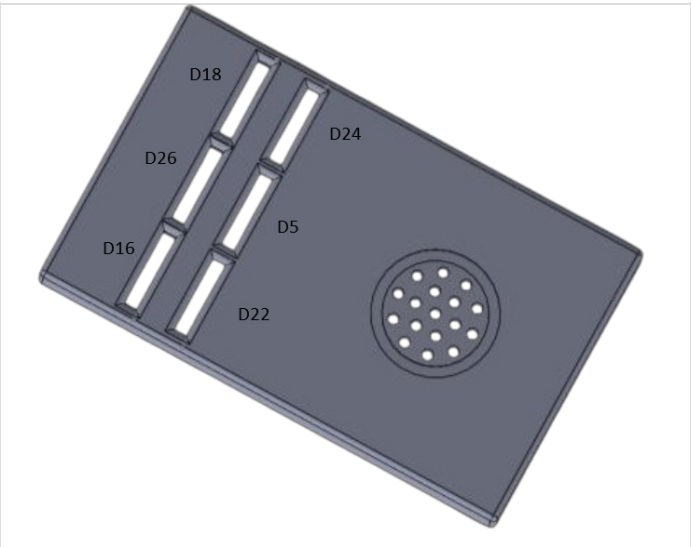
```
# Arrêter tous les servos
for servo in self.servos:
servo.stop()

# Nettoyer les GPIO
GPIO.cleanup()

# Arrêter et fermer le flux audio
if hasattr(self, 'stream'):
self.stream.stop_stream()
self.stream.close()

# Terminer PyAudio
if hasattr(self, 'p'):
self.p.terminate()

logger.info("Nettoyage terminé")
def run(self):
"""Boucle principale de l'application"""
logger.info("Démarrage de l'application Lecteur Braille")

try:
while self.running:
# Sélection de mode
if self.current_mode == 0:
self.mode_selection()
# Mode 1
elif self.current_mode == 1:
self.mode_1()
# Mode 2
elif self.current_mode == 2:
self.mode_2()
except KeyboardInterrupt:
logger.info("Application terminée par l'utilisateur")
except Exception as e:
logger.error(f"Erreur inattendue: {e}")
finally:
self.cleanup()
if __name__ == "__main__":
reader = BrailleReader()
reader.run()
```

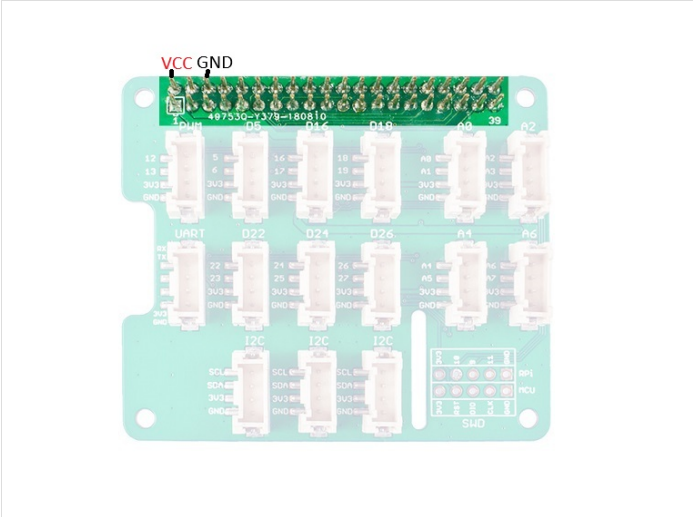
Étape 8 - Branchement

Branchez les servo-moteurs comme sur la photo à gauche sur les broches D18, D26, D16, D24, D5, D22.

Branchez le haut parleur sur la prise jack et sur les broches du shield comme sur l'image.

