Syntezator robotycznego głosu

Autorzy: Jakub Adamczyk, Mikołaj Tomalik

Spis treści

1 Opis projektu

Projekt miał na celu stworzenie oprogramowania do urządzenia Raspberry Pi w języku Python mającego przeprowadzać zamianę ludzkiego głosu na robotyczny w czasie rzeczywistym.

Projekt składa się z programu (zestawu skryptów w Pythonie) oraz odpowiednio skonfigurowanego systemu odpowiedniego dla Raspberry Pi. W przypadku wykorzystywania już skonfigurowanego urządzenia należy z niego korzystać zgodnie z opisem zawartym w rozdziale "Dokumentacja użytkowa". W przypadku konieczności samodzielnej instalacji oprogramowania należy podążać zgodnie ze wskazówkami z rozdziału "Dokumentacja techniczna", podrozdział "Konfiguracja istniejącego programu".

Główną ideą było wykorzystanie programowej symulacji urządzenia *ring modulator*, znanego m. in. z charakterystycznych głosów Daleków z oryginalnej serii "Doktora Who". Wzorowaliśmy się tutaj na implementacji w języku JavaScript wykonanej przez BBC, translacji do Pythona tego programu autorstwa Neila Lakina oraz artykułu opisującego podstawy teoretyczne oraz matematykę stojącą za takim urządzeniem (patrz pozycje 1, 2 i 3 bibliografii).

Udało się zrealizować wszystkie główne założenia projektu:

- stworzenie programu w języku Python działającego z odpowiednią wydajnością na Raspberry Pi
- uzyskanie efektu robotycznego głosu
- zachowanie odpowiedniej jakości dźwięku zarówno pod względem wejścia (filtracja szumu) oraz wyjścia (efekty dźwiękowe niepogarszające znacząco jakości)

2 Dokumentacja użytkowa

Posiadając Raspberry Pi ze skonfigurowanym oprogramowaniem należy je tylko podłączyć do prądu oraz podłączyć słuchawki i mikrofon. Urządzenie potrzebuje kilkanaście sekund na włączenie się, następnie można już mówić i słyszeć w odpowiedzi robotyczny głos. Jest to działanie zasadniczo na zasadzie plug-and-play.

3 Dokumentacja techniczna

3.1 Instalacja oprogramowania

Program wymaga zainstalowania na urządzeniu Raspberry Pi (poprzez wpisanie następujących komend w terminalu):

- najnowszej wersji wybranego systemu operacyjnego:

```
sudo apt-get update
sudo apt-get upgrade
```

- interpretera języka Python w wersji 3.X wraz z instalatorem pakietów pip:

```
sudo apt-get install python3-dev python3-pip
```

- wersji deweloperskiej biblioteki libasound do komunikacji ze sterownikami karty dźwiękowej ALSA:

```
sudo apt-get install libasound-dev
```

- wersji deweloperskiej biblioteki portaudio do komunikacji między programami a biblioteką libasound i sterownikami ALSA:

```
sudo apt-get install portaudio19-dev
```

- biblioteki PyAudio do komunikacji między sprzętem dźwiękowym a Pythonem:

```
pip install pyaudio --user
sudo apt-get install python3-pyaudio
```

- bibliotek numpy i scipy do obsługi i przetwarzania danych audio w Pythonie:

```
pip install numpy
sudo apt-get install python3-scipy
pip install scipy
```

3.2 Konfiguracja urządzeń audio input/output

Dla prawidłowego działania mikrofonu i słuchawek w Pythonie na Raspberry Pi trzeba je ręcznie skonfigurować. Urządzenia te system identifykuje jako pary (numer karty, numer urządzenia). Aby sprawdzić, jakie numery posiadają nasze pożądane urządzenia należy użyć komend:

```
arecord -l
aplay -l
```

Listing 1: uzyskiwanie numerów kart i urządzeń (arecord - input, aplay - output)

Wejście jack 3.5 mm (output, słuchawki) ma zwykle opis `Analog` lub `bcm2835 ALSA` (nie `bcm2835 IEC958/HDMI`).

