Aurélien SENGER
Jérémy MANCEAUX
Jérémie SUZAN
Gery ZABLOCKI

une solution qui permet aux personnes malentendantes ou muettes de se faire comprendre

Notre système est composé de :

- un leap motion
- un appareil capable d'exécuter un programme en python 2.7
- un écran (affichage du signe interprété)
- une sortie son (diction du signe interprété)

Le TTS (Text To Speech) utilise les librairies d'accessibilité de base présentes sur presque tous les OS.

Les imports ci-dessous sont nécessaires au bon fonctionnement du programme :

```
from operator import itemgetter
import sys
sys.path.insert(0, "../lib")
import Leap, os, thread, time, pprint, marshal
import pyttsx
```

Pour utiliser l'API du LeapMotion en mode "Listener" il faut ré-implémenter l'interface Listener.

```
class SampleListener(Leap.Listener):
        def set_mode(self, mode):
        def on_connect(self, controller):
        def on_frame(self, controller):
        def get_frameMatrix(self):
```

Le code est composé des fonctions suivantes

```
def recordSign(signTable):
        #permet l'enregistrement d'un nouveau signe dans la base de données

def mean(val1, weight1, val2, weight2):
        #utilisé par recordSign pour un calcul de moyenne

def vectorToFloat(data):
        #transforme tous les vecteurs d'un enregistrement en tuples de float

def floatToVector(data):
        #fait l'opération inverse

def add_sign(sign, signs):
        #ajoute un signe sign(Leap.Vector) à la liste de signes signs (utilise des tuples
de float)

def load_signs():
        #charge les signes enregistrés dans la base de données (avec des tuples de float)

def store_signs(signs):
        #enregistre les signes dans la base de données (avec des tuples de float)

def save_sign(sign):
        #ajoute un signe à la base de données

def save_signs(list_sign):
        #ajoute une liste de signes à la base de données

def get_saved_signs():
        #recupère tous les signes sauvegardés dans la base de données

def distance(mvt1,mvt2):
        #calcule le carré de la distance entre deux frame de mouvement (Pythagore à n
dimensions)

def ressemblance(base,entree):
        #utilise distance en garantissant la correspondance des mains

def match(signed,signs):
        #cherche dans la base des signes celui qui correspond le plus à celui qui vient
d'être signé et retourne le mot correspondant

def sign_to_tab(frames):
        #transforme une liste de frame correspondant à un signe en données utilisables
par notre algorithme, c'est ici que sont définis les données pertinentes de chaque signe

def main():
        #fonction principale initialise le leapmotion
```

Fonctionnement général de l'algo

- enregistrement d'un signe sous forme d'une liste de toute les frames (structure de donnée décrivant la position de la main à un temps donné)
- découpage du signe en 11 intervalles pour unifier les signes
- extraction des données pertinentes des 10 changements d'intervalles et enregistrement dans un tableau de 10 lignes et 2 colonnes (une colonne par main)
- comparaison avec les enregistrement (réalisés de la même manière) dans la base de donnée
- la comparaison est une somme pas à pas des carrés des distances entre le nouveau signe et l'un de ceux enregistrés
- le signe reconnu est celui le plus ressemblant

Pour chaque main, les informations pertinentes sont les suivantes :

- rotation_axis : vecteur de rotation (sur les 3 axes)
- rotation_angle : angle global de rotation
- translation : vecteur de translation (sur les 3 axes)
- grab_strength : coefficient d'ouverture/fermeture de la main
- pinch_strength : écartement du pouce
- palm_normal : vecteur orientant la main

Le fonctionnement global de l'objet SignIt peut être découpé en plusieurs modes :

- **IDLE** : attend que l'utilisateur choisit un des deux modes suivants
- **Record** : va enregistrer 3 signes pour ajouter leur moyenne dans la base de données
- **Play** : reconnaît les signes faits par l'utilisateur

Le graphe suivant décrit les différents modes accessibles pour l'utilisateur :