Indoor positioning met Arduino's

Bram Leenders & Patrick van Looy

18 juni 2014

1 Inleiding

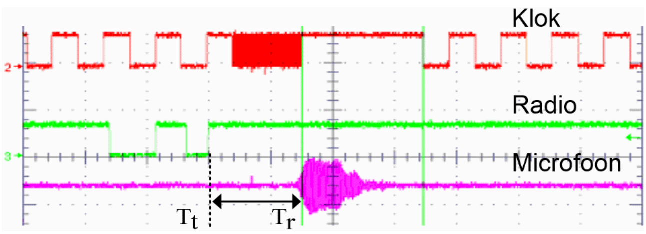
...

2 Gerelateerd werk

...

3 Implementatie

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



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

4 Resultaten en discussie

...

5 Conclusie

...

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 coordinaten
#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]);
}
```

Referenties

[1] 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.