Należy następnie utworzyć plik `.asoundrc` w katalogu domowym (domyślnie `/home/pi`). Ma on treść podaną w listingu 2 poniżej (numery kart i urządzeń trzeba uzupełnić danymi uzyskanymi za pomocą kodu z listingu 1).

```
pcm.!default {
  type asym
  capture.pcm "mic"
  playback.pcm "speaker"
}
pcm.mic {
  type plug
  slave {
    pcm "hw:<card number>,<device number>"
  }
}
pcm.speaker {
  type plug
  slave {
    pcm "hw:<card number>,<device number>"
  }
}
```

Listing 2: Treść pliku .asoundrc

3.3 Konfiguracja uruchomienia przy starcie

Aby program uruchamiał się przy starcie systemu, zdefiniowano go jako usługę (service) systemu Raspbian (zadziała także na dowolnym podobnym systemie Unixowym).
W tym celu dodano plik `/etc/systemd/system/audio.service` o treści podanej w listingu 3. Zakłada on, że plik `main.py` znajduje się w katalogu domowym użytkownika pi - jeżeli jest inaczej, należy odpowiednio zmienić drugą ścieżkę dla zmiennej `ExecStart`.

```
[Unit]
Description=Audio

[Service]
ExecStart=/usr/bin/python3 /home/pi/main.py
WorkingDirectory=/home/pi
StandardOutput=inherit
StandardError=inherit
Restart=always
User=pi

[Install]
WantedBy=multi-user.target
```

Listing 3: Treść pliku audio.service

3.4 Parametry wejściowe programu

Program wykorzystuje parametry zapisane jako zmienne globalne w pliku `parameters.py`.

Parametry `PA_TYPE` oraz `NP_TYPE` odpowiadają typom danych używanym w bibliotekach PyAudio oraz numpy do reprezentacji danych audio w tablicach. Ze względów kompatybilności tych bibliotek muszą być one analogiczne dla prawidłowego działania programu, np. `PA_TYPE=pyaudio.paInt32` oraz `NP_TYPE=numpy.int32`.

Doświadczalnie sprawdzono, iż typ zmiennoprzecinkowy z niewiadomych powodów nie działa, natomiast większa liczba bitów dla liczb całkowitych zwiększa jakość dźwięku (np. `numpy.int32` zamiast `numpy.int16`).

Przez ramkę (frame) rozumie się fragment dźwięku w czasie, tj. wartość jego amplitudy. Tablice przekazywane w programach to zbiory ramek o określonych wartościach.

Parametr `RATE` (sampling rate) wyznacza liczbę ramek na sekundę dla urządzenia wejściowego. Jego wartość domyślna to sampling rate dla domyślnego dla danego systemu urządzenia wejściowego. Nieprawidłowy dla danego urządzenia sampling rate może wiązać się z niższą jakością dźwięku lub całkowitym jego zniekształceniem.

Parametr `CHUNK` to długość pojedynczego zbioru ramek. Program nagrywa tyle ramek na każdym kanale, a następnie przekazuje je do przetwarzania funkcji callback. Większy chunk oznacza krótsze ramki oraz potencjalnie większą jakość nagrywania, jednak nakłada to znaczne obciążenie na sprzęt. Wartość domyślna 1024 jest typowa dla większości standardowych urządzeń.

Parametr `CHANNELS` to liczba nagrywanych i/lub odtwarzanych kanałów. Wynosi zwykle 1 (dźwięk mono) lub 2 (dźwięk stereo). Jeżeli istnieje taka możliwość, to dla wyższej jakości dźwięku należy wybrać większą liczbę. Dane z obu kanałów są przeplatane (interleaved), a więc są to na przemian ramki lewego i prawego kanału. W razie potrzeby można je zgodnie z tą zasadą rozdzielić.

Ważne, iż w praktyce operuje się na liczbie ramek równej `CHANNELS * CHUNK`, gdyż każdy kanał nagrywa się osobno.

