

# Indoor positioning met Arduino's

Bram Leenders & Patrick van Looy

28 juni 2014

## 1 Inleiding

Voor veel verschillende situaties is het enorm handig om locatiebepaling geïmplementeerd te hebben op nodes zodat deze weten wat hun locatie is. Denk bijvoorbeeld aan een sensornetwerk in het bos, bedoelt om bosbranden te detecteren. Het zou hierbij wel handig zijn dat als een node een brand constateert, het ook gelijk door kan geven waar precies de brand is ontstaan. Op deze manier kan de brand doelgericht en snel bestreden worden.

Locatiebepaling kan op verschillende manieren gedaan worden, bijvoorbeeld door het gebruik van GPS. Dit onderzoek legt vooral de focus op locatiebepaling door middel van hoogfrequent geluid. Het doel van dit onderzoek is om de locatie van een Arduino met een ultrasoon-ontvanger te bepalen ten opzichte van vier ultrasoon-verzenders die zich op bekende posities bevinden en afwisselend een geluidspuls versturen. Hiervoor moet een lokalisatiealgoritme ontwikkeld worden dat gebruik maakt van de signalen die verzonden worden door deze vier bakens.

Allereerst zullen we een probleemstelling formuleren zodat we een uitgangspunt voor onze tests hebben, dit doen we in sectie 2. In sectie 3 worden drie mogelijke manieren van afstandbepaling toegelicht. Daarnaast wordt in sectie 4 de implementatie beschreven. Vervolgens behandelen we in sectie 5 de resultaten die we door middel van onze tests hebben gekregen. Als laatste trekken we hieruit een conclusie in sectie 6.

## 2 Probleemstelling

## 3 Gerelateerd werk

Er zijn verschillende manieren om met behulp van radio en/of geluidssignalen een afstand te meten, we zullen de volgende drie kort toelichten:

- Received Signal Strength Indication (RSSI)
- Time Difference of Arrival (TDOA)
- Time of Flight (TOF)

Deze drie zijn met de beschikbare hardware (Arduino, RF24 chip en microfoon) implementeerbaar, dus er moet een keuze uit deze drie gemaakt worden.

### 3.1 Received Signal Strength Indication

Bij RSSI wordt de sterkte van het signaal gebruikt om een schatting te maken van de afstand tussen een zender en een ontvanger. Deze methode heeft een aantal nadelen, zoals beschreven door Seshadi et al. [4]. De belangrijkste nadelen zijn de wisselende signaalsterkte, kosten van meet- en zendapparatuur en verstoringen van objecten tussen zender en ontvanger. Vanwege deze redenen hebben we niet voor RSSI als methode gekozen.

### 3.2 Time Difference of Arrival

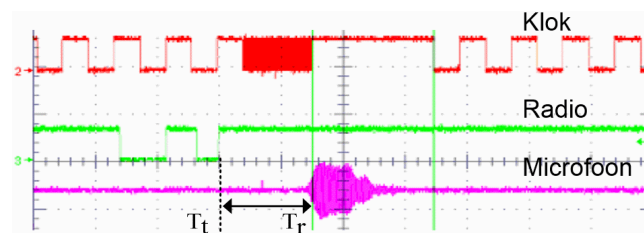
Bij TDOA wordt gebruik gemaakt van het verschil in afstand tussen twee zenders. Als twee zenders tegelijkertijd een signaal uitzenden, kan een ontvanger een mogelijk verschil in ontvangsttijd meten. Dit verschil in ontvangsttijd kan dan omgezet worden naar een verschil in afstand tussen de twee zenders. Deze methode wordt verder uitgewerkt door Gustaffson et al. [2].

### 3.3 Time of Flight

Voor het onderzoek van deze paper is de time of flight (TOF) gebruikt om de afstand tussen beacons en de ontvanger te meten. Deze methode maakt ook gebruik van het verschil in ontvangsttijd van twee signalen. Echter, niet tussen signalen van twee nodes, maar tussen twee types signalen: radio en geluidssignalen.

TOF maakt gebruik van het verschil in propagatiesnelheid van licht en geluid; radiosignalen gaan met lichtsnelheid (ca.  $3 \cdot 10^8$  m/s), maar geluid gaat veel langzamer (ca 340 m/s). Door beacons tegelijkertijd een radio- en een geluidssignaal uit te laten zenden kan met behulp van het verschil in ontvangsttijden de afstand tussen het beacon en een ontvanger berekend worden. Deze techniek wordt beschreven door Barshan en Ballur [1].

Figuur 1 geeft een voorbeeld van inkomende signalen bij een dergelijke aanpak.



