Курсовий проект на тему “Розробка додатку для швидкого запуску програм голосом”

1. Актуальність завдання

В сучасному світі люди усе більше і більше звикають до інтрактивності девайсів. У мобільних пристроях давно живуть Google Assistent , Siri та інші помічники, що готові відповідати на питання голосом та реагувати на нашу мову. У світі десктоп пристроїв схожі помічники все ще не дуже поширені. У Windows Cortana, але оскільки операційна система не є безкоштовною, цей помічник не є доступним для кожного. Але користувачі Лінуксу також мають багато опцій розумних помічників, серед них такі як Mycroft, Kalliope, Stephanie, Jasper, Jarvis та багато інших. Вони пропонують такі функтції, як сторення нагадувань, нотаток, пошук, створення нагадувань та інші. У даній роботі була зроблена спроба розробити додаток, що так само зміг би розуміти голос людини і реагувати на нього. Додаток може бути використаний для оптимізації роботи із системою, може бути корисним людям, що мають обмежені можливості або просто надають перевагу аудіо шортказам замість візуальних.

2. Завдання роботи

Розробити програмне забеспечення (додаток), що дозволить запускати різні програми за допомогою голосових команд.

3. Вибір апаратного забезпечення

В якості апаратного забеспечення був використаний мікрофон, встановлений у ноутбуці - як приклад сенсору, який використовується мільйонами людей кожного дня.

4. Засоби розробки

Для розробки були використані такі засоби :

- мова програмування JavaScript,
- мова програмування python,
- бібліотеки SpeechRecognition та PyAudio,
- JavaScript Gnome Desktop environment API.

5. Опис використаних бібліотек для роботи із аудіо

Використана в проекті бібліотека PyAudio забезпечує прив'язку Python для PortAudio, крос-платформної бібліотеки аудіо-вводу-виводу. За допомогою PyAudio можна легко використовувати Python для відтворення та запису аудіо на різних платформах, таких як GNU / Linux, Microsoft Windows та Apple Mac OS X / macOS. PyAudio поширюється за ліцензією MIT.

Якщо говорити про розпізнавання мови в цілому. Розпізнавання мови сягає корінням у дослідження, проведені в лабораторіях Bell на початку 1950-х. Ранні системи були обмежені одним оратором і мали обмежений словниковий запас - близько десятка слів. Сучасні системи розпізнавання мови пройшли довгий шлях з часів своїх давніх аналогів. Вони можуть розпізнавати мовлення кількох мовців і мають величезний словниковий запас на багатьох мовах.

Перший крок до розпізнавання мови - це, звичайно, голос. Мова повинна бути перетворена з фізичного звуку в електричний сигнал за допомогою мікрофона, а потім у цифрові дані за допомогою аналого-цифрового перетворювача.

Більшість сучасних систем розпізнавання мовлення покладаються на те, що відоме як модель прихованого Маркова (HMM). Цей підхід працює на припущенні, що мовленнєвий сигнал при розгляді на досить короткому часовому масштабі (скажімо, десять мілісекунд) може бути розумно апроксимований як стаціонарний процес - тобто процес, в якому статистичні властивості не змінюються з часом.

У типовому HMM мовний сигнал розділений на 10-мілісекундні фрагменти. Спектр потужності кожного фрагмента, який по суті є графіком потужності сигналу як функції від частоти, відображається на вектор дійсних чисел, відомий як цепстральні коефіцієнти. Розмір цього вектора, як правило, невеликий - іноді і 10, хоча більш точні системи можуть мати розмір 32 і більше. Кінцевий вихід HMM є послідовністю цих векторів.

Для декодування мови в текст групи векторів узгоджуються з однією або кількома фонемами - фундаментальною одиницею мови. Це обчислення вимагає підготовки, оскільки звук фонеми різниться залежно від динаміка і навіть різниться від одного до іншого висловлювання тим самим динаміком. Потім застосовується спеціальний алгоритм для визначення найбільш вірогідного слова (або слів), що виробляють задану послідовність фонем.

Можна уявити, що весь цей процес може бути обчислювально дорогим. У багатьох сучасних системах розпізнавання мови нейронні мережі використовуються для спрощення мовного сигналу з використанням методів перетворення ознак та зменшення розмірності до розпізнавання HMM. Детектори голосової активності (VAD) також використовуються для зменшення звукового сигналу лише до тих частин, які можуть містити мову. Це запобігає марнуванню часу розпізнавача на аналіз непотрібних частин сигналу.

На щастя, такі системи уже побудовані і програмістам зазвичай не доводиться турбуватися ні про. Ряд служб розпізнавання мови доступні для використання в Інтернеті через API, і багато з цих служб пропонують пакети SDK для Python. У даній роботі було використано пакет SpeechRecognition.

Бібліотека SpeechRecognition виконує роль обгортки для декількох популярних мовних API і, отже, надзвичайно гнучка. Один з них - Google Web Speech API - підтримує ключ API за замовчуванням, який жорстко закодований у бібліотеці SpeechRecognition.

Гнучкість та простота використання пакета SpeechRecognition роблять його чудовим вибором для будь-якого проекту Python.

6. Опис роботи розробленого додатку (розширення)

Робота із додатком починається в момент встановлення додатку.

Користувач має описати у файлу конфігурацій ті команди і їх голосові тригери, які хоче використовувати.

Команди треба описувати у форматі

`<голосова команда> : <команда, яка буде запущена додатком> \n`

Одній команді відповідає один рядок. Дозволяється робити відступи між командами у вигляді пустих рядків. Дозволяється визначати декілька різних голосових команд для однієї команди, що буде запущена додатком (таким чином можна зменшити кількіть похибок, спричинених проблемами розпізнавання голосу). У якості команди, що запускає додаток має бути команда, яку можна використати для запуску програми із терміналу (можна скористатися інтернет пошуком для її визначення). Також команду можна визначати із аргументами (наприклад відкривати якийсь конкректний файл у текстовому редакторі або картинку або відео). Програма не працює із тими командами, що мають виконуватися від імені суперюзера (sudo). Таке обмеження продиктоване мірами безпеки.

Після закінчення редагування файлу конфігурацій треба перезапустити оболочку Gnome. Для цього треба вийти із системи і залогінитися знову.

Після логіну конфігурація додатку буде прочитана і завантажена.

У разі зміни конфігурацій, вони почнуть діяти лише після вопторного виходу і входу в систему. Файл конфігурації зчитується лише один раз підчас ініціалізації аддону.

Після цього для активації додатку треба натиснути на іконку “мікрофону”, що знаходиться в правому кутку панелі зверху. Кнопка на панелі зображена на риснку 1.

Після цього біля панелі з’явиться повідомлення про початок процесу розпізнавання голосу. Користувач може говорити. У користувача є 4 секунди на команду. Відображення процесу прослуховування зображений на рисунку 2.

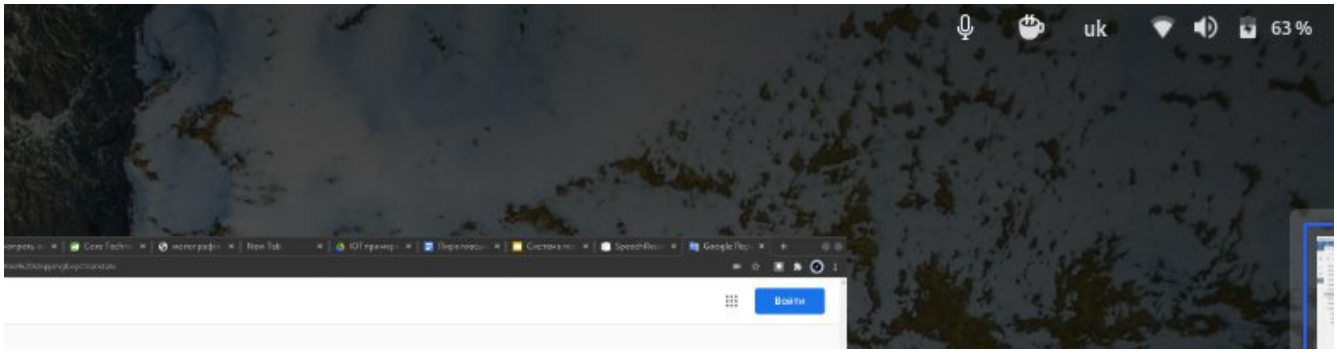

Рисунок 1. Кнопка активації помічника

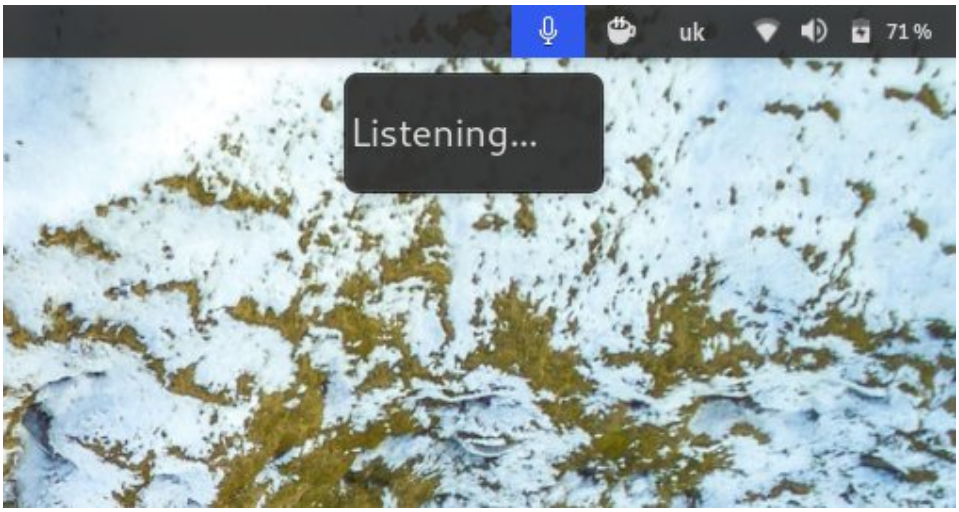

Рисунок 2. Помічник у стані прослуховування

Після цього розширення починає процес позпізнавання тексту. Визначає сказану фразу, шукає потрібну фразу у конфігурації і у випадку, якщо такий триггер був знайдений у конфігурації - запускає відповідну команду. На рисунку 3 зображено повідоблення помічника, коли він розпізнав введений користувачем текст. У даному випадку користувач сказав назву браузеру, що хоче запустити.

І після цього закінчує свою роботу і чекає до нового запиту від користувача.

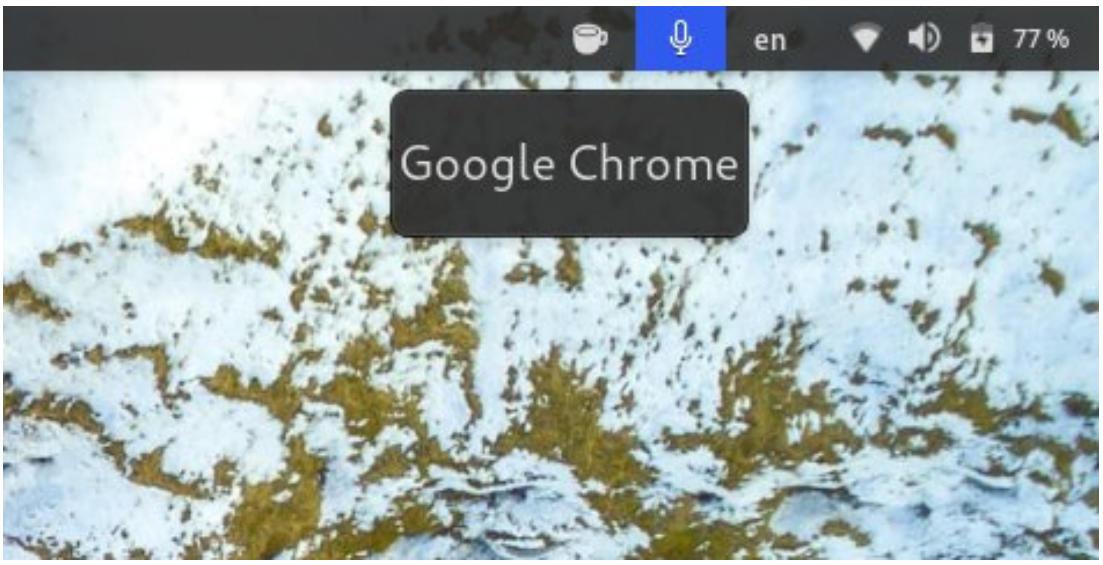

Рисунок 3. Помічник розпізнав текст

У разі, якщо система не змогла розпізнати фразу, сказану користувачем, або даної фрази не було знайдено в конфігурації - додаток попросить про повтор команди і знову спробує розпізнати і знайти команду. Рисунок 4 зображує повторне запрошення помічника до введення голосової команди.