3.5 Efekty przetwarzania (synteza)

Główną funkcją programu jest zamiana nagrywanego dźwięku na robotyczny. Realizuje się to poprzez symulację urządzenia *ring modulator* (patrz punkty 1 i 3 bibliografii) z użyciem funkcji `module_audio` (moduł `modulator`) oraz wywoływanych przez nią funkcji pomocniczych. Zaletą tego podejścia jest wykorzystanie czysto Pythonowego kodu oraz brak konieczności obciążającego sprzętowo FFT do samej syntezy dźwięku. Jeżeli chciałoby się dodatkowo wykorzystać inne efekty (a większość z nich wymaga użycia FFT), to trzeba także pamiętać, że operacja odwrotna IFFT zniekształca nieco dźwięk na krańcach przedziału, a więc na granicach każdego chunka, co znacząco obniża jakość dźwięku (sprawdziliśmy to doświadczalnie).

3.6 Playback

W pętli nieskończonej program nagrywa dźwięk. Po nagraniu całego chunku wywołuje funkcję `callback`, która w argumencie `data` otrzymuje nagrany dźwięk w formacie wskazanym w `PA_TYPE`, ale w postaci strumienia bajtowego. Wykorzystanie funkcji biblioteki numpy `frombuffer` i przetworzenie tych danych na postać tablicy typu `NP_TYPE` jest niezbędne dla dalszego przetwarzania tych danych.

Dalsza część funkcji to filtrowanie i przetwarzanie danych audio, wedle uznania programisty. Na koniec zwraca się przetworzone dane jako krotkę tablicy numpy z danymi oraz stałą PyAudio `pyaudio.paContinue` (oznacza ona kontynuację nagrywania).

3.7 Filtrowanie dźwięku

Największy wpływ na jakość dźwięku na wpływ otoczenia. Charakterystyczną cechą szumu jest niska (w porównaniu do głosu przy mikrofonie) amplituda, czego da się pozbyć filtrem cyfrowym IIR. Służy do tego funkcja `filter_audio`, która tworzy filtr za pomocą funkcji biblioteki numpy `iirdesign`. Na drodze doświadczalnej wybrano filtr eliptyczny jako domyślny. Filtrowanie to wykonywane jest ze względu na wydajność (wymaga użycia FFT) tylko jednokrotnie, na wejściu audio. Zastosowano także prosty filtr amplitudowy, czyli zerowanie dźwięku o niskiej amplitudzie - potencjalnego szumu.

4 Dokumentacja procesu

Proces tworzenia programu był podzielony na dwie równolegle realizowane części: sprzętową realizację na Raspberry Pi oraz napisanie oprogramowania.

4.1 Proces tworzenia oprogramowania

Pierwszym wyzwaniem było nagrywanie i odtwarzanie dźwięku w czasie rzeczywistym. Okazało się, że pomimo popularności problemu odtwarzania dźwięku w Pythonie zdecydowana większość bibliotek obsługuje tylko obsługę plików .wav z dysku lub też odtwarzanie lub nagrywanie blokujące. Jedyną biblioteką, którą udało nam się znaleźć pozwalającą na audio playback była stosunkowo niskopoziomowa biblioteka PyAudio, stanowiąca de facto bindingi dla Pythona do biblioteki PortAudio - wieloplatformowej biblioteki w C zapewniającej warstwę między programami a sterownikami karty dźwiękowej.

Jak się okazało, funkcja callback wykorzystywana przez bibliotekę PyAudio zwracała strumień bajtowy (pomimo przyjmowania tablic biblioteki numpy jako argumentów), co wymagało konwersji. W tym momencie okazało się, że liczby typu float są w wielu przypadkach niepoprawnie obsługiwane i musieliśmy ograniczyć się do liczb typu int. Okazało się to później bardzo korzystną decyzją, gdyż pozwoliło sprawdzić charakterystyki szumu i łatwo użyć filtru amplitudowego.

Samo odfiltrowywanie szumu wymagało zapoznania się z podstawami przetwarzania sygnałów cyfrowych i rodzajami filtrów. Wybraliśmy filtr IIR jako popularny i prosty w użyciu z pomocą biblioteki numpy. Użyliśmy filtra lowpass jako dającego najlepsze efekty.

