Tijdsynchronisatie

Patrick van Looy & Bram Leenders

23 mei 2014

1 Inleiding

Om energiezuinige draadloze communicatie mogelijk te maken, is er tijdsynchronisatie nodig tussen de verschillende nodes. Wanneer nodes gesynchroniseerd zijn kunnen ze op vaste momenten naar elkaar zenden en luisteren, en kunnen ze de rest van de tijd in een slaapstand zijn. Omdat communicatie relatief veel energie vraagt en een slaapstand zeer weinig, biedt synchronisatie dus mogelijkheden voor energiebesparing.

Tijdsynchronisatie kan op verschillende manieren geimplementeerd worden. In dit onderzoek gaan we een verschijnsel nabootsen wat we in de natuur ook kunnen vinden. De voorbeelden die hierbij gehanteerd worden, zijn vuurvliegjes en krekels. Vuurvliegjes, bijvoorbeeld, willen van nature synchroon knipperen[1]. Ditzelfde gedrag kan door Arduino's nagebootst worden, met behulp van radio- of geluidssignalen.

In dit onderzoek kijken we naar een specifiek algoritme, het firefly algoritme[2, 3]. Dit algoritme implementeren we zowel voor radiocommunicatie als voor communicatie met behulp van geluid. In sectie 2 geven we een kort overzicht van eisen waaraan het protocol dient te voldoen. Sectie 3 beschrijft de implementatie voor radiocommunicatie, en sectie 4 beschrijft dit voor synchronisatie met behulp van geluid. In sectie 5 wordt besproken hoe de beide implemenaties functioneren, en geven we een korte discussie over de beide manieren van synchronisatie.

2 Probleemstelling

Dit is een probleem!

Eisen:

- Nodes kunnen uit een groep verdwijnen zonder rest te beinvloeden.
- Nodes kunnen toegevoegd worden aan een groep, en het geheel synchroniseert.
- Twee groepen kunnen samengevoegd worden, en het geheel synchroniseert.
- Wanneer nodes uit synchronisatie raken, worden deze bijgesteld.
- Frequentie van communicatie ligt zo laag mogelijk.

3 Radiosynchronisatie

4 Tjirpende Arduino's

De implementatie beschreven in de vorige secties is niet afhankelijk van het precieze signaal dat de Arduino's geven. Het is alleen afhankelijk van het moment waarop het signaal uitgezonden en ontvangen wordt, en de tijd hiertussen mag niet exorbitant groot worden of wisselend lang en kort duren.

In plaats van een radiosignaal kunnen ook andere signalen uitgewisseld worden, bijvoorbeeld een geluidssignaal. De implementatie hiervan heeft wel wat meer voeten in de aarde, omdat er erg veel ruis is in de vorm van omgevingsgeluid. Tevens heeft de Arduino niet een standaardimplementatie die "pieken"kan detecteren; er is dus geen functie voor microfoons die vergelijkbaar is met `radio.available()`.

4.1 Analoge signaalverwerking

Een microfoon levert geen geschikt signaal op dat digitaal verwerkt kan worden. Het signaal is te zacht, bevat veel ruis en is analoog. Om het door de Arduino te laten verwerken moet het signaal versterkt worden en omgezet worden naar een digitaal signaal. Dit doen we in drie stappen:

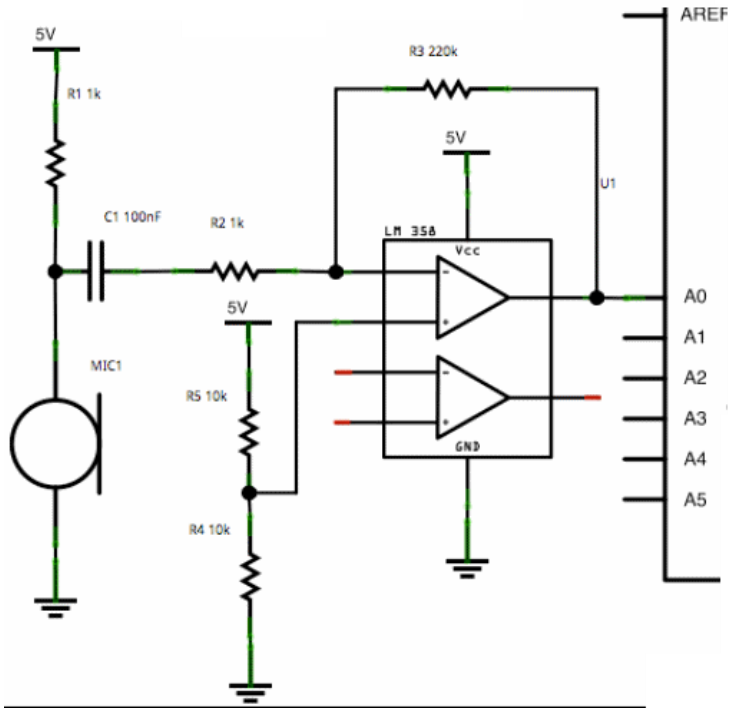
- *High-pass filter:* dit filter laat alleen de tonen boven een bepaalde ondergrens door, waardoor lage omgevingsgeluiden gefilterd worden. Dit vermindert dus de hoeveelheid ruis in het signaal.
- *Versterker:* omdat de microfoon een zwak signaal levert, moet het versterkt worden.
- *Omzetten naar digitaal signaal:* met behulp van een ADC (analog to digital converter) kan het gefilterde, versterke signaal omgezet worden naar een digitale input.