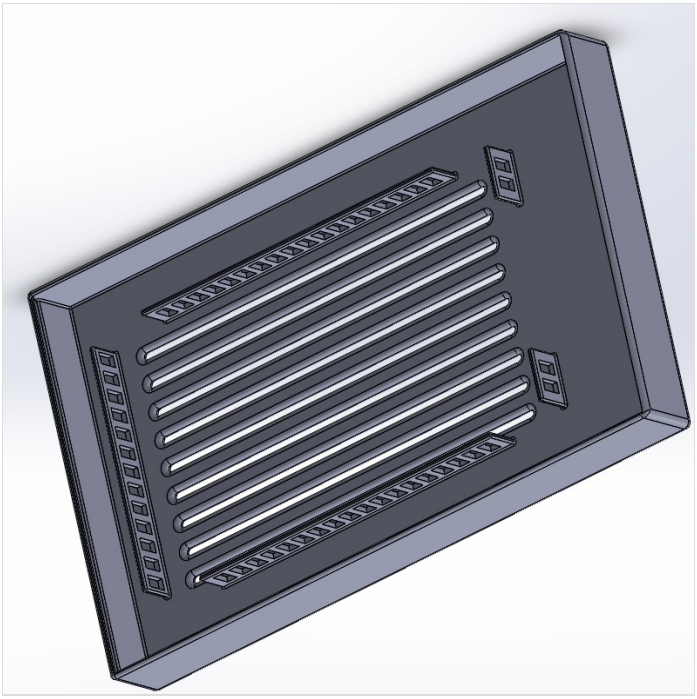
Branchez le Microphone sur un port USB.

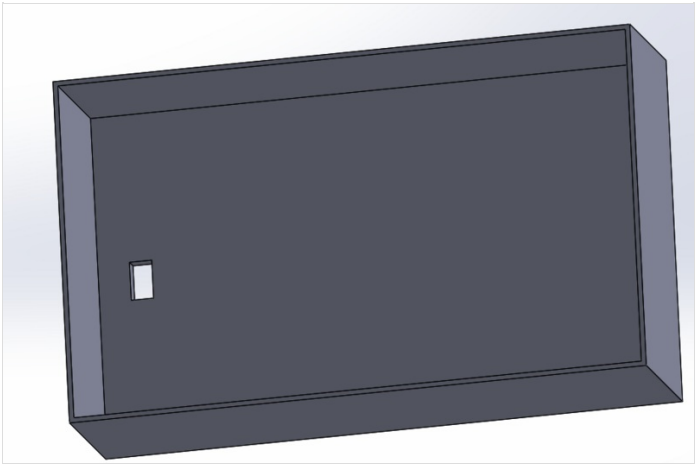


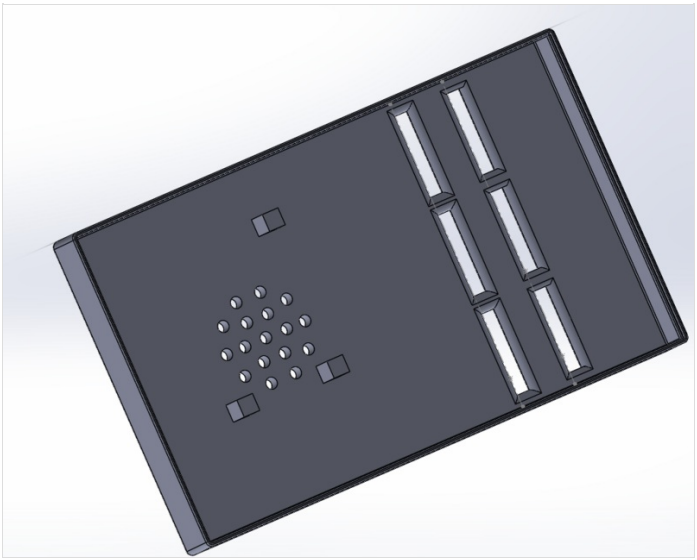


Étape 9 - Conception SolidWorks

Afin d'avoir un support pour les composants assurez vous de crée un boitier dimensionné en fonction de vos composants :
- Premièrement crée un socle qui accueillera la source d'énergie (batterie externe)
- Ensuite crée un boite qui servira de pièce intermédiaire et qui accueillera la carte Raspberry pi avec son microphone
-Enfin crée un couvercle en vous assurant de laisser apparaitre la matrice 3*2 et des sortis de son pour le haut parleur