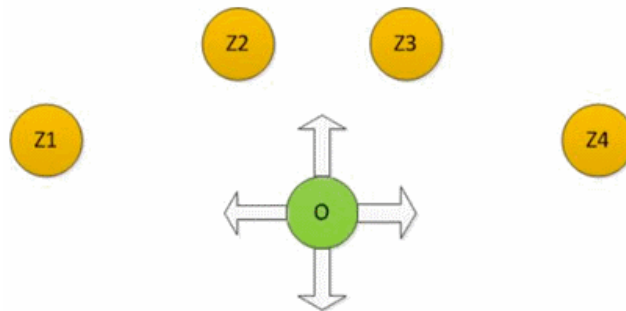
**Figuur 1:** Tijdsdiagram radio en microfoon input. *Bron: [3]*

## 4 Implementatie

De implementatie gebruikt de time of flight (TOF) om de afstand tussen beacons en de ontvanger te meten.

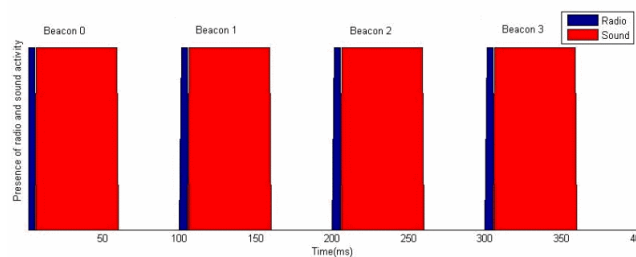
## 4.1 Opstelling voor positiebepaling met hoogfrequent geluid

De opstelling voor het bepalen van de positie bestaat uit vier bakens, die zijn genummerd van 0 tot 3. Elk baken bestaat uit een Arduino mini met daarop aangesloten een NRF2401L+-radio en een ultrasoon-zender. De bakens zijn bevestigd op een statief. De locaties van de bakens is bekend. Een voorbeeld van een opstelling met de vier bakens en een ontvanger is te zien in afbeelding 2.



**Figuur 2:** Opstelling met vier zenders en een ontvanger.

De activiteiten van de verschillende bakens zijn als volgt: een van de bakens (Baken 0) verstuurt radioberichten met een interval van 100 ms. Dit bericht bestaat uit een uint8 (een getal tussen 0 en 255) waarin het nummer staat van de baken die aan de beurt is voor het versturen van een geluidspuls. (Baken 0 stuurt ook een bericht als baken 0 zelf aan de beurt is.) Als een baken een bericht ontvangt waarin zijn identificatienummer staat, verstuurt deze direct hierna een geluidspuls op een frequentie van circa 40kHz en met een duur van 50 ms. De bakens zijn dus nooit tegelijkertijd actief. Figuur 3 toont de sequentie van activiteiten van de verschillende bakens. Verder zijn in tabel 1 de verschillende instellingen te zien waarop de radios onderling communiceren.



**Figuur 3:** Opstelling met vier zenders en een ontvanger.

Kanaal	76 (standaard RF24 instelling)
Automatisch herverzenden	Uit
Transmissiesnelheid	2 Mbps
Adres verzendende pipe	0xdeadbeefa1LL
Payload-grootte	1 byte

**Tabel 1:** Instellingen radio

## 5 Resultaten en discussie

...

## 6 Conclusie

...

## Referenties

- [1] Billur Barshan. Fast processing techniques for accurate ultrasonic range measurements. *Measurement Science and technology*, 11(1):45, 2000.
- [2] Fredrik Gustafsson and Fredrik Gunnarsson. Positioning using time-difference of arrival measurements. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 6, pages VI–553. IEEE, 2003.
- [3] Jaehyun Park, Sunghee Choi, and Jangmyung Lee. Beacon scheduling algorithm for localization of a mobile robot. In *Intelligent Robotics and Applications*, pages 594–603. Springer, 2011.
- [4] Vinay Seshadri, Gergely V Zaruba, and Manfred Huber. A bayesian sampling approach to in-door localization of wireless devices using received signal strength indication. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 75–84. IEEE, 2005.