Het circuit dat voor dit onderzoek gebruikt is, staat in figuur 1. Hierbij wordt gebruik gemaakt van de ADC die standaard beschikbaar is op de Arduino Uno, die een analoog input signaal tussen 0 en 5 volt heeft en als digitale output een getal tussen de 0 en 1024 geeft.

4.2 Signaalverwerking

In figuur 2 is zichtbaar dat het versturen van een geluidssignaal een sterk wisselend inputsignaal geeft. We kunnen dus stellen dat als het verschil tussen twee opeenvolgende metingen erg verschilt, dat er dan zeer waarschijnlijk een signaal ontvangen wordt. Neem $v(t)$ de waarde gemeten op tijdstip t zijn, dan $|v(t) - v(t+1)| > \text{threshold} \Rightarrow$ signaal ontvangen.

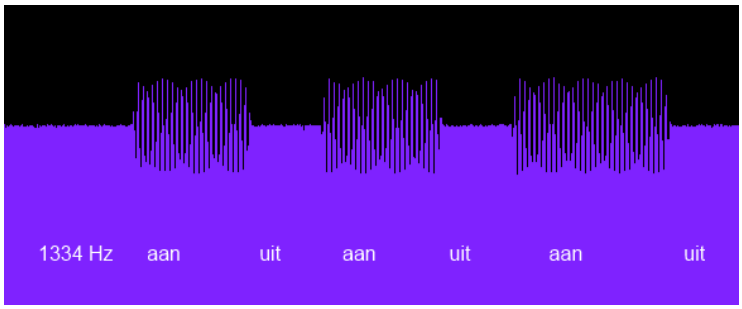
Hierbij is het van belang dat de threshold hoog genoeg gekozen wordt om ruis uit te sluiten, maar ook niet zo hoog dat signalen niet opgemerkt worden. Omdat dit moeilijk van tevoren vast te stellen is, hebben we gebruik gemaakt van een dynamische threshold gebaseerd op de gemiddelde afwijking. De gemiddelde afwijking (avgdiff) als functie van de tijd is



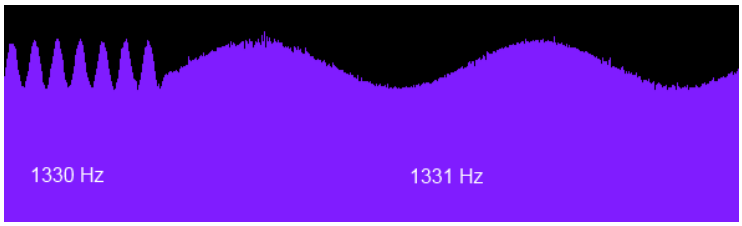
Figuur 1: High-pass filter met versterker en ADC.
Bron: CreaTe Protobox quick reference sheet.

$$\text{avgdiff}(t+1) = 0.1 \times |v(t) - v(t+1)| + 0.9 \times \text{avgdiff}(t)$$

Dit is dus een gemiddelde van de afwijkingen tussen metingen, waarbij het "gewicht"van een meting exponentieel snel afneemt. De eerste meting telt dus vrijwel niet mee, en de laatste meting relatief zwaar (10%). De threshold is $3 \cdot$ avgdiff: als metingen meer dan drie keer zoveel verschillen als gemiddeld, gaan we ervanuit dat er een signaal ontvangen wordt.



Figuur 2: Inkomend signaal bij pulserend signaal.



Figuur 3: Inkomend signaal bij verschillende geluidsfrequencies.

5 Resultaten en discussie

6 Conclusie

Referenties

[1] John Buck. Synchronous rhythmic flashing of fireflies. ii. *Quarterly Review of Biology*, pages 265–289, 1988.

[2] Robert Leidenfrost and Wilfried Elmenreich. Firefly clock synchronization in an 802.15. 4 wireless network. *EURASIP Journal on Embedded Systems*, 2009:7, 2009.

[3] Xin-She Yang and Xingshi He. Firefly algorithm: recent advances and applications. *International Journal of Swarm Intelligence*, 1(1):36–50, 2013.