Autonome Mobiele Robots

NXT - Particle filter based simultaneous localization and mapping

Auteurs: Tom Peerdeman & René Aparicio Saez

Datum: December 16, 2013

Contents

Tom Peerdeman - 10266186 René Aparicio Saéz - 10214054

1 Materiaal

Om de experimenten uit dit rapport te kunnen uitvoeren zijn de volgende materialen gebruikt:
- PC/Laptop met Matlab
- Boek: Autonomous Mobile Robots 2th Edition - Roland Siegwart et al.
- NXT-Robot
- Logitech Webcam
- Gloeilamp
- Zwarte tape

2 Introduction

Een autonome mobiele robots moet een kaart kunnen opbouwen van zijn omgeving. Aan de hand van zijn eigen gemaakte kaart zou hij met behulp van bekende kaarten kunnen bepalen waar hij zich in de wereld bevindt. Een veel gebruikte methode om een kaart al rijdend op te bouwen is SLAM (Simultaneous Localization And Mapping). Er kan dan met een bepaalde zekerheid bepaald worden waar de robot zich momenteel bevindt. Het is de bedoeling dat de robot in een gebied rond kan rijden en hierbij een goede kaart kan maken. Zo moet hij bijvoorbeeld na het rijden van een rondje weer hetzelfde deel van de kaart zien (mits er niks veranderd is aan de omgeving).

3 Feature detection

De gebruikte versie van SLAM, fastSLAM, maakt gebruik van ruwe odometrie data en features welke gedetecteerd worden door de laserscan. Als feature zal gebruik gemaakt worden van hoeken van lijnen. Om de corners te kunnen vinden zullen eerst de lