

Dooku – Asystent głosowy



Autor projektu:
Franciszek Majka

1. Abstrakt

Dooku to asystent głosowy zdolny do rozpoznawania konkretnych słów kluczowych i wywoływania zdefiniowanych komend.

Korzystając z Dooku można uruchamiać programy, modyfikować zasady nasłuchiwania, symulować naciśnięcia klawiszy i więcej...

Komendy i ich działanie można łatwo zdefiniować w pliku konfiguracyjnym, a ustawienia oferują sporą elastyczność.

Dooku korzysta z otwartoźródłowej **biblioteki PocketSphinx** w celu ciągłego rozpoznawania mowy.

Do wygenerowania wykrywalnych przez asystenta słów wykorzystywana jest funkcjonalność **programu Norman**, będącego nowszą alternatywą dla programu Blather.

Jeden z modułów Normana listuje słowa składowe komend i na ich bazie generuje plik słownikowy i plik modelu językowego. Utworzone pliki są przekazywane jako argumenty do programu rozpoznawania mowy, dlatego Dooku jest w stanie rozpoznać słowa składowe komend.

2. Wstęp

2.1. Cel

Celem projektu było stworzenie **asystenta głosowego** zdolnego do rozpoznawania szeregu komend. **Asystent Dooku** miał stanowić pomocne narzędzie ułatwiające poruszanie się po systemie, przyspieszając i zwiększając wygodę korzystania z komputera.

Kolejną istotną możliwością mogłaby być integracja z zewnętrznymi autorskimi programami.

2.2. Zakres

Początkowym zakresem pracy była instalacja i konfiguracja programu Blather.

Z powodu problemów z instalacją przestarzałego już programu Blather, zakres został przeniesiony na podobną instalację i konfigurację programu Norman.

Normana udało się uruchomić na maszynie wirtualnej z zainstalowanym systemem Linux. Następnie została podjęta (bezskuteczna) próba odtworzenia pracy na platformie Windows. Aby program mógł zostać uruchomiony na systemie Windows, Normana i moduły wchodzące w jego skład trzeba było zainstalować na WSL (Windows Subsystem for Linux), udostępniającym zintegrowane środowisko Linux dla Windows.

Z powodu ograniczeń WSL związanych z kartą dźwiękową program Norman jednak nie mógł działać w całości poprawnie.

Ostateczny zakres pracy objął:

- Zdefiniowanie konstrukcji pliku konfiguracyjnego
- Obsługa konwersji komend na pliki słownikowe przy użyciu modułu Normana
- Przekazanie odpowiednich argumentów do programu `pocketsphinx_continuous`
- Napisanie skryptu w Pythonie przetwarzającego rozpoznany tekst i wywołującego zdefiniowane polecenia
- Automatyzacja całego procesu poprzez skrypty w Bashu i Pythonie

2.3. Metodyka

Wykorzystane narzędzia oraz cel wykorzystania:

- **WSL** (Windows Subsystem for Linux) – uruchomienie programu na systemie Windows
- **Moduł `convert-commands.js` programu Norman** - generowanie plików słownikowych
- **Program `pocketsphinx_continuous`** - rozpoznawanie mowy i output w postaci tekstu
- **Skrypty w języku Python** - przetwarzanie otrzymanego tekstu i wykonywanie komend

- **Skrypty w języku Bash** - przekazywanie rozpoznanego tekstu do skryptu przetwarzającego tekst, automatyzacja konwersji, konfiguracji i uruchamiania programu.

3. Część teoretyczna

3.1. Podobieństwo cosinusowe

Aby polepszyć możliwości rozpoznawania komend, w trakcie ich przetwarzania przez program liczone jest podobieństwo cosinusowe między otrzymaną komendą a wszystkimi zdefiniowanymi komendami zdefiniowanymi w pliku konfiguracyjnym. Dzięki temu w większości przypadków nawet jeśli nie zostanie wypowiedziana dokładna komenda, program będzie w stanie "domyślić się" o którą komendę chodziło i ją wywołać.

W analizie danych podobieństwo cosinusowe jest miarą podobieństwa między dwoma wektorami. Podobieństwo cosinusowe to cosinus kąta między dwoma wektorami, więc jest to iloczyn skalarny dwóch wektorów podzielony przez iloczyn ich długości.ⁱ

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

Wzór na podobieństwo cosinusoweⁱⁱ

3.2. Potoki

Do przekazywania mowy rozpoznanej przez program *pocketsphinx_continuous* do skryptu przetwarzającego komendy wykorzystywany jest mechanizm zwany potokiem

Potok (ang. pipe) – jeden z mechanizmów komunikacji międzyprocesowej umożliwiający wymianę danych pomiędzy dwoma procesami. Odbywa się to najczęściej poprzez połączenie standardowego wyjścia jednego procesu ze standardowym wejściem drugiego.ⁱⁱⁱ

4. Część praktyczna

4.1. Elementy składowe

Na cały program składa się kilka plików i programów:

- **conf.py** – plik konfiguracyjny; użytkownik jako klucze słownika wpisuje komendy, które asystent ma rozumieć, a przypisane im wartości to predefiniowane słowa kluczowe lub lista argumentów
- **convert.py** – czyta zawartość pliku *conf.py*, następnie zapisuje wszystkie komendy w pliku *conf.js* jako obiekt JavaScript
- **conf.js** – plik zawierający wszystkie wyrazy wchodzące na skład komend
- **convert-commands.js** – jeden z modułów programu Norman, konwertuje zawartość pliku *conf.js* na pliki słownikowe wykorzystywane przez program do rozpoznawania mowy *pocketsphinx_continuous*
- **convert.sh** – automatyzuje proces konwersji; wykonuje skrypt *convert.py*, kopiuje wynikowy plik *conf.js* do folderu Normana, po czym uruchamia skrypt *convert-commands.js*, w wyniku czego powstają pliki potrzebne do zrozumienia komend przez program rozpoznający mowę
- **pocketsphinx_continuous** – część otwartoźródłowego silnika do rozpoznawania mowy PocketSphinx. Program *pocketsphinx_continuous* może nasłuchiwać dźwięku z mikrofonu; w argumentach wywołania programu można podać zestaw możliwych do rozpoznania słów; bazując na słowniku program wypisuje wykryte słowa
- **dooku.py** – skrypt odczytujący komendy ze standardowego wejścia, przetwarzający ich treść i wywołujący przypisane im funkcje
- **run.sh** – skrypt bashowy automatyzujący włączanie asystenta; uruchamia program *pocketsphinx_continuous* wraz z niezbędnymi argumentami i poprzez potok przekazuje rozpoznane przez niego komendy na wejście skryptu *dooku.py*

4.2. Plik konfiguracyjny

Głównym elementem pliku konfiguracyjnego **conf.py** jest słownik mapujący wszystkie zrozumiałe przez asystenta komendy do akcji z tą komendą powiązanych. Akcję może określać słowo kluczowe lub lista.

Wszystkie dozwolone wartości wraz z definicją działania:

- **keyword** – wyraz wywołujący asystenta, musi być pierwszym słowem w wypowiedzi aby program przetworzył wartość wejściową
- **exit** – komenda wyłączająca program
- **keyboard** – komenda przekazywana do zewnętrznego skryptu w Pythonie wywołującego naciśnięcia klawiszy przy pomocy biblioteki *keyboard*

- **number** – słowo reprezentujące liczbę, może być przekazane jako argument przy komendzie typu *keyboard* poprzez znak #, np. komenda *"volume #"* oznacza, że drugim wyrazem w komendzie będzie dowolna zdefiniowana w ten sposób liczba
- **continuous_enable** – komenda włączająca tryb ciągłego nasłuchiwania, tj. nie trzeba wypowiadać słowa kluczowego przed powiedzeniem komendy
- **continuous_disable** – komenda wyłączająca tryb ciągłego nasłuchiwania
- **chain** – słowo kluczowe rozdzielające komendy; wykorzystanie go umożliwia przekazanie asystentowi do wykonania kilku komend jedna po drugiej
- **[]** – w przypadku podania listy po wypowiedzeniu komendy poprzez funkcję *execvp()* zostanie wywołany program będący pierwszym elementem listy, wraz z kolejnymi elementami jako argumentami przy wywoływaniu programu, np. *["notepad.exe", "C:/dooku.txt"]* otworzy w notatniku plik o nazwie *dooku.txt* na dysku C użytkownika

4.3. Korzystanie z programu

Pierwszym krokiem jest edycja pliku **conf.py**, wpisanie tam swoich komend oraz działania jakie chcemy tymi komendami osiągnąć.

```

0  Insert your commands
1  commands = {
2      "DOOKU": "keyword",
3      "GOODBYE": "exit",
4      "THEN": "chain",
5      "ONE": "number",
6      "TWO": "number",
7      "COPY": "keyboard",
8      "PASTE": "keyboard",
9      "VOLUME UP #": "keyboard",
10     "VOLUME DOWN #": "keyboard",
11     "LISTEN": "continuous_enable",
12     "OVER": "continuous_disable",
13     "OPEN CHROME": ["/mnt/c/Program Files/Google/Chrome/Application/chrome.exe"],
14     "OPEN NOTEPAD": ["/mnt/c/Program Files/Notepad++/notepad++.exe"],
15 }

```

Przykładowa zawartość pliku **conf.js**

Następnie należy wykonać skrypt **convert.sh**, w wyniku czego otrzymamy pliki słownikowe niezbędne do zrozumienia komend przez program rozpoznający mowę.

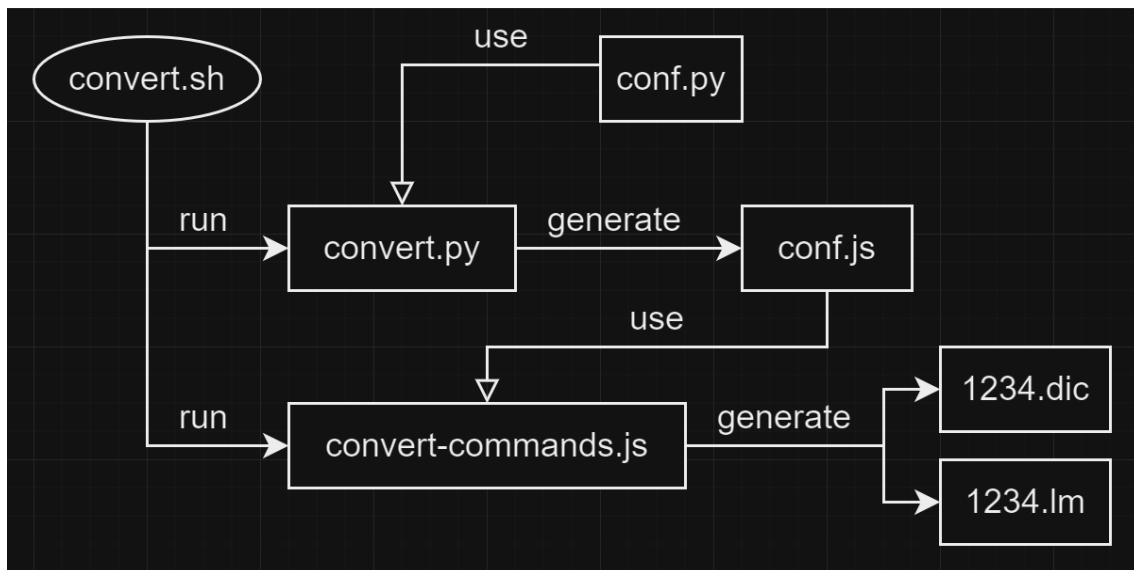


Diagram pokazujący etap konwersji poprzez skrypt **convert.sh**

Teraz można już włączyć samego **asystenta Dooku** poprzez wykonanie skryptu **run.sh**

Skrypt uruchamia program **pocketsphinx_continuous**, przekazując w argumentach wygenerowane w procesie konwersji pliki słownikowe. Przekazuje też output programu do skryptu **dooku.py**, który bazując na zawartości pliku konfiguracyjnego **conf.py** analizuje otrzymywany na wejściu tekst i parsuje komendy.