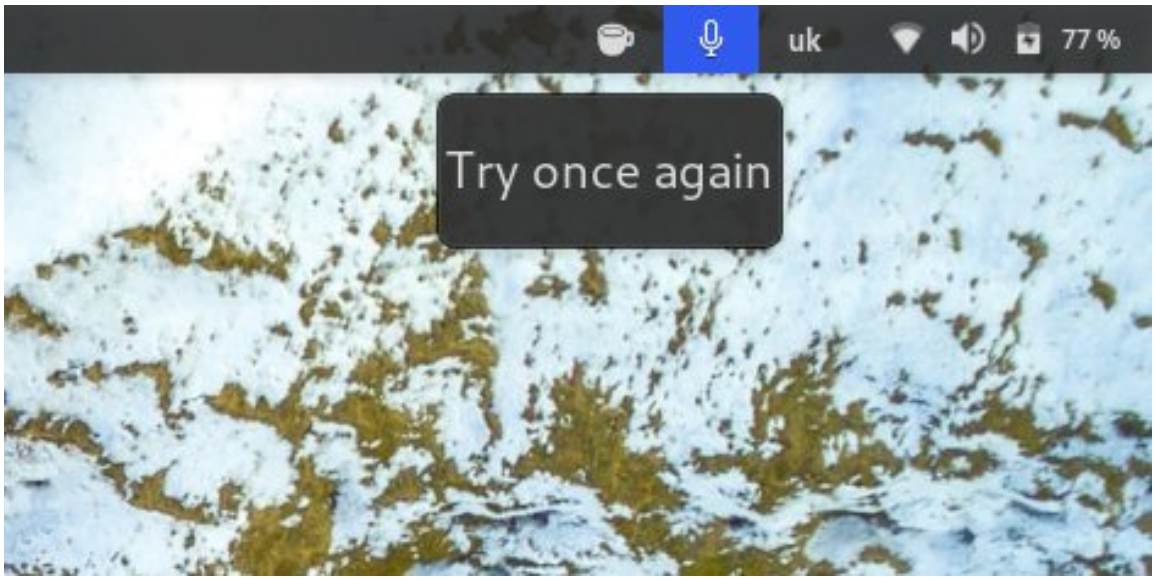

Рисунок 4. Запрошення до повторного введення команди

У разі, якщо користувач передумав - він може натиснути на будь яке місце на робочому столі для того, аби закрити процес прослуховування.

7. Опис життєвого циклу розширення для Gnome Desktop Environmen

Інтерфейс та розширення оболонки GNOME написані на GJS, що є прив'язками JavaScript для API GNOME C. Сюди входять такі бібліотеки, як Gtk, GLib / Gio, Clutter, GStreamer та багато інших. Подібно до того, як PyGObject є прив'язками Python для тих самих бібліотек.

JavaScript - це мова, заснована на прототипі, що для нас означає, що ми можемо змінювати інтерфейс та поведінку GNOME Shell під час її роботи. Це те, що відоме як "виправлення мавп". Наприклад, ви можете замінити функцію addMenuItem () класу PopupMenu, і всі існуючі або новостворені класи та підкласи PopupMenu негайно почнуть використовувати вашу заміну.

Розширення оболонки GNOME - це фактично виправлення, які застосовуються до оболонки GNOME, коли їх увімкнено, і повертаються назад, коли їх вимкнено. Насправді, єдина реальна різниця між виправленнями, об’єднаними в оболонку GNOME, і розширенням полягає у застосуванні виправлення.

8. Опис модулів розробленого додатку для запуску програм голосом

Розроблений додаток складається із двох модулей.

Перший модуль, що написаний на мові програмування JavaScript відповідає за роботу програми у якості розширення Gnome Desktop Environment.

Він відповідає за створення кнопки на головній панелі задач робочого столу, взаємодію із користувачем, за читання файлу кофігурації та запуску програм із конфігутації.

Перший модуль використовує такі елементи Gnome JS API як:

- St (Shell Toolkit) - для створення графічниз елементів (таких як текстовий напис із статусом, кнопка тощо);
- panelMenu - для створення елементу (кнопки) на головній панелі робочого столу;
- Gio - для роботи із іконками;
- GLib - для виконання команд (запуск програм);
- mainloop - для менеджмену потоку виконання програми.

Другий модуль - написаний на мові програмування Python і відповідає за роботу із пристоєм аудіо введення (мікрофоном) та розпізнавання аудіо за допомогою Google Speech Recognition API.