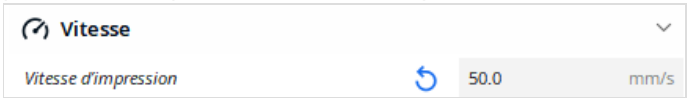


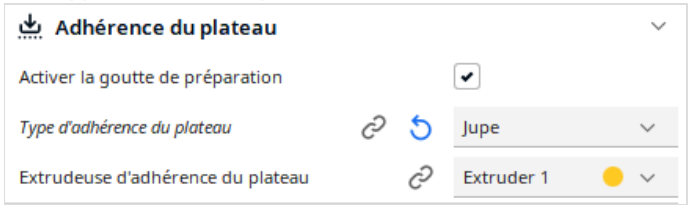


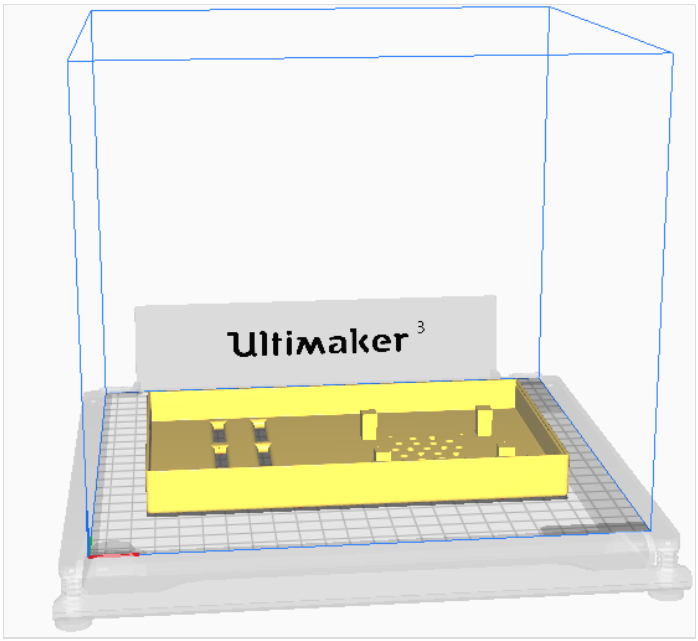


Étape 10 - Impression

Pour matérialisé notre boitier conçu précédemment sur SolidWorks munissez vous d'une imprimante 3D et du logiciel cura. Configuré les paramètres d'impression des différentes partis du boitier (vitesse de la buse, support) et lancé l'impression.