## A Bijlage 1 - Code

```
/*
   Positioning system for Arduino One with RF24 radio chip
*/
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"
#include "MatrixMath.h"

#define N (3)
// Kunstmatige waarde voor Z coördinaten
#define Z 1.0
// Percentage verschil (0 < MAX_DIFF <= 1) dat tussen twee
// metingen mag zitten.
#define MAX_DIFF (0.25)
// Percentage dat de nieuwste meting in het gemiddelde meetelt (0
// < WEIGHT <= 1)
// Bij WEIGHT=1 wordt er geen gemiddelde bijgehouden, maar is de
// nieuwste meting de enige die meetelt.
#define WEIGHT (0.2)

RF24 radio(3, 9);
unsigned long radiotime;
unsigned long audiotime;
unsigned long timelimit = 50000LL;
uint8_t activeBeacon;

float pos[4][2] = { // Positions van de beacons; pos[1][1] is de y
                    // positie van beacon 1
                    {0.0, 75.0},
                    {72.0, 0.0},
                    {294.0, 0.0},
                    {372.0, 136.0}
};

float D[4];

void setup() {
    // initialize the serial communication:
    Serial.begin(9600);
    printf_begin();

    // Setup and configure rf radio
    radio.begin();
    radio.setRetries(0,0);

    radio.setDataRate(RF24_2MBPS);
    radio.setChannel(76);
    radio.setPayloadSize(1);
    radio.openReadingPipe(1, 0xdeadbeefa1LL);
    radio.openWritingPipe(0xdeadbeefa1LL);
    radio.startListening();
    radio.setAutoAck(false);
}

void loop() {
    while(radio.available()) {
        radio.read(&activeBeacon, sizeof(uint8_t));
    }
}
```

```

while (! radio.available());

radiotime = micros();
radio.read( &activeBeacon, sizeof(uint8_t));

if(activeBeacon > 3) { return; }

while(analogRead(A0) < 50) {
    audiotime = micros();
    if(audiotime - radiotime > timelimit) {
        return;
    }
}

float diff = audiotime - radiotime;
diff = diff * 0.03432; // Afstand tot beacon in cm

//Zwak uitschieters een beetje af: max 30% increase
if(diff > (D[activeBeacon]* (1.0 + MAX_DIFF)) && D[activeBeacon]
    > 0) {
    diff = D[activeBeacon] * (1.0 + MAX_DIFF);
}

if(diff < (D[activeBeacon]* (1.0 - MAX_DIFF))) {
    diff = D[activeBeacon]*(1.0 - MAX_DIFF);
}

D[activeBeacon] = D[activeBeacon]*(1.0 - WEIGHT) + diff*WEIGHT;
// Weer schuivend gemiddelde */
//D[activeBeacon] = diff;

if(activeBeacon == 3) {
    calcPosition();
}
}

float A[N][N];
float B[N];

void calcPosition() {
    // Relatieve afstanden tussen de nodes; gebruikt node 3 nog
    // niet!
    A[0][0] = 2*pos[1][0] - 2*pos[0][0]; A[0][1] = 2*pos[1][1] -
        2*pos[0][1]; A[0][2] = Z;
    A[1][0] = 2*pos[2][0] - 2*pos[1][0]; A[1][1] = 2*pos[2][1] -
        2*pos[1][1]; A[1][2] = Z;
    A[2][0] = 2*pos[0][0] - 2*pos[2][0]; A[2][1] = 2*pos[0][1] -
        2*pos[2][1]; A[2][2] = Z;
    Matrix.Invert((float*)A,N);

    B[0] = (D[0]*D[0]) - (D[1]*D[1]) - (pos[0][0]*pos[0][0]) +
        (pos[1][0]*pos[1][0]) - (pos[0][1]*pos[0][1]) +
        (pos[1][1]*pos[1][1]);
    B[1] = (D[1]*D[1]) - (D[2]*D[2]) - (pos[1][0]*pos[1][0]) +
        (pos[2][0]*pos[2][0]) - (pos[1][1]*pos[1][1]) +
        (pos[2][1]*pos[2][1]);
    B[2] = (D[2]*D[2]) - (D[0]*D[0]) - (pos[2][0]*pos[2][0]) +
        (pos[0][0]*pos[0][0]) - (pos[2][1]*pos[2][1]) +
        (pos[0][1]*pos[0][1]);

    float P3[N];
    Matrix.Multiply((float*)A,(float*)B,N,N,1,(float*)P3);
}

