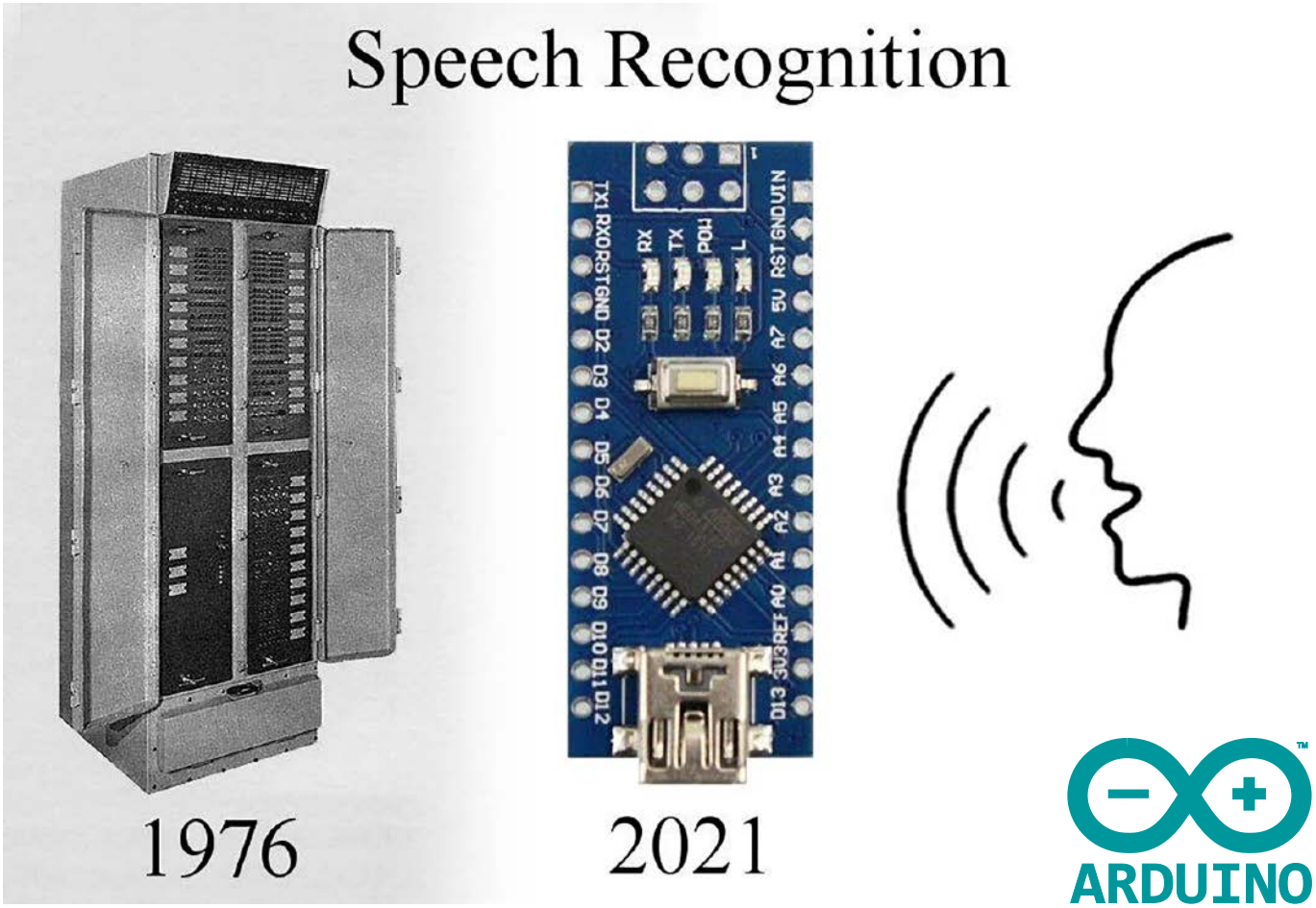


Rozpoznawanie głosu z użyciem Arduino

Kiedyś do rozpoznawania głosu konieczne były ogromne zasoby sprzętowe, niedawno można było realizować tego rodzaju algorytmy na zwykłych komputerach domowych, a obecnie można zmieścić algorytm inferencji na małym kompaktowym mikrokontrolerze. Przyjrzyjmy się, jak to jest możliwe.

Algorytmy uczenia maszynowego są coraz wydajniejsze, co pozwala na uruchamianie ich na coraz bardziej kompaktowych systemach. Autor zaprezentowanego systemu – Peter Balch, nudząc się w czasie epidemii, odnalazł w swojej biblioteczce raport IEEE na temat rozpoznawania mowy. Dokument pochodził z późnych lat siedemdziesiątych. „Czy Arduino Nano może zrobić to samo, co komputer z tamtej epoki?" było oczywistym pytaniem, jakie pojawiło się w jego głowie po przeczytaniu tego opracowania.

Jak wypada Nano w porównaniu z maszyną z lat 70. XX wieku? Arduino Nano ma 2 kB RAM, 32 kB pamięci programu i działa z prędkością około 10 MIPS (w zależności od kombinacji instrukcji). Wtedy w użyciu były minikomputery, które działały z szybkością od 0,5 do 8 MIPS i miały od 2 kB do 32 kB pamięci podzielonej pomiędzy pamięć programu i danych. Większość grup badawczych, z opisanych w raporcie IEEE, posiadała maszyny typu PDP-8 lub PDP-11. Jedna z grup miała do dyspozycji ogromny IBM-360 z 128 kB, ale osiągający poniżej 1 MIPS. Inna grupa z kolei zmieniła przeznaczenie systemu kierowania ogniem rakietowym Univac, który pracował z prędkością ok. 1 MIPS. Nano jest więc na właściwym miejscu do zbudowania prostego systemu rozpoznawania mowy, skoro 50 lat temu komputery o podobnej wydajności były w stanie to robić. Oczywiście, istnieją pewne projekty rozpoznawania mowy działające np. na Arduino Nano, ale wymagają one połączenia internetowego i wysyłania wszystkich rozmów np. do Amazona lub Google albo też wymagają mocniejszej platformy niż mały mikrokontroler (najczęściej stosuje się Raspberry Pi). Oczywiście moduł Arduino Nano nie będzie tak wydajny jak większe platformy, ale „czy może w ogóle zrobić coś pożytecznego?" pyta retorycznie autor.

Jednym z istotniejszych problemów związanych z rozpoznawaniem mowy jest ciągłość mowy z użyciem bogatego słownictwa. Na drugim końcu skali tego zagadnienia mamy wykrywanie pojedynczego słowa z ubogiego słownika. Opisywany w tym artykule projekt skupi się na tym drugim zagadnieniu. Jaki jest pożytek z tego rodzaju systemu? Może znaleźć wiele zastosowań – może sterować pracą niewielkiego, przenośnego multimetru czy komunikatora, który nie ma ekranu ani klawiatury. Każdy system,

Tabela 1. Wyprowadzenia modułu z układem MAX9814

Wyprowadzenie modułu	Funkcja
GND	Masa
VDD	Zasilanie (5 V)
GAIN	Wzmocnienie
OUT	Wyjście sygnału analogowego
A/R	Attack/Release – regulacja AGC

jaki ma być kontrolowany bez użycia rąk, mógłby skorzystać ze sterowania używającego prostych poleceń głosowych. Przykładów takiego sprzętu jest wiele – zdalnie sterowany robot, odtwarzacz MP3 do joggingu itp. Istnieje wiele miejsc, w których może się przydać kilkanaście poleceń słownych. Jeśli przyjrzymy się typowym projektom korzystającym z Amazon Alexa czy Apple Siri, to okaże się, że wiele z nich może wykorzystać niewielkie systemy bez połączenia z Internetem do rozpoznawania prostych komend. Finalnie tego rodzaju system można uzupełnić wyjściem głosowym, na przykład za pomocą biblioteki Talkie, aby układ mógł nie tylko słuchać, ale i mówić.

Sprzęt

Do tego projektu wybrano Arduino Nano (ale będzie działał również z Arduino Uno lub Mini lub podobnym, o ile używa mikrokontrolera ATmega328 z zegarem 16 MHz). Układ ten uzupełniony jest mikrofonem ze wzmacniaczem. Autor wybrał wzmacniacz mikrofonowy MAX9814, ponieważ wyposażony jest on w automatyczną kontrolę wzmocnienia. Oprócz tego będzie potrzebna podstawowa wiedza o tym, jak należy programować Arduino. W Internecie dostępnych jest wiele samouczków dotyczących tego ekosystemu.

Moduł ze wzmacniaczem mikrofonowym to koszt około 10 zł. Układ MAX9814 zawiera w sobie wzmacniacz mikrofonowy z blokiem AGC (układ automatycznej kontroli wzmocnienia). Moduł ma 5 wyprowadzeń, do których podłączamy poszczególne sygnały, zgodnie z opisem z **tabeli 1**. Wejście A/R określa parametry pracy algorytmu AGC. Jest to stosunek czasu narastania do opadania podczas regulacji wzmocnienia. Jeśli pin ten podłączymy do masy, to wyniesie on 1:500, jeśli do napięcia zasilania (VDD), to 1:2000, a jeśli pozostawimy niepodłączony, to równy będzie 1:4000. Czas ten jest zależny od pojemności kondensatora, dołączanego do jednej z nóżek układu i wynosi 2,4×C (w milisekundach, gdzie C jest wyrażony w mikrofaradach). Wyprowadzenie Gain steruje poziomem wzmocnienia. Jeśli pin ten zostanie podłączony do masy, to wzmocnienie układu AGC równe będzie 50 dB, a jeśli do plusa, to wyniesie 40 dB. Jeśli pin Gain pozostanie niepodłączony, to wzmocnienie AGC osiągnie 60 dB.

W obwodzie pokazanym na **rysunku 1** pin A/R jest intencjonalnie niepodłączony, a Gain jest połączone z VDD, co daje najniższe możliwe w układzie wzmocnienie. Piny te można podłączyć do wyprowadzeń cyfrowych Arduino i sterować nimi z poziomu oprogramowania. Aby uzyskać stan taki, jak dla wejścia niepodłączonego, należy linię GPIO Arduino ustawić jako wejście. Mikrofon podłączony z układem MAX9814 jest połączony za pomocą ekranowanego kabla do wejścia Arduino. Aby uzyskać najlepszą jakość dźwięku, mikrofon powinien być umieszczony u boku ust, aby uniknąć trzasków przy tzw. głoskach wybuchowych (p, t, k itp.) czy odgłosach oddychania. Wzmocnienie na poziomie 40 dB daje najlepszy stosunek sygnału do szumu dla mikrofonu na wysięgniku blisko ust. Przy większych poziomach wzmocnienia szum tła jest zbyt silny. Układ AGC redukuje sygnał mowy do rozsądnego poziomu, ale kiedy nie mówimy, szum powoli powraca do sygnału.

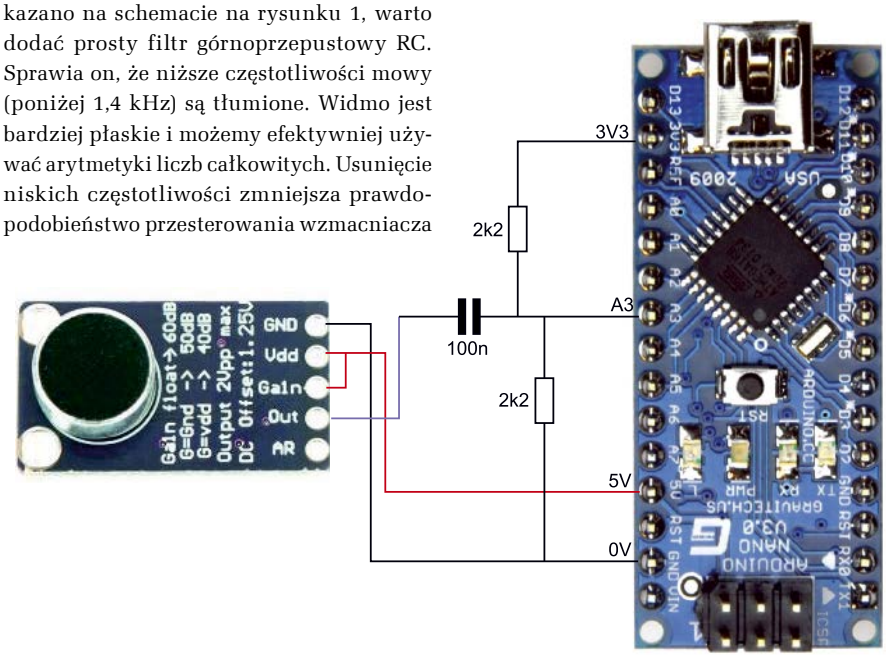
Sygnał dźwiękowy z modułu jest wyśrodkowany na poziomie około 1,25 V i zmienia się w zakresie od 0 do 2,5 V. Przetwornik ADC w module Arduino ma rozdzielczość równą 10 bitów, więc wartość liczbowa mieści się w zakresie od 0 do 1023. Jeśli jako napięcie odniesienia zastosujemy 3,3 V, to pojedynczy najmniejszy bit odpowiada około 3,223 mV, co oznacza, że zakres pracy mikrofonu pokrywają wartości od 0 do 775. Moduł można podłączyć bezpośrednio do wejścia ADC, jednakże jak pokazano na schemacie na rysunku 1, warto dodać prosty filtr górnoprzepustowy RC. Sprawia on, że niższe częstotliwości mowy (poniżej 1,4 kHz) są tłumione. Widmo jest bardziej płaskie i możemy efektywniej używać arytmetyki liczb całkowitych. Usunięcie niskich częstotliwości zmniejsza prawdopodobieństwo przesterowania wzmacniacza i przetwornika ADC. W literaturze specjalistycznej znaleźć można szersze omówienie zasadności stosowania filtrów górnoprzepustowych w przedwzmacniaczach audio do systemów rozpoznawania mowy. Dwa rezystory są używane do wyśrodkowania wejścia ADC wokół napięcia połowy zasilania, czyli 1,65 V.

Finalnie należy zapamiętać, gdzie podłączony jest sygnał analogowy. Konieczne jest to, aby w odpowiedni sposób zdefiniować wejście w programie. Dla układu jak na schemacie na rysunku 1 wystarczy umieścić w szkicu Arduino linijkę:

```
const int AUDIO_IN = A7;
```

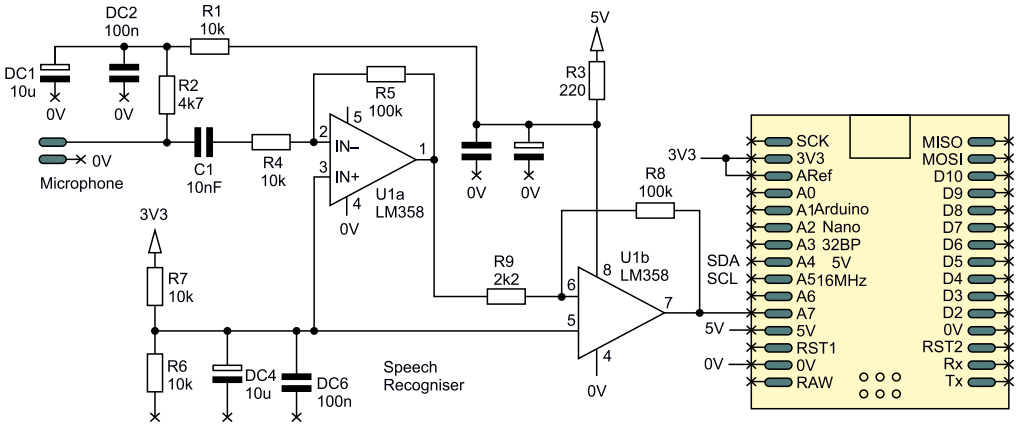
Jeśli nie możecie zdobyć modułu z MAX9814, ale macie pod ręką elementy takie, jak mikrofon elektretowy i wzmacniacz operacyjny (np. LM358), to można zamiast tego skonstruować prosty przedwzmacniacz mikrofonowy. Układ LM358 to dość kiepski wzmacniacz operacyjny – charakteryzuje się dużym szumem, a amplituda sygnału wyjściowego nie będzie większa niż około 1,5 V. Jednakże układ będzie bez problemu działał z zasilaniem 5 V i zapewnia dostatecznie dobrą jakość do tego systemu. Obwód, którego użył autor konstrukcji, pokazano na **rysunku 2**. To nic specjalnego – typowy wzmacniacz mikrofonowy, który można zmontować na płytce stykowej. Całkowite wzmocnienie wynosi około 200. To sprowadza sygnał wyjściowy do odpowiedniego zakresu, jeśli mikrofon znajduje się blisko ust. Elementy C1 i R4 działają jak filtr górnoprzepustowy z delikatnym spadkiem poniżej 1,5 kHz. Wejście nieodwracające wzmacniaczy operacyjnych jest dołączone do połowy potencjału pomiędzy masą a 3,3 V.

System wykorzystuje przetwornik ADC z napięciem odniesienia równym 3,3 V, więc wyjście LM358 będzie miało odpowiednią



Rysunek 1. Schemat ideowy układu z modułem mikrofonu podłączonym do modułu Arduino

amplitudę. Zasilanie 3,3 V ze stabilizatora jest dość zaszumione, więc układ zawiera kondensatory DC4 i DC6 do filtrowania zasilania. Sam LM358 jest zasilany z wyjścia 5 V z płytki Nano. Wyprowadzenie 5 V jest filtrowane przez elementy R3, DC3 i DC5. Napięcie 5 V jest jeszcze filtrowane dalej przez R1, DC1, DC2 i działa jako źródło zasilania dla mikrofonu poprzez R2.



Rysunek 2. Schemat wzmacniacza do mikrofonu

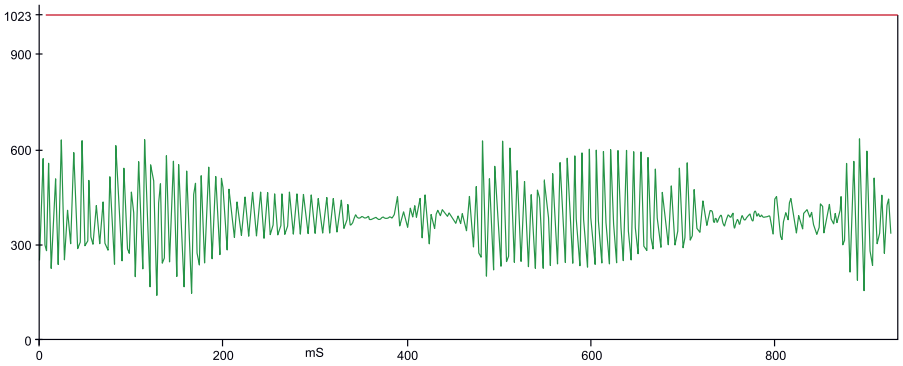
Zbieranie danych

Standardowym sposobem korzystania z ADC w Arduino Nano jest wywołanie funkcji *analogRead()*, ale funkcja ta jest raczej powolna. Inicjalizuje ADC i wybiera właściwy pin wejściowy. Następnie rozpoczyna konwersję i czeka, aż konwersja zostanie zakończona. Wszystko to zajmuje około 100 mikrosekund. Lepiej jest uruchomić konwersję i robić coś innego, czekając na jej wynik. Dlatego też zbieranie danych w omawianym systemie zrealizowano inaczej, niż typowo realizuje się na Arduino. W sekcji *Setup()* zastosowana jest standardowa biblioteka Arduino do inicjalizacji ADC:

```
analogReference(EXTERNAL);
analogRead(AUDIO_IN);
```

Napięcie odniesienia dla ADC jest podawane na pin ARef, który jest podłączony do 3,3 V. Aby skonfigurować wejście analogowe, wystarczy raz wywołać *analogRead()*. Następne odczyty możemy realizować w inny, szybszy sposób.

W głównej pętli programu ustawiamy bit ADSC (*ADC Start Conversion*), aby rozpocząć konwersję. Biblioteka Arduino skonfigurowała ADC w trybie pojedynczej konwersji, więc trzeba ustawić bit ADSC, aby rozpoczynał każdą konwersję. Bit ADIF (flaga przerwania ADC) jest ustawiana po zakończeniu konwersji. Oznacza to, że można robić coś innego, gdy ADC jest zajęty. Skasowanie ADIF (ustawienie go na 1) powoduje, że bit ADIE (*ADC Interrupt Enable*) zostaje wyczyszczony przez bibliotekę Arduino, więc żadne przerwanie nie występuje, a flaga jest sprawdzana, aby wykryć, kiedy konwersja ADC jest zakończona. Wynik konwersji ADC jest 10-bitowy i odczytywany poprzez odczytanie 8-bitowego rejestru ADCL, a następnie rejestru ADCH. Kiedy czytasz ADCL, wartość w ADCH jest zamrożona, dopóki też nie zostanie odczytana.



Rysunek 3. Przykładowy przebieg zebrany za pomocą urządzenia

Dzięki temu mamy pewność, że nie zostaną pomieszane wartości kolejnych ramek – różnych próbek. Rejestry ADCL i ADCH trzeba odczytywać we właściwej kolejności. Kompletny kod skryptu jest bardzo prosty i został pokazany na **listingu 1**.

Kod przetwarzania próbki jest wykonywany, podczas gdy ADC realizuje następną konwersję ADC. Napięcie ze wzmacniacza jest wycentrowane wokół 512. Do dalszego przetwarzania sygnału dobrze jest, aby było wyśrodkowane wokół 0. Więc odejmujemy średnią bieżącą przychodzącej wartości od zmiennej wartość, jak pokazano na **listingu 2**.

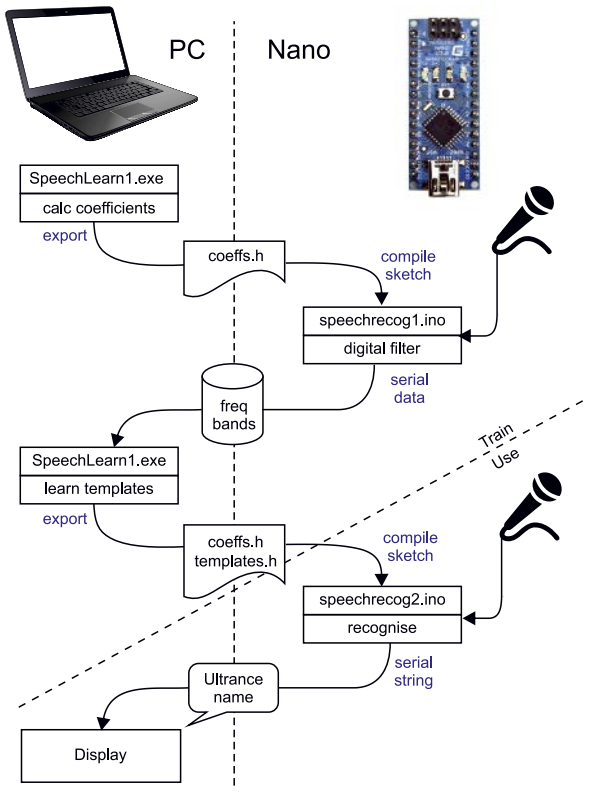
Kompletny szkic Arduino, który przetwarza wyniki ADC, można pobrać ze strony projektu. Pobiera on próbki z prędkością około 9 ksps. Może

Listing 1. Kod odpowiedzialny za odczyt wartości z przetwornika ADC

```
while (true) {
  while (!getBit (ADCSRA, ADIF)); // czekaj na ADC
  byte wart1 = ADCL;
  byte wart2 = ADCH;
  bitSet(ADCSRA, ADIF); // wyczyść flagę
  bitSet(ADCSRA, ADSC); // rozpocznij konwersję ADC
  int wartosc = wart1;
  wartosc += wart2 << 8;
  // tutaj wstawić należy przetwarzanie próbki
}
```

Listing 2. Przetwarzanie wyników z przetwornika ADC

```
static int zero = 512;
jeśli (wartość <zero)
  zero--;
else
  zero ++;
wartosc = wartość - zero;
```



Rysunek 4. Ogólna architektura powstawania oprogramowania dla systemu

przesyłać wartości do komputera przez łącze szeregowe, ale transmisja szeregowa spowalnia je do około 1100 sps (przy 57600 baud). Korzystając z narzędzia Serial Plotter w Arduino IDE, można uzyskać wyświetlenie przebiegu sygnału tak, jak pokazano na **rysunku 3**.

Architektura oprogramowania

Cały system zastosowany do nauki sieci i do prowadzenia inferencji na mikrokontrolerze jest nieco skomplikowany... Szkolenie sieci neuronowych odbywa się na komputerze PC, ale przeszkolony system działa już w całości na Arduino. Schemat konstrukcji tego systemu i sposób postępowania z danymi zostały pokazane w uproszczeniu na **rysunku 4**. Arduino wysyła przykładowe wypowiedzi do komputera, a komputer generuje szablony wypowiedzi. Komputer eksportuje szablony jako plik ****.h*, który jest kompilowany w szkicu. Szkic uruchomiony na mikrokontrolerze może wtedy rozpoznawać wypowiedzi bez połączenia z komputerem.

Program *SpeechRecog1.exe* dla systemu Windows oblicza współczynniki dla filtra cyfrowego, opisanego w dalszej części artykułu. Współczynniki filtra cyfrowego są eksportowane jako plik *Coeffs.h*. Szkic *speechrecog1.ino* jest następnie kompilowany przy użyciu wygenerowanych współczynników.

Program Speechrecog1.ino nagrywa przykładowe wypowiedzi i wysyła je do komputera. Na komputerze program SpeechRecog1 oblicza na ich podstawie szablony, które służą do rozpoznawania tych wypowiedzi. Opcjonalnie *SpeechRecog1* zbiera więcej wypowiedzi do testowania. Testuje szablony przy użyciu zebranych danych. Szablony są eksportowane jako plik *Templates.h*. Kolejny szkic Arduino – *speechrecog2.ino,* jest kompilowany przy użyciu plików *Templates.h* i *Coeffs.h*. Szkic *speechrecog2.ino* wykorzystuje szablony do rozpoznawania wypowiedzi. W dalszej części artykułu przyjrzymy się bliżej każdej z części systemu.

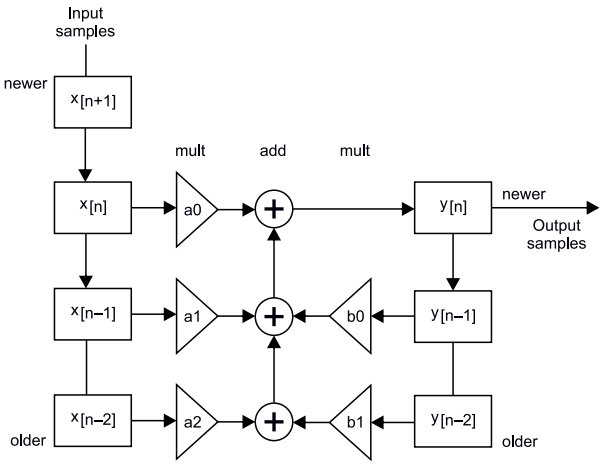
Filtry pasmowe

Typowe słowo może trwać ok. sekundy. Arduino Nano ma tylko 2 kB pamięci RAM, więc nie możemy przechowywać wszystkich próbek dźwięku do analizy. Konieczne jest wykonanie większości analiz w czasie rzeczywistym, gdy przybywają próbki. Rozpoznawanie mowy zwykle rozpoczyna się od pomiaru „energii" w różnych pasmach częstotliwości – amplitudy sygnału. Pierwszym etapem jest przepuszczenie sygnału wejściowego przez różne filtry pasmowo-przepustowe, aby wyznaczyć spektrum.

Najpopularniejszym sposobem filtrowania danych jest wykonanie transformaty Fouriera na sygnale w celu uzyskania jego widma. Arduino Nano nie ma wystarczającej mocy obliczeniowej, aby obliczyć taką transformatę w czasie przychodzenia próbek. Bardziej odpowiednim sposobem analizy tych danych jest użycie filtrów cyfrowych. Filtr cyfrowy wykonuje proste obliczenia na poprzednich N próbkach itp., aby obliczyć następną wartość wyjściową filtra. Schemat na **rysunku 5** pokazuje typowy filtr. Obliczenia można zapisać np. w następujący sposób:

$$y[n] = a0 \cdot x[n] + a1 \cdot x[n-1] + a2 \cdot x[n-2] + + b1 \cdot y[n-1] + b2 \cdot y[n-2]$$

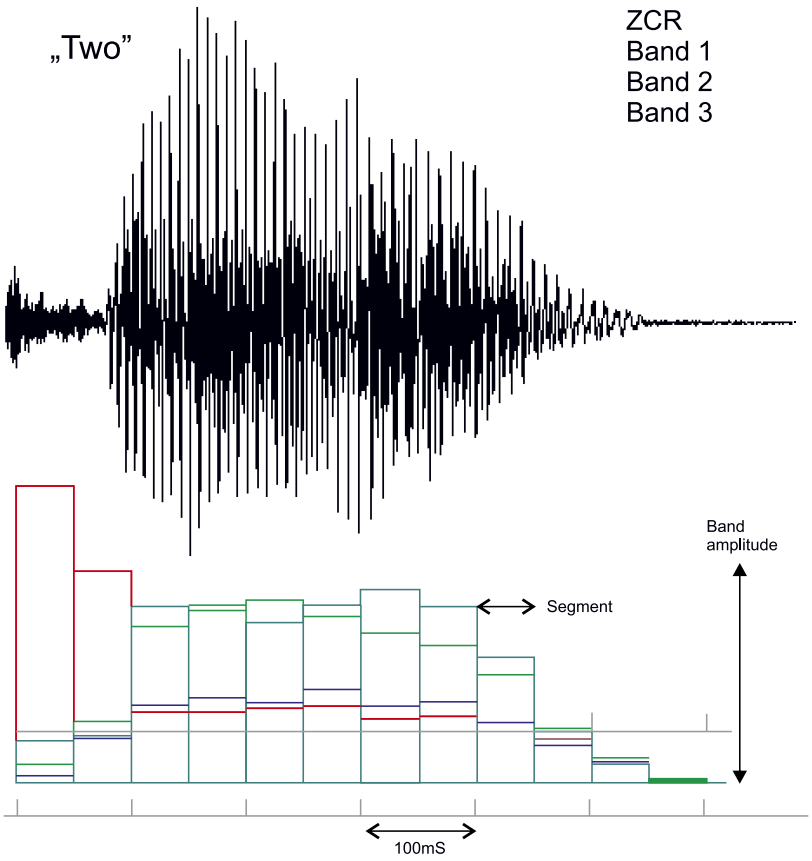
gdzie x[n] jest wejściową wartością próbki, a y[n] jest wartością wyjściową; x[n-1], y[n-2] itd. to poprzednie wartości.



Rysunek 5. Schemat blokowy filtrów pasmowych układu

Program zachowuje dwie poprzednie wartości wejściowe i dwie poprzednie wartości wyjściowe. Ponieważ przechowuje 2 z każdej wartości, jest znany jako filtr drugiego rzędu. Jeżeli wyjście zależy tylko od poprzednich wartości wejścia, to nazywa się je filtrem o skończonej odpowiedzi impulsowej (FIR). Jeżeli wyjście zależy tylko od poprzednich wartości wyjściowych, jest to filtr o nieskończonej odpowiedzi impulsowej (IIR). (FIR jest czasami nazywany filtrem nierekurencyjnym, a IIR filtrem rekurencyjnym). W przypadku filtra pasmowoprzepustowego rząd filtra określa, jak strome jest pasmo filtra. Im wyższy rząd, tym większą mamy kontrolę nad krzywą odpowiedzi filtra. Oczywiście im wyższy rząd, tym więcej potrzebnych jest współczynników i tym więcej obliczeń matematycznych trzeba wykonać. Filtr FIR wymaga większej liczby współczynników i większej liczby obliczeń matematycznych, aby uzyskać taką samą krzywą odpowiedzi jak filtr IIR. Ale filtr IIR jest mniej stabilny. Filtr IIR jest mniej odporny na zakłócenia i błędy. Jest to szczególnie znamienne, gdy używa się arytmetyki liczb całkowitych, tak jak robione jest to w Arduino Nano.

Trudno jest oszacować wartość prędkości tego, jak Nano może dodawać i mnożyć. Zależy to m.in. od sposobu pobierania i przechowywania wartości. Dodawanie 8-bitowe zajmuje od 0,4 do 0,9 µs. Mnożenie trwa



Rysunek 6. Przykładowy szablon

około 50% dłużej. 16-bitowe dodawanie lub mnożenie zajmuje dwa razy więcej, a 32-bitowe dodawanie lub mnożenie zajmuje około 5 razy więcej czasu niż jeden bajt. Arytmetyka zmiennoprzecinkowa z dzieleniem zajmuje znacznie więcej czasu, dlatego też nie jest stosowana w tym systemie.

Jeśli chcemy uzyskać częstotliwość próbkowania równą 8000 sps, czyli 125 µs na próbkę, z 4 pasmami częstotliwości, to mamy 31 µS na próbkę na pasmo. W tym czasie trzeba jeszcze zebrać dane z ADC, obliczyć amplitudę pasm i zapisać wyniki w tablicy. W rezultacie system ograniczony jest do kilkunastu operacji arytmetycznych na próbkę. Nie można sobie pozwolić na więcej niż filtr IIR drugiego rzędu. W artykule źródłowym (link na końcu artykułu) znajduje się dokładniejszy opis, jak wyznaczane są współczynniki dla filtrów cyfrowych i jak są one implementowane i testowane.

Szablony

Arduino dzieli całą wypowiedź na segmenty o długości 50 ms każdy (w niektórych dokumentach nazywane są one ramkami). W każdym segmencie system mierzy amplitudę każdego z pięciu pasm. Zakłada się, że wypowiedź ma 13 segmentów, czyli w sumie zapisana jest jako 65 16-bitowych liczb całkowitych obejmujących 0,65 s. Oczywiście niektóre wypowiedzi są krótsze, więc kilka ostatnich segmentów będzie bliskich zeru, a niektóre wypowiedzi są dłuższe, więc ostatnie ich części zostaną utracone.

Zakłada się, że wypowiedź rozpoczyna się, gdy całkowita energia w pasmach przekroczy pewien zadany próg. Po zapisaniu wszystkich 13 segmentów system musi wybrać, które z zapisanych, znanych słów najbardziej przypomina wygenerowane 65 liczb. Załóżmy, że wypowiedzi, które próbujemy rozpoznać, to cyfry od zera do dziewięciu. Pierwszym krokiem jest normalizacja danych. Ma to na celu sprawienie, że wszystkie wypowiedzi będą miały taką samą głośność. Dane dla każdego pasma każdego segmentu są mnożone przez stałą, dzięki czemu mają stałą średnią energię (np. równą 100).

Analizując książkę IEEE z lat 70., autor napotkał wiele opisów technik, które stosowano do analizy wypowiadanych słów, jednakże jak w przypadku wielu prac badawczych autorzy i tutaj nie chcieli zdradzać wszystkich swoich tajemnic, więc zamieścili jedynie niejasny zarys tego, jak rozpoznają wypowiadane cyfry. Większość z nich wydawała się całkiem zadowolona z samego wykonania kilku nagrań, dokonania transformacji Fouriera i narysowania kilku wykresów. To jednak nie satysfakcjonuje autora projektu, więc oczywiście chciał pójść krok dalej.

Listing 3. Obliczanie różnicy pomiędzy szablonem a analizowaną wypowiedzią

```
for t = each template
    difference[t] = 0
    for seg = each segment
       for band = each band
          difference[t] = difference[t] + abs(template[t,seg,band] - incoming[seg,band])
```

Pierwszą myślą było zastosowanie technik statystycznych, takich jak analiza składowych głównych, analiza czynnikowa, analiza skupień itp. Najbardziej oczywista byłaby liniowa analiza dyskryminacyjna (LDA). Po kilku próbach okazało się jednak, że taki algorytm kiepsko radzi sobie z odróżnianiem jednego rodzaju wypowiedzi od drugiego. Inną alternatywą jest śledzenia formantów – maksimów w widmie, zazwyczaj co najmniej dwóch największych. Algorytm ten śledzi amplitudy i częstotliwości tych maksimów, jednak ma sens jedynie w przypadku ciągłego widma, a nie, gdy do dyspozycji mamy zaledwie 4 pasma częstotliwości. Z tych powodów wykorzystanie prostych szablonów, takich jak pokazany na **rysunku 6**, jest najprostsze i pozwala na najlepsze rozróżnianie różnych cyfr. Szablon jest przykładem typowej wypowiedzi danego rodzaju. Każdy z nich zawiera 65 wartości. Każda wartość jest porównywana z odpowiadającą jej w analizowanej wypowiedzi. Różnica obliczana jest w sposób pokazany na **listingu 3**.

