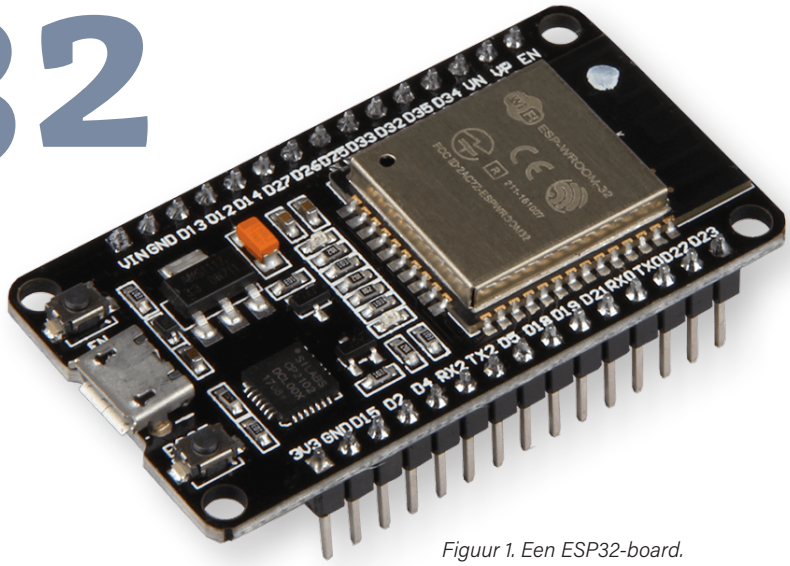
Acoustic fingerprinting met de ESP32

songherkenning met het open-source project Olaf



Figuur 1. Een ESP32-board.

Joren Six, Universiteit Gent (België)

Een paar jaar geleden kreeg computerwetenschapper Joren Six de opdracht om een draagbaar computerplatform – dat wil zeggen, een microcontroller – een liedje te laten herkennen. Zijn experimenten met een ESP32 leidden tot een volwaardig multiplatform open-source project dat hij Olaf noemde, wat staat voor "Overly Lightweight Acoustic Fingerprinting". Olaf is een bibliotheek voor muziekherkenning die eenvoudig te gebruiken is op embedded platforms met weinig geheugen en rekenkracht. Volg hem bij de ontwikkeling!

Ergens rond 2019 was ik computerwetenschapper aan de Universiteit Gent in België. Ik kreeg de opdracht om audio-herkenningstechnologie te ontwikkelen voor een elektronisch kostuum. Het concept bestond erin om de lampjes in het kostuum zich te laten synchroniseren met een specifiek liedje. Slechts één bepaald liedje zou de lampjes mogen activeren; alle andere muziek moest genegeerd worden. Normaal gesproken worden muziekherkenning en synchronisatie bereikt met behulp van acoustic fingerprinting-technieken. De uitdaging was dat dit op een betaalbare batterijgevoede microcontroller met weinig rekenkracht en geheugen moest werken. Destijds bouwde ik met goed gevolg een eerste prototype, maar het idee bleef in mijn hoofd rondspoken. Toen de vierde verjaardag van mijn dochtertje naderde, besloot ik het prototype om te bouwen tot een extravagant verjaardagscadeau: een 'Elsa'-jurk die reageert op het liedje "Let It Go" van de soundtrack van *Frozen* [1]. Voortbordurend op het eerste prototype bestelde ik een RGB-LED-strip, een robuuste Li-Ion-accu, een I²S digitale microfoon en natuurlijk een Elsa-jurk.

Ik had een ESP32-microcontroller (**figuur 1**) bij de hand en gebruikte deze als de centrale component van het systeem. Hij ondersteunt I²S, bevat een floating-point unit (FPU), is eenvoudig te combineren met LED-strips en heeft voldoende RAM-geheugen. De FPU vereenvoudigt het gebruik van dezelfde code op zowel conventionele computers als embedded apparaten, omdat fixed-point wiskunde niet meer nodig is.