```

```

printf("Position (P3): (%d,%d)\n", (int) P3[0], (int) P3[1]);

// Relatieve afstanden tussen de nodes; gebruikt node 2 nog
niet!
A[0][0] = 2*pos[1][0] - 2*pos[0][0]; A[0][1] = 2*pos[1][1] -
2*pos[0][1]; A[0][2] = Z;
A[1][0] = 2*pos[3][0] - 2*pos[1][0]; A[1][1] = 2*pos[3][1] -
2*pos[1][1]; A[1][2] = Z;
A[2][0] = 2*pos[0][0] - 2*pos[3][0]; A[2][1] = 2*pos[0][1] -
2*pos[3][1]; A[2][2] = Z;
Matrix.Invert((float*)A,N);

B[0] = (D[0]*D[0]) - (D[1]*D[1]) - (pos[0][0]*pos[0][0]) +
(pos[1][0]*pos[1][0]) - (pos[0][1]*pos[0][1]) +
(pos[1][1]*pos[1][1]);
B[1] = (D[1]*D[1]) - (D[3]*D[3]) - (pos[1][0]*pos[1][0]) +
(pos[3][0]*pos[3][0]) - (pos[1][1]*pos[1][1]) +
(pos[3][1]*pos[3][1]);
B[2] = (D[3]*D[3]) - (D[0]*D[0]) - (pos[3][0]*pos[3][0]) +
(pos[0][0]*pos[0][0]) - (pos[3][1]*pos[3][1]) +
(pos[0][1]*pos[0][1]);
float P2[N];
Matrix.Multiply((float*)A,(float*)B,N,N,1,(float*)P2);
printf("Position (P2): (%d,%d)\n", (int) P2[0], (int) P2[1]);

// Relatieve afstanden tussen de nodes; gebruikt node 1 nog
niet!
A[0][0] = 2*pos[2][0] - 2*pos[0][0]; A[0][1] = 2*pos[2][1] -
2*pos[0][1]; A[0][2] = Z;
A[1][0] = 2*pos[3][0] - 2*pos[2][0]; A[1][1] = 2*pos[3][1] -
2*pos[2][1]; A[1][2] = Z;
A[2][0] = 2*pos[0][0] - 2*pos[3][0]; A[2][1] = 2*pos[0][1] -
2*pos[3][1]; A[2][2] = Z;
Matrix.Invert((float*)A,N);

B[0] = (D[0]*D[0]) - (D[2]*D[2]) - (pos[0][0]*pos[0][0]) +
(pos[2][0]*pos[2][0]) - (pos[0][1]*pos[0][1]) +
(pos[2][1]*pos[2][1]);
B[1] = (D[2]*D[2]) - (D[3]*D[3]) - (pos[2][0]*pos[2][0]) +
(pos[3][0]*pos[3][0]) - (pos[2][1]*pos[2][1]) +
(pos[3][1]*pos[3][1]);
B[2] = (D[3]*D[3]) - (D[0]*D[0]) - (pos[3][0]*pos[3][0]) +
(pos[0][0]*pos[0][0]) - (pos[3][1]*pos[3][1]) +
(pos[0][1]*pos[0][1]);
float P1[N];
Matrix.Multiply((float*)A,(float*)B,N,N,1,(float*)P1);
printf("Position (P1): (%d,%d)\n", (int) P1[0], (int) P1[1]);

// Relatieve afstanden tussen de nodes; gebruikt node 0 nog
niet!
A[0][0] = 2*pos[2][0] - 2*pos[1][0]; A[0][1] = 2*pos[2][1] -
2*pos[1][1]; A[0][2] = Z;
A[1][0] = 2*pos[3][0] - 2*pos[2][0]; A[1][1] = 2*pos[3][1] -
2*pos[2][1]; A[1][2] = Z;
A[2][0] = 2*pos[1][0] - 2*pos[3][0]; A[2][1] = 2*pos[1][1] -
2*pos[3][1]; A[2][2] = Z;
Matrix.Invert((float*)A,N);

```

```

B[0] = (D[1]*D[1]) - (D[2]*D[2]) - (pos[1][0]*pos[1][0]) +
      (pos[2][0]*pos[2][0]) - (pos[1][1]*pos[1][1]) +
      (pos[2][1]*pos[2][1]);
B[1] = (D[2]*D[2]) - (D[3]*D[3]) - (pos[2][0]*pos[2][0]) +
      (pos[3][0]*pos[3][0]) - (pos[2][1]*pos[2][1]) +
      (pos[3][1]*pos[3][1]);
B[2] = (D[3]*D[3]) - (D[1]*D[1]) - (pos[3][0]*pos[3][0]) +
      (pos[1][0]*pos[1][0]) - (pos[3][1]*pos[3][1]) +
      (pos[1][1]*pos[1][1]);
float P0[N];
Matrix.Multiply((float*)A,(float*)B,N,N,1,(float*)P0);
printf("Position (P0): (%d,%d)\n", (int) P0[0], (int) P0[1]);

// Bereken het gemiddelde van de verschillende metingen:
int avg[N];
avg[0] = (int) (P0[0] + P1[0] + P2[0] + P3[0]) / 4.0;
avg[1] = (int) (P0[1] + P1[1] + P2[1] + P3[1]) / 4.0;
avg[2] = (int) (P0[2] + P1[2] + P2[2] + P3[2]) / 4.0;

printf("Position: (%d,%d)\n\n", avg[0], avg[1]);
}

```