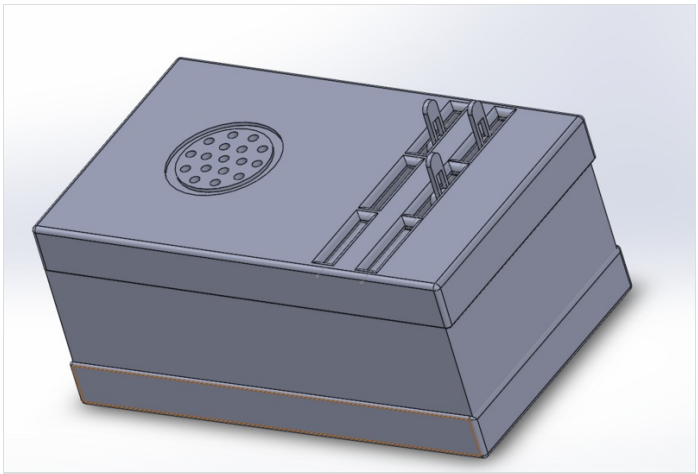




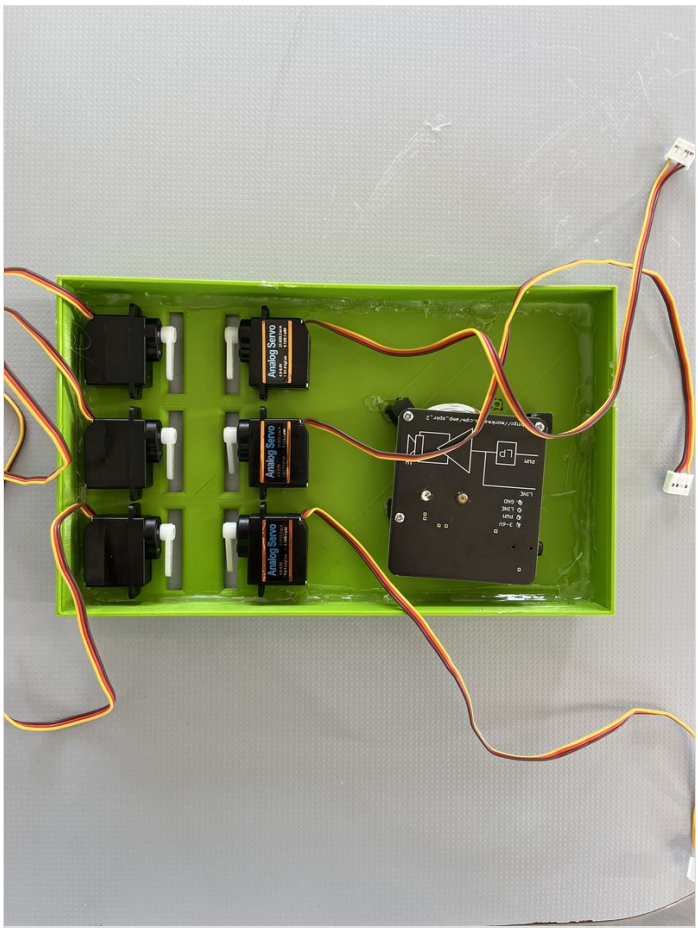


Étape 11 - Assemblage

Une fois toute les étapes précédente réalisé, placez tout les composant a leur place sans oublier des les fixer a l'aide de vis ou colle liquide, puis fermer le boitier à l'aide des clips précédemment fabriquer pendant l'étape "SOLIDWORKS "et testez votre système d'apprentissage du braille




Trxcom



Étape 12 - Conseils, erreurs à éviter et pistes d'amélioration

Lors de la réalisation du boîtier, il est important de bien tester chaque élément séparément avant de tout assembler. Les servomoteurs, le micro et le haut-parleur doivent être fonctionnels de manière indépendante pour éviter de perdre du temps en cas de problème. Il est aussi conseillé de bien calibrer les servos avant de fixer le couvercle ou les pièces internes, car un mauvais alignement peut fausser l'affichage en braille.
La reconnaissance vocale fonctionne mieux si le vocabulaire est restreint. Il est donc préférable de limiter les mots utilisés à ceux réellement nécessaires. De même, il faut penser à ajouter un petit délai après chaque message vocal pour éviter que le système ne se déclenche tout seul à cause de sa propre voix (problème de "self-listening").
Plusieurs erreurs sont à éviter. Par exemple, sous-estimer la puissance nécessaire pour les électroaimants peut mener à un système inefficace ou instable. C'est l'une des raisons pour lesquelles nous avons choisi les servomoteurs, plus simples à gérer. Autre point à surveiller : l'alimentation électrique. Si elle est mal dimensionnée, cela peut provoquer des coupures ou des bugs sur le Raspberry Pi. Il est aussi risqué de vouloir tout tester d'un coup sans passer par des essais progressifs, car cela rend le diagnostic des erreurs plus compliqué.
Pour améliorer le projet, plusieurs pistes sont possibles. Le site web (Mode 3), qui permet l'envoi de texte depuis un navigateur. On pourrait aussi imaginer un système avec batterie ou alimentation mobile pour rendre le boîtier totalement autonome. Enfin, des ajouts comme un retour sonore d'erreur ou des vibrations pourraient guider davantage l'utilisateur, surtout en cas de mauvaise commande.

Étape 13 - Crédits

Le Ray Gabin T15
Sylvestre Valentin T15
Mur Damien T13
2025
Lycée le Likes Quimper