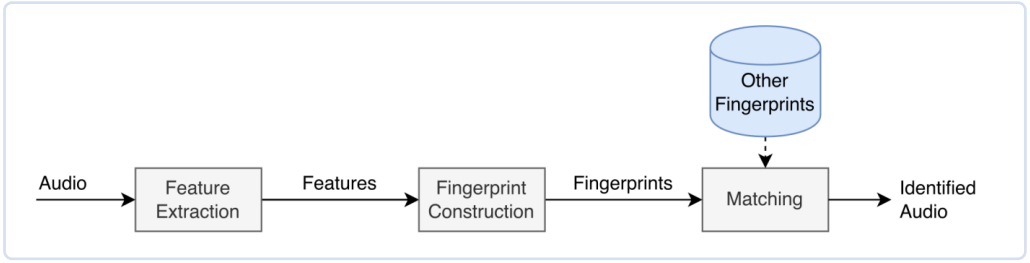
Bouw van de ESP32-versie

De eerste versie van het project maakte gebruik van een ESP32 Thing van Sparkfun – gekozen vanwege de geïntegreerde accuvoeding – in combinatie met een MEMS-microfoon, om precies te zijn een INMP441. Deze combinatie werd aangevuld met een LED-strip met individueel adresseerbare LED's, een generiek type dat gemakkelijk te vinden is op eBay of AliExpress.

Hardwarematig gezien is het een eenvoudige klus. Het is een kwestie van de I/O-pinnen van de microfoonmodule (SDA, SCK, WS) aan te sluiten op geschikte GPIO's op de ESP32, zoals GPIO 32, 33 en 25. En natuurlijk moet de microfoonmodule ook gevoed worden.

De bedrading van de LED's (**figuur 2**) is afhankelijk van het type LED-strip dat wordt gebruikt. Voor WS2812B-gebaseerde strips zijn drie draden (voeding, aarde, data) nodig. Ik heb de *FastLED*-bibliotheek gebruikt in de ESP32-code, dus als je het project wilt kopiëren, zorg er dan voor dat de LED-strip die je gebruikt compatibel is met *FastLED*. Na het solderen van de componenten en (met hulp van mijn partner) het innaaien van de LED-strip, was het project klaar. In de video die op [1] is te vinden, kun je het resultaat van onze inspanningen bekijken.

Figuur 2. RGB LED-strip.



Figuur 3. Blokschema van geluidsherkenning.

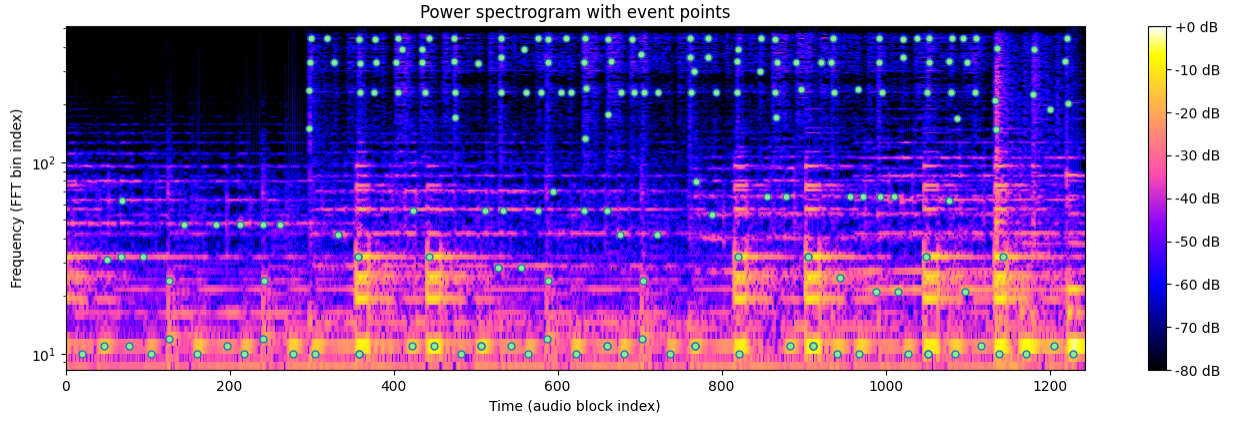
De video laat aanvankelijk een liedje horen dat niet wordt herkend en de lampjes niet activeert. Vervolgens wordt 'Let It Go' gespeeld en correct herkend. Als het liedje pauzeert, blijven de lampjes even aan voordat ze uitgaan, om onderbrekingen bij de herkenning op te vangen. Tot slot wordt het nummer hervat en opnieuw correct herkend. Je kunt op [2] meer lezen over de laatste updates van het project.
Natuurlijk gebeurt, zoals meestal het geval bij microcontrollerprojecten, alle tovenarij in de code.