Jednak niektóre zespoły są ważniejsze niż inne, a niektóre segmenty są ważniejsze niż inne. Na przykład dla cyfry 3 (*three* po angielsku) pierwsza część słowa jest znacznie bardziej istotna do rozpoznania słowa niż ostatnie dwie litery e. Dlatego też każda liczba w szablonie ma przypisane wagi:
difference[t] = difference[t] + importance[t,seg,band] * abs(template[t,seg,band] - incoming[seg,band])

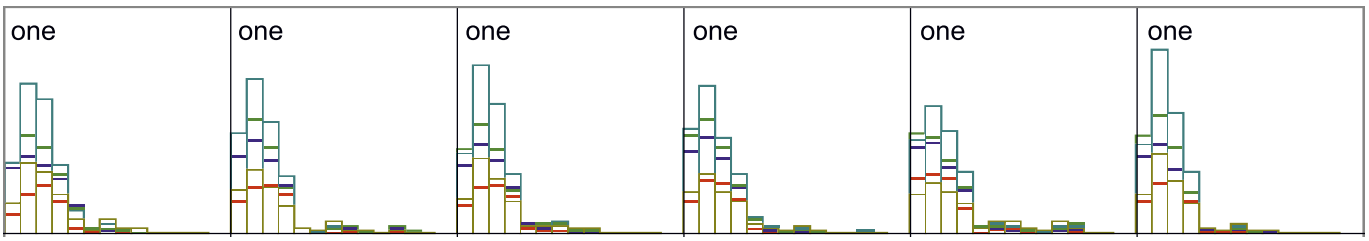
Jak ocenia się znaczenie danego pasma czy segmentu? Jeśli wartości a(t, seg, band) różnią się znacznie dla tej klasy wypowiedzi, wartość szablonu jest mniej istotna niż wtedy, gdy wartości są prawie zawsze takie same. Waga definiowana jest zatem jako 1/(50 + odchylenie standardowe).

Szkolenie algorytmu

Program *SpeechRecog1.exe*, użyty do obliczenia współczynników, może być również użyty do wyznaczania szablonów. Wystarczy podłączyć Arduino do komputera i wybrać odpowiedni port COM. Do Arduino ładujemy szkic *speechrecog1.ino*. Wykorzystuje on ADC do próbkowania mowy z prędkością około 8000 sps. Filtruje próbki do czterech pasm częstotliwości oraz wyznacza ZCR i przechowuje 13 segmentów danych o długości 50 ms każdy. Wypowiedź zaczyna się, gdy całkowita energia w paśmie przekroczy zdefiniowany próg. Po zapisaniu 13 segmentów danych otrzymywanych jest 65 liczb, które przesyłane są do komputera.

W programie należy kliknąć zakładkę Szablony, a następnie zakładkę Trenuj szablony, aby wyświetlić wypowiedzi, które posłużą do obliczenia szablonów. W programie klikamy na menu plik i otwieramy plik *Train2raw.txt* (przykładowe nagrania są dostępne na stronie z projektem, ale można oczywiście nagrać i podstawić własne nagrania). W programie można kliknąć dowolną komórkę, a zostaną wyświetlone segmenty dla tego przykładu. Po wybraniu kilku komórek wszystkie zostaną wyświetlone, aby można było je ze sobą porównać (**rysunek 7**). Po kliknięciu na dowolną komórkę w siatce wyświetlona zostanie na wykresie wypowiedź – oś pozioma to czas, a pionowa to amplituda każdego pasma (**rysunek 8**). Czerwony pasek to ZCR. Po kliknięciu lewym przyciskiem kolumny siatki program wyświetli średnią oraz odchylenie standardowe. Program oblicza średnią i odchylenie standardowe dla każdego segmentu i pasma dla każdego szablonu. Innymi słowy, 10 szablonów zawiera teraz średnią z zebranych wcześniej danych.

Na tym etapie można porównywać wszystkie przykłady z szablonem. Który z nich jest najbardziej podobny do szablonu? Można je przesuwać w prawo i w lewo, aby uzyskać najlepsze dopasowanie. Następnie program porównuje próbkę z szablonem dla wszystkich wierszy, a wyniki wyświetlane są po prawej stronie. Pokazywana liczba jest różnicą między wypowiedzią a tym szablonem. Najkrótsza odległość jest najlepsza i ta jest wyświetlana w siatce, jako najlepsze dopasowanie.



Rysunek 7. Szablony uzyskane dla różnych słów

Następnie można przystąpić do nauki algorytmu. Przekazuje się do niego słowa, nagrywane poprzez Arduino. Po nauczeniu systemu można go przetestować zestawem treningowych słów. Lista wypowiedzi nie musi pasować do zestawu uczącego – można dodać kilka niepoprawnych słów.

Gdy mamy już zestaw szablonów, które dają zadowalające efekty, należy je wyeksportować do Arduino jako plik *Templates.h*, gotowy do dołączenia do szkicu w Arduino C. Należy skopiować plik *Templates.h* do tego samego katalogu, co szkic *speechrecog2.ino*. Powinieneś także skopiować plik *Coeffs.h* do katalogu szkicu. Można teraz skompilować go i wgrać do pamięci modułu Arduino.

Gotowe rozpoznawanie głosu

Szkic *speechrecog2.ino* wykonuje rozpoznawanie mowy na Arduino Nano, Uno, Mini itp., po skopiowaniu plików *Templates.h* i *Coeffs.h* do tego samego katalogu, w którym znajduje się szkic *speechrecog2.ino*, który należy ponownie skompilować i przesłać do modułu Arduino.

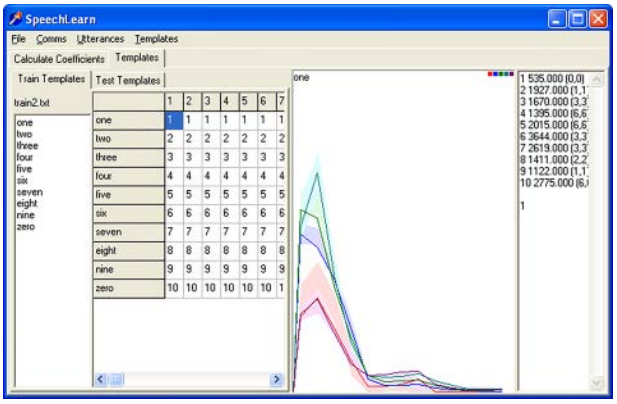
Działanie systemu obrazuje **rysunek 9**. Przeteornik ADC w Arduino służy do digitalizacji przychodzącego sygnału audio. Wynikiem jest 16-bitowa zmienna int wyśrodkowana na 0. Kilka cyfrowych filtrów pasmowoprzepustowych IIR dzieli sygnał na pasma częstotliwości. Czas dzielony jest na odcinki po 50 ms. Mierzona jest amplituda każdego pasma w każdym segmencie. Ustalona liczba segmentów (w obecnej implementacji 13) stanowi wypowiedź. Wypowiedź zaczyna się, gdy amplituda przekracza ustalony próg. System mierzy średnią amplitudę całej wypowiedzi, aby można było znormalizować dane wejściowe. Następnie obliczana jest szybkość przejścia przez zero (ZCR) sygnału.

Wartości amplitud w poszczególnych pasmach są porównywane z wartościami z szablonu. Segmenty można przesuwać w lewo lub w prawo, aby poprawić dopasowanie. Zgłaszane jest najlepsze dopasowanie. Szkic *speechrecog2.ino* wysyła tekst rozpoznanego słowa do komputera przez łącze szeregowe, ale można użyć go w dowolnym projekcie do kontrolowania działania innych urządzeń.

Podsumowanie

Jak donosi autor, w idealnych warunkach system uzyskuje od 90% do 95% poprawnego rozpoznawania, czyli mniej więcej tyle, ile uzyskiwały systemy w latach 70. Cel został osiągnięty. Z pewnością możliwe jest ulepszenie tego kodu, aby uzyskiwał lepsze wykrywanie i rozpoznawanie słów. „To projekt „eksperymentalny. To coś, nad czym możesz popracować i ulepszać. To nie jest coś, co możesz po prostu zbudować i zadziała za pierwszym razem" – podsumowuje autor.

Jak zatem poprawić system? Autor zachęca wszystkich zainteresowanych do rozbudowy i poprawiania projektu. Ma nawet kilka propozycji, o których pisze. Można użyć innego algorytmu rozpoznawania. Być może będzie to wymagało zbudowania własnego oprogramowania do szkolenia algorytmu na komputerze PC, ale Arduino jest gotowe do dostarczania wszystkich wymaganych danych. Do systemu można dodać bibliotekę Talkie, aby przekazać informację zwrotną o rozpoznanym słowie w postaci mówionej.



Rysunek 8. Program do szkolenia sieci

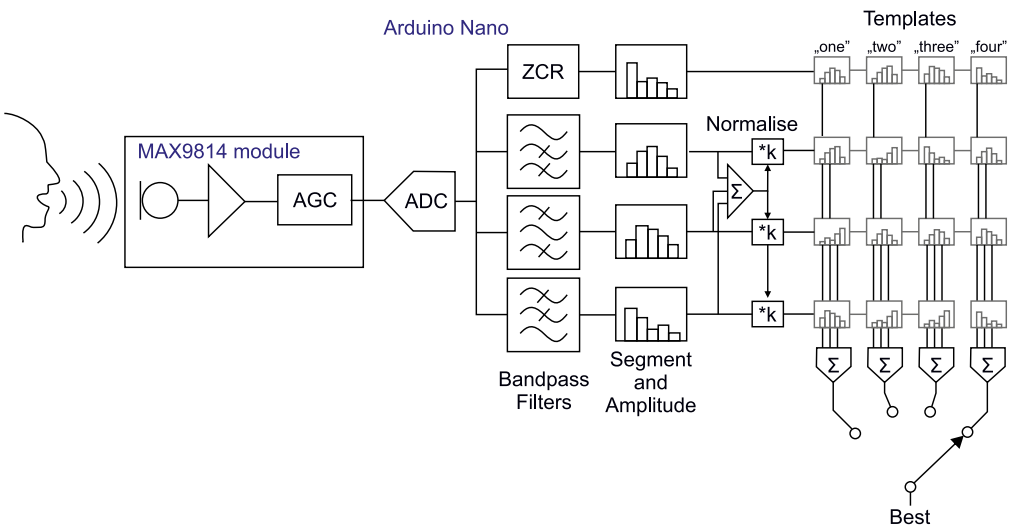
Wszystkie współczesne systemy rozpoznawania mowy rozpoczynają od transformaty Fouriera, po której może następować np. analiza spektrum lub wykorzystująca współczynniki LPC. Arduino z ATmega328 nie jest wystarczająco szybkie, aby to zrobić, gdy przychodzi dźwięk i nie ma wystarczająco dużej pamięci, aby pomieścić próbki kompletnej wypowiedzi do późniejszej analizy (ATmega328 może wykorzystywać istniejące współczynniki LPC do generowania mowy w czasie rzeczywistym, ale nie może obliczyć współczynników). Z uwagi na te ograniczenia Arduino skazane jest na użycie filtrów cyfrowych. Punktem wyjścia dla każdego algorytmu rozpoznawania mowy będą pasma i segmenty, opisane powyżej. To, jak analizowane są te pasma i segmenty oraz jak wykrywane są poszczególne słowa, zależne jest już od implementacji konkretnego algorytmu, który (po przeszkoleniu) można uruchomić na Arduino.

W opisanym systemie wykorzystano „algorytm K najbliższych sąsiadów", ale jest wiele innych, które można wypróbować w tym zastosowaniu. Autor testował część z nich, takie jak np. liniową analizę dyskryminacyjną LDA (nie zadziałała ona, być może dlatego, że wypowiedzi nie można rozdzielić liniowo). Maszyna wektorów nośnych (SVM) powinna być w stanie obejść ten problem, ale jak przyznaje się autor – nie ma on w tym zakresie doświadczenia. Podobnie kwadratowa analiza dyskryminacyjna (QDA) powinna działać z danymi nieliniowymi, które można rozdzielić, ale to także dla autora jest mało znanym zagadnieniem, podobnie jak i inne algorytmy, takie jak np. wielomianowa regresja logistyczna.

Inne algorytmy, które są popularne w rozpoznawaniu mowy, to np. ukryte modele Markowa (HMM), być może dlatego, że stanowią lepszą alternatywę dla dynamicznego dopasowania w czasie, co jest jednakże bardziej istotne dla dłuższych wypowiedzi. Najlepiej sprawdziłaby się tutaj sieć neuronowa. Wielowarstwowe sieci neuronowe mogą rozpoznawać wzorce, których nie można rozdzielić liniowo, ale wymagają one ogromnych ilości danych szkoleniowych. Dostępnych jest wiele bezpłatnych programów treningowych sieci neuronowych, na przykład w Pythonie lub R. Można też używać algorytmów genetycznych do stworzenia klasyfikatora, który będzie klasyfikował słowa.

Nikodem Czechowski, EP

Źródło: https://bit.ly/3tfch56



Rysunek 9. Uproszczony schemat systemu rozpoznawania głosu