Największym wyzwaniem była sama synteza robotycznego głosu. Okazało się, że problem syntezy głosu jest bardzo rzadki, a istniejące rozwiązania są trudne w użyciu. Próbowaliśmy dostosować do naszego celu liczne biblioteki, jednak tylko biblioteka pysndfx ostatecznie zadziałała. Znacząco pogarszała jednak jakość dźwięku na wyjściu.

Ostatecznie użyliśmy techniki *ring modulation* dzięki przykładom na stronie BBC (implementacja w języku JavaScript), znalezionej w internecie translacji do języka Python oraz artykułowi naukowemu, wykorzystanemu także przez BBC, wyjaśniającego podstawy teoretyczne tej operacji. Ostateczny kod wykorzystuje do samej syntezy dźwięku tylko język Python i stanowi ulepszoną wersję kodu Neila Lakina (patrz punkt 3 dokumentacji). Usunęliśmy niepotrzebne części kodu, zoptymalizowaliśmy go i zdebugowaliśmy (m. in. oryginalny kod nie obsługiwał sytuacji dzielenia przez 0 podczas ciszy w nagrywaniu).

Ze względu na duże obciążenie sprzętowe operacji FFT musieliśmy zrezygnować z podwójnej filtracji dźwięku (na wejściu oraz po przetworzeniu) i ograniczyć się tylko do filtracji wejścia. Ostateczna wersja programu mimo tego jednak zapewnia wysoką jakość dźwięku na wyjściu.

4.2 Proces realizacji sprzętowej

Głównym wyzwaniem przy uruchomieniu programu na Raspberry Pi z systemem operacyjnym Raspian okazało się uruchomienie sterowników ALSA. Jak się okazało, nie współpracują one dobrze z jądrem tego systemu i przez to ciężko było wskazać w PyAudio, jakich urządzeń należy używać do nagrywania i odtwarzania dźwięku.

Do rozwiązania tego problemu zaadaptowaliśmy kod wykrywający podłączone urządzenia audio input/output i ustawiliśmy odpowiednie wartości domyślne (patrz punkt 7 bibliografii).

Skonfigurowaliśmy także odpowiednio samo Raspberry Pi, gdyż nie do końca radzi sobie z wyborem sprzętu przy używaniu go w języku programowania. W tym celu skorzystaliśmy z tutorialu Google'a (patrz punkt 6 bibliografii) i stworzyliśmy odpowiedni plik konfiguracyjny (patrz podrozdział 3.2 dokumentacji).

Na koniec, aby ułatwić obsługę urządzenia przez końcowego użytkownika, ustawiliśmy automatyczne działanie naszego programu przez cały czas działania urządzenia, jak opisano w podrozdziale 3.3. Dzięki temu działa ono na zasadzie plug-and-play.

5 Bibliografia

1. Implementacja urządzenia *ring modulator* w języku JavaScript przez BBC:
   `https://webaudio.prototyping.bbc.co.uk/ring-modulator/#code`
2. Oryginalny artykuł (wykorzystany też przez BBC) o podstawach teoretycznych działania urządzenia *ring modulator*:
   `http://recherche.ircam.fr/pub/dafx11/Papers/66_e.pdf`
3. Tłumaczenie kodu z języka JavaScript do języka Python:
   `https://github.com/nrlakin/robot_voice`
4. Dokumentacja biblioteki PyAudio:
   `https://people.csail.mit.edu/hubert/pyaudio/docs/`
5. Instalacja biblioteki PyAudio na Raspberry Pi:
   `https://raspberrypi.stackexchange.com/questions/75031/cannot-install-pyaudio`
6. Wykrywanie i konfiguracja urządzeń audio input/output w systemie operacyjnym:
   `https://developers.google.com/assistant/sdk/guides/library/python/embed/audio?hardware=rpi&fbclid=IwAR3kINsD0TweuAqXBFh1tFRcpIV9Vmox5jz29o_KXbnvg2onuh46xWZfp2I`
7. Wykrywanie urządzeń audio input/output w PyAudio:
   `https://gist.github.com/mansam/9332445`