Onder de motorkap

Hoe kunnen we computers gebruiken om songs of muziek te herkennen? Dat kan op een aantal manieren. Een veelgebruikte benadering is herkenning op basis van spectrale pieken. Dit is de technologie die gebruikt wordt door Olaf en bekende muziekherkenningsdiensten zoals Shazam.
Het concept hierachter is vrij eenvoudig: de app neemt de audio die op je telefoon is opgenomen en zet deze om in een formaat dat een computer gemakkelijk kan vergelijken met andere nummers (**figuur 3**). In de geluidsleer wordt dit proces aangeduid als audio fingerprinting, waarbij een liedje wordt gecomprimeerd tot een unieke handtekening die herkenbaar blijft, zelfs te midden van veel achtergrondgeluid. Deze handtekeningen zijn gebaseerd op spectrogrammen, die de frequentie-inhoud van het nummer in de loop van de tijd visueel weergeven. Spectrogrammen geven ook het vermogensniveau van het signaal weer en tonen de waargenomen luidheid in de vorm van kleurvariaties.
Als Shazam echter rechtstreeks spectrogrammen als deze zou vergelijken, zou het extreem lang duren voordat er een resultaat verschijnt. Een belangrijke doorbraak was dat het algoritme zich concentreert op de pieken in het spectrogram, omdat deze precies die informatie bevatten die ook door het menselijk brein wordt verwerkt.
Olaf neemt dus een geluidsmonster op en berekent vervolgens een spectrogram. Vervolgens wordt het spectrogram vereenvoudigd tot een spreidingsdiagram van de frequentiepieken. Een voorbeeld van zo'n spectrogram met gemarkeerde pieken is te zien in figuur 4.
Nu moeten deze spreidingsplots, die in wezen de meest prominente signalen bij verschillende frequenties in de loop van de tijd weergeven, worden vergeleken met een database van een groot aantal bekende songs. Of, in het geval van Elsa's kostuum, met één specifieke song. In plaats van te zoeken naar een overeenkomst in een database voor al deze punten in de exacte volgorde, is een slimme techniek om naast elkaar liggende pieken te verbinden om zo een groot aantal paren te creëren. Vervolgens zoekt het algoritme naar corresponderende paren binnen een gestructureerde database die een willekeurig aantal nummers bevat. Als er genoeg overeenkomsten worden gevonden en deze in de loop van de tijd correct op elkaar aansluiten, kan Olaf (of Shazam) het nummer identificeren.

De ontwikkeling van het project

De initiële code voor het prototype uit 2019 was niet bijzonder goed georganiseerd. Tijdens mijn tweede poging voerde ik echter aanzienlijke verbeteringen door, waardoor ik een niveau bereikte dat ik de code kon delen op GitHub. Op dat moment kreeg het project de naam *Olaf – Overly Lightweight Acoustic Fingerprinting* [3].
De code onderging verschillende iteraties en breidde zich uit voorbij het oorspronkelijke doel, en ontwikkelde zich tot een veelzijdig, algemeen toepasbaar acoustic fingerprinting-systeem met talloze potentiële toepassingen. De indrukwekkende prestaties van Olaf kunnen onder andere worden toegeschreven aan het resource-efficiënte ontwerp en het gebruik van de PFFFT-bibliotheek.