Перший модуль використовує другий модуль.

9. Використані посилання

1. https://realpython.com/python-speech-recognition/
2. https://pypi.org/project/PyAudio/
3. https://wiki.gnome.org/Projects/GnomeShell/Extensions/StepByStepTutorial
4. https://wiki.gnome.org/Projects/GnomeShell/Extensions/Writing
5. https://gjs-docs.gnome.org/shell01~0.1_api/

10. Вихідний код розробленого рішення

extention.js

```
/* St = Shell Toolkit */
const St = imports.gi.St;
const PanelMenu = imports.ui.panelMenu;
const Gio = imports.gi.Gio;
const Main = imports.ui.main;
const GLib = imports.gi.GLib;
const Mainloop = imports.mainloop;

let dndButton;
let globalStillListeningFlag;
let configMap = {}

const Me = imports.misc.extensionUtils.getCurrentExtension();
imports.searchPath.unshift(Me.path);

function init() {
}

function thisPath(name) {
    return Me.path + "/" + name;
}


function enable() {
    readConfigFile();
```

```
        dndButton = new PanelMenu.Button(1, "DoNotDisturb",
false);

        let box = new St.BoxLayout();
        let icon = new St.Icon({
            style_class: "system-status-icon",
        });

icon.set_gicon(Gio.icon_new_for_string(thisPath("mic.svg")));
        box.add(icon);

        dndButton.actor.add_child(box);
        Main.panel.addToStatusArea("DoNotDisturbRole", dndButton,
0, "right");

        // Menu
        let dndMenu = dndButton.menu;
        const snLabel = new St.Label({
            text: 'text',
            style: "margin: 10px; font-size: large",
        });

        // PopupMenu expects all children have _delegate
        snLabel._delegate = null;
        dndMenu.box.add(snLabel);
        dndMenu.connect("open-state-changed", function (menu,
isOpen) {
                globalStillListeningFlag = isOpen;
                if (isOpen) {
                    snLabel.set_text("Listening...");
                    myLoopFunction(snLabel);
                }
```

```
            }
        );
    }

    function disable() {
        dndButton.destroy();
    }


    function readConfigFile() {
        let file = Gio.file_new_for_path(thisPath("main.config"));
        file.load_contents_async(null, function (file, res) {
            let contents;
            try {
                contents                                                =
file.load_contents_finish(res)[1].toString();
                let   commands   =   contents.split('\n').map(el   =>
el.trim()).filter(el => el.length > 0 && el.indexOf(':') > -1);
                commands.forEach(command => {
                        let a = command.split(':');
                        let a1 = a[0].trim();
                        let a2 = a[1].trim();
                        if (a1 && a2) {
                            configMap[a1.toLowerCase()] = a2;
                        }
                    }
                )

            } catch (error) {
                print(error)
            }
        });
```

```
    }

    function listen(snLabel) {
        let [res, out] = GLib.spawn_command_line_sync("python2.7 "
+ thisPath("main.py"));
        snLabel.set_text(out.toString());
        let s = out.toString().toLowerCase().trim();
        let configMapElement = configMap[s];
        if (globalStillListeningFlag) {
            if (configMapElement) {
                GLib.spawn_command_line_async(configMapElement);
            } else {
                snLabel.set_text("Try once again");
                myLoopFunction(snLabel);
            }
        }
    }

    let myLoopFunction = (snLabel) => {
        Mainloop.timeout_add(500, () => {
                snLabel.set_text("Listening...");
                if (globalStillListeningFlag) {
                    Mainloop.timeout_add(500,         ()         =>
listen(snLabel));
                }
            }
        );
    }
```

main.py

```
# coding: utf-8
import speech_recognition as sr

r = sr.Recognizer()
with sr.Microphone() as source:
    audio = r.listen(source, 2, 4)

try:
    print(r.recognize_google(audio))
except sr.UnknownValueError:
    print("sorry. could not understand")
except sr.RequestError as e:
    print("Error; {0}".format(e))
```

main.config example

```
open chrome : google-chrome
chrome : google-chrome
google-chrome : google-chrome
google chrome : google-chrome
google home : google-chrome
browser : google-chrome
internet : google-chrome
sublime text : subl
sublime : subl
text : subl
terminal : gnome-terminal
command line : gnome-terminal
show movie : vlc
vlc : vlc
```

```
videos : vlc

notes : subl ~/notes
my notes : subl ~/notes
open notes : subl ~/notes
```