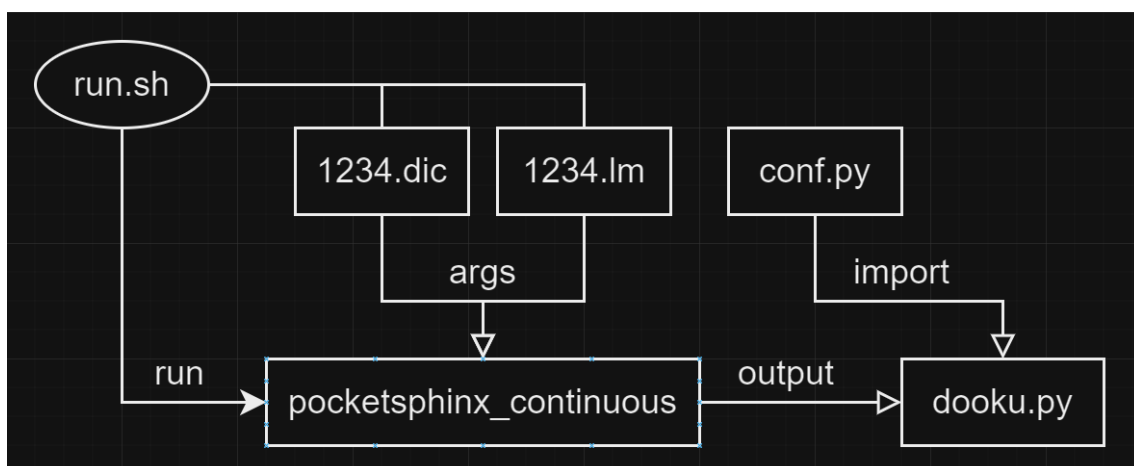


Diagram opisujący proces działania programu

4.4. Skrypt przetwarzający komendy

Za to, aby wypowiedzenie jakiejś komendy rzeczywiście odniosło jakikolwiek efekt odpowiedzialny jest skrypt **dooku.py**.

Skrypt składa się z głównej pętli czytającej tekst ze standardowego wejścia, następnie zestaw funkcji przetwarza ten tekst w celu rozpoznania w nim komend zdefiniowanych w pliku konfiguracyjnym.

Lista funkcji wchodzących w skład skryptu:

- **check_keyword** – sprawdza czy pierwszy wyraz w otrzymanej treści jest słowem kluczowym; jeśli nie, komenda jest ignorowana
- **split_cmd** – przeszukuje treść w poszukiwaniu słów rozdzielających komendy (klucze o wartości "chain" w pliku konfiguracyjnym); w przypadku znalezienia dzieli komendę; ostatecznie zwraca listę jednej lub więcej komend
- **process_cmd()** – funkcja przetwarzająca komendę, wywołuje zbiór podrzędnych funkcji; przetwarza osobno każdą komendę z listy zwróconej przez funkcję *split_cmd*
- **find_cmd()** – porównuje otrzymaną komendę do listy zdefiniowanych komend; proces obejmuje wywołanie dwóch podrzędnych funkcji: *strip_cmd* oraz *cos_sim*
- **strip_cmd()** – funkcja podmienia wartości liczbowe (klucze o wartości "number" w pliku konfiguracyjnym) na znak "#" aby można ją było porównać otrzymaną komendę z tymi zdefiniowanymi w konfiguracji; oprócz zmienionej komendy zwraca również listę znalezionych wartości liczbowych
- **cos_sim** – liczy podobieństwo cosinusowe między otrzymaną komendą a każdą zdefiniowaną komendą; zwraca tę najbardziej podobną do otrzymanej, chyba że wystąpiła więcej niż jedna – w owym przypadku komenda jest ignorowana
- **insert_cmd_args** – po znalezieniu komendy wartości liczbowe są z powrotem wstawiane w miejsce znaków "#"
- **run_cmd** – bazując na treści komendy wywołuje wskazany program z argumentami

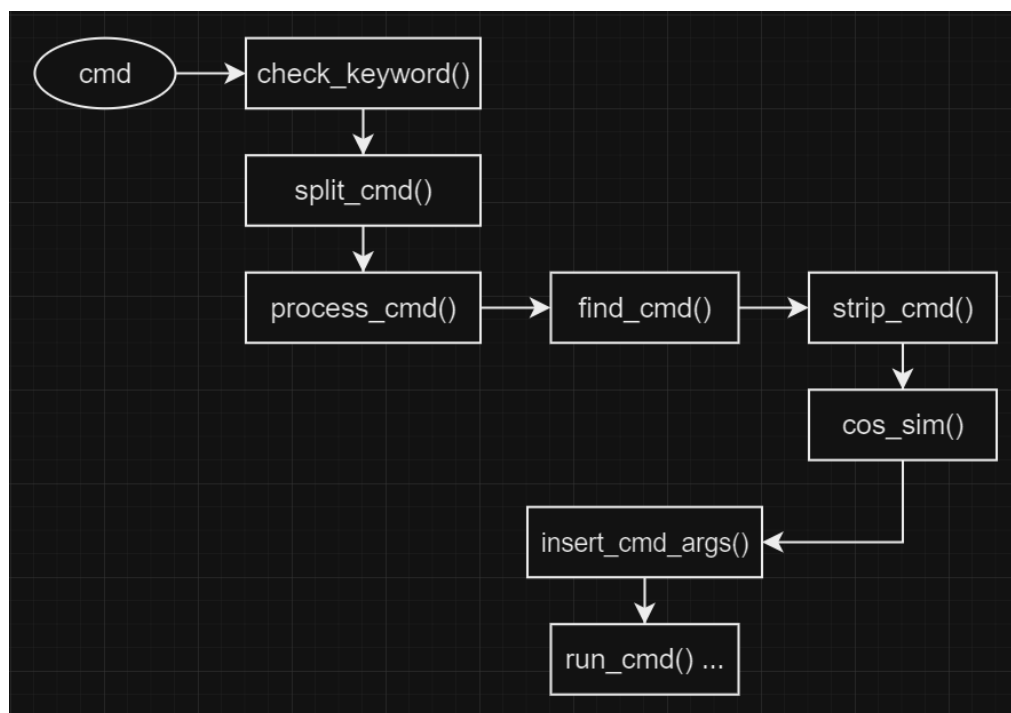


Diagram prezentujący kolejność wywoływania funkcji w skrypcie

5. Podsumowanie

W rezultacie otrzymano asystenta głosowego który względnie dobrze radzi sobie z rozpoznawaniem komend.

Napotkane problemy:

Narzędzie do rozpoznawania mowy nie radzi sobie zbyt dobrze z podobnie brzmiącymi wyrazami, np. słowo *DOOKU* nierzadko było mylnie rozpoznawane jako *TWO* lub *NEW*. Przypadkowe hałasy niebędące głosem człowieka również były błędnie rozpoznawane jako wyrazy, np. odgłos kropli wody spadającej na parapet został rozpoznany jako *CUT*. Te problemy powodują że korzystanie z asystenta w hałaśliwym otoczeniu jest raczej niemożliwe.

Dostosowanie czułości programu *pocketsphinx_continuous* poprzez podanie odpowiednich argumentów przy uruchomieniu może załagodzić powyższe problemy, nie zostało to jednak dogłębnie sprawdzone.

```
COMMAND: DOOKU PASTE
FOUND: PASTE (sim=1.00)
['PASTE']
COMMAND:
COMMAND:
COMMAND:
COMMAND:
COMMAND:
COMMAND:
COMMAND:
COMMAND: THREE
COMMAND: NEW OPEN CHROME
COMMAND: DOOKU OPEN CHROME
FOUND: OPEN CHROME (sim=1.00)
```

Fragment informacji zwrotnej z działania programu

Z powyższego zrzutu ekranu wynika, że program do rozpoznawania mowy często reaguje przy drobnych hałasach, ostatecznie nie wykrywając żadnego dźwięku lub rozumiejąc hałas jako przypadkowe słowo (*THREE*). Jeśli nie mówi się bardzo wyraźnie lub jest się daleko od mikrofonu, podobnie brzmiące słowa mogą być źle rozpoznane (*DOOKU* – *NEW*)

6. Bibliografia

ⁱ Źródło: https://en.wikipedia.org/wiki/Cosine_similarity

ⁱⁱ Grafika pozyskana ze strony: <https://builtin.com/machine-learning/cosine-similarity>

ⁱⁱⁱ Źródło: [https://pl.wikipedia.org/wiki/Potok_\(Unix\)](https://pl.wikipedia.org/wiki/Potok_(Unix))