Figuur 4. Een spectrogram laat zien hoe welke frequenties voorhanden zijn in muziek op een bepaald moment. De groene stippen zijn geëxtraheerd door Olaf en tonen pieken in het spectrogram.

Dit acroniem zegt sommige lezers misschien niet veel, vooral als net als bij Bob Pease je favoriete programmeertaal soldeertin is! PFFFT [4] staat voor *Pretty Fast FFT*; het is een bibliotheek die is ontworpen als een veel lichter alternatief voor het bekende *FFTW*. Het is inderdaad compact en werkt lekker snel, waardoor het ideaal is voor de ESP32. Op embedded apparaten worden referentie-fingerprints opgeslagen in het geheugen, waardoor er geen database nodig is. Op traditionele computers worden de fingerprints opgeslagen in een krachtig databasesysteem – LMDB. Hoewel het buiten het bestek van dit artikel valt om de voordelen ervan te bespreken, moedig ik lezers aan om dit bij interesse verder te onderzoeken.

Olaf is niet langer alleen een coole gadget gebaseerd op ESP32. Het is veranderd in een volwassen applicatie/bibliotheek, ontworpen voor landmark-based acoustic fingerprinting. Olaf kan op een efficiënte manier fingerprints extraheren uit een audiostroom, waarna deze kunnen worden opgeslagen in een database of overeenkomsten tussen geëxtraheerde en opgeslagen fingerprints kunnen worden geïdentificeerd.

Er bleken weinig lichtgewicht bibliotheken voor acoustic fingerprinting te bestaan die eenvoudig te implementeren zijn op embedded platformen, die vaak ernstige beperkingen hebben wat betreft geheugen en rekenkracht.

Olaf is geschreven in C, en ik heb geprobeerd om de code zo overdraagbaar mogelijk te houden. Hij richt zich voornamelijk op 32-bit ARM chips zoals bepaalde Teensy-modellen, specifieke Arduino-boards en de ESP32. Andere moderne embedded platformen met vergelijkbare specificaties kunnen ook compatibel zijn. Ongetwijfeld zal Olaf, dat open-source is, de bouw van innovatieve projecten stimuleren die muziek/geluidsherkenning en IoT-apparaten combineren!

Een belangrijk gevolg van deze inspanningen voor overdraagbaarheid is dat Olaf ook op PC's draait, en bovendien snel. De code kan gecompileerd worden om lokaal op je PC te draaien, maar er is een ander alternatief: Olaf kan ook in een webbrowser draaien! Zie het kader **Uitvoering in een webbrowser**.

De broncode, met een werkend voorbeeld voor ESP32 plus kleine debug-tools die ik heb geschreven, is te vinden in mijn GitHub-repository. Er is ook een PlatformIO-projectbestand ter referentie [6].

Bouw het zelf met een ESP32

Voor audioverwerking had het Olaf-project een beetje RAM nodig, wat vaak een probleem is bij veel microcontrollers. Met een key-value store op een PC had Olaf ongeveer 512 KB RAM nodig, terwijl de ESP32-versie aanzienlijk minder nodig had – ongeveer 200 KB. Ik werk slechts sporadisch met microcontrollers, dus een gebruiksvriendelijke omgeving is belangrijk: de beperkte tijd die ik met embedded apparaten doorbreng, moet niet verloren gaan met het configureren en onderhouden van een ontwikkelomgeving. De ESP32-IDE met PlatformIO of de Arduino IDE bleken aan die eisen te voldoen. Computationeel was er meer headroom en veel microcontrollers werkten (Teensy 3+, RP2040, elk dinge met een Cortex-M). Een FPU zou kunnen helpen om het stroomverbruik laag te houden, vooral relevant als je met batterijen werkt.

Ik heb ook een paar M5StickC's van M5Stack gebruikt, omdat ze een eenvoudig te gebruiken package vormen en een geïntegreerde microfoon hebben: ik ben meer van de softwarekant dan van het 'hardcore hacken' van hardware, dus dat was een handige keuze. Ik heb ook een paar domotica-apparaten gebouwd (meer informatie op mijn website) en een paar andere projecten, zoals het regelen van de ventilatie in huis of het controleren van het niveau van de regenwatertank.

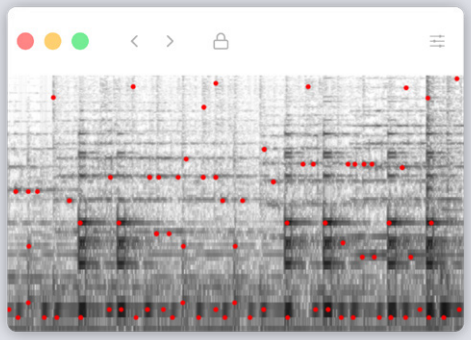
Tijdens het voorbereiden van dit artikel heb ik code voor een ESP32-demo en aanvullende documentatie toegevoegd aan mijn GitHub-repository [7]. Nu werkt de laatste versie van Olaf betrouwbaar op ESP32 met een goed verkrijgbare MEMS-microfoon. Op die helppagina vind je ook instructies over het gebruik van I²S-microfoons zoals de INMP441. Er is een demoprogramma dat audio van de microfoon via WiFi naar een computer stuurt, zodat je naar de microfoon kunt luisteren. Zo weet je dat de microfoon naar behoren werkt en dat de audiosamples correct worden geïnterpreteerd. Het valideert de I²S-instellingen, inclusief bufferomvang, sample rates, audioformaten en stereo of mono instelling.

Uitvoering in een webbrowser

Heb je ooit van WebAssembly [5] (of WASM)? WASM is een overdraagbaar binair codeformaat, samen met een bijbehorend tekstformaat, ontworpen voor uitvoerbare programma's. Het belangrijkste doel is om de ontwikkeling van krachtige applicaties binnen webpagina's te vergemakkelijken.

Het blijkt dat er manieren zijn om C-code te compileren naar WASM, zoals de Emscripten-compiler. Volgens de website stelt Emscripten je in staat om C en C++ code op het web te draaien op bijna native snelheid, helemaal zonder de noodzaak van plugins. Het combineren van de Web Audio API met de WASM-versie van Olaf opent mogelijkheden voor webgebaseerde akcoustic fingerprinting-toepassingen.

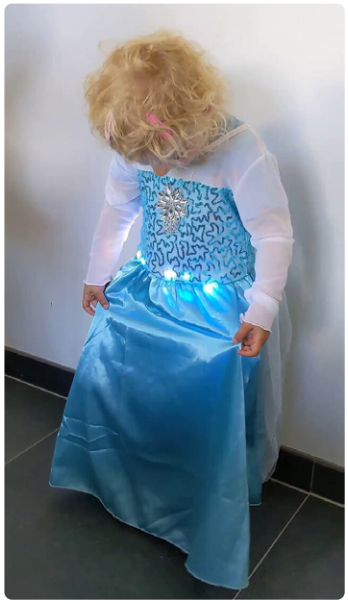
Op mijn blog kun je met Olaf experimenteren in je browser en mijn laatste berichten erover lezen. Precies dezelfde code die op de ESP32 draait, zoals eerder gedemonstreerd, draait nu in je browser. Dit betekent dat Olaf actief luistert om het liedje "Let It Go' van de Frozen-soundtrack te herkennen. Voor je gemak kun je links op je scherm het liedje afspelen vanuit de YouTube-video, en rechts kun je Olaf starten, zodat het binnenkomende audio kan analyseren. Olaf berekent de snelle fouriertransformatie (FFT) en visualiseert deze met behulp van Pixi.js. Na een paar seconden zouden de rode vingerafdrukken (zie screenshot) groen moeten worden, wat een succesvolle match aangeeft. Als je het nummer stopt, worden de vingerafdrukken uiteindelijk weer rood. Net als bij de eerder genoemde videodemonstratie duurt de overgang van een match naar geen match een paar seconden om rekening te houden met hiaten in de herkenning.



Er is nog een ander codevoorbeeld dat laat zien hoe je binnenkomende audio van een INMP441 microfoon kunt matchen met fingerprints die zijn opgeslagen in een headerbestand. In tegenstelling tot de PC versie gebruikt de embedded versie van Olaf geen key-value opslag, maar een lijst van hashes opgeslagen in het *src/olaf_fp_ref_mem.h* header-bestand, weat dient als de fingerprint-index.
Standaard is *src/olaf_fp_ref_mem.h* opgenomen in de ESP32-code. Om dit header-bestand te testen en te debuggen, kun je de mem-versie van Olaf opvragen op je computer: `bin/olaf query olaf_audio_your_audio_file.raw "arandomidentifier"`. De ESP32-versie is in essentie identiek aan de mem-versie, met als enige verschil dat de audio in de ESP32-versie afkomstig is van een MEMS-microfoon en niet uit een bestand.
Als je er zeker van bent dat de mem-versie van Olaf werkt zoals bedoeld en de INMP441-microfoon is getest, kan de ESP32-versie worden overgebracht naar de ESP32-hardware met behulp van de Arduino IDE.

Hoe gaat het verder?

Ik hoop dat dit artikel je op ideeën heeft gebracht voor projecten met akoestische herkenning. Het is altijd spannend om te zien hoe je, door te beginnen met een klein, speels project en het steeds verder te verfijnen, indrukwekkende resultaten kunt bereiken. In dit geval ging het project van een eerste prototype dat niet helemaal aan mijn verwachtingen voldeed naar een tweede prototype op basis van een ESP32, dat verschillende verbeteringen opleverde en ongetwijfeld veel plezier voor mijn dochter (**figuur 5**).



Figuur 5. Het kostuum met ESP32-lichteffecten.

Na verloop van tijd en door iteratieve verfijningen ontwikkelde het project zich tot een uitgebreide, overdraagbare, open-source, cross-platform, high-performance applicatie en bibliotheek. Ik heb twee academische artikelen over het onderwerp geschreven om ervoor te zorgen dat dit werk erkend en gewaardeerd kan worden binnen de onderzoeksgemeenschap. De vraag aan de lezers is nu: welke spannende projecten heb jij in gedachten om je ESP32 voor in te zetten?

230578-03

Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via joren.six@ugent.be of naar de redactie van Elektor via redactie@elektor.com.

Over de auteur

Joren Six is een computerwetenschapper die onderzoek doet op het gebied van muziekinformatica, Music Information Retrieval en computationele etnomusicologie. Hij behaalde een PhD aan de Universiteit Gent, België, en is momenteel betrokken bij verschillende projecten waarin IT en akoestiek worden gecombineerd.

Gerelateerde producten

- **JOY-iT NodeMCU ESP32 Development Board**
  www.elektor.nl/19973
- **ESP32-DevKitC-32E**
  www.elektor.nl/20518
- **ESP32-C3-DevKitM-1**
  www.elektor.nl/20324

WEBLINKS

[1] Eerste prototype van het project: https://0110.be/posts/Olaf_-_Acoustic_fingerprinting_on_the_ESP32_and_in_the_Browser
[2] Laatste updates van het project: https://0110.be/search?q=olaf
[3] De project-repository: https://github.com/JorenSix/Olaf
[4] Pretty Fast FFT-bibliotheek: https://bitbucket.org/jpommier/pffft/src/master/
[5] WebAssembly bij Wikipedia: https://en.wikipedia.org/wiki/WebAssembly
[6] Downloads bij dit artikel: https://0110.be/files/attachments/475/ESP32-Olaf.zip
[7] ESP32 Olaf-project op GitHub: https://github.com/JorenSix/Olaf/tree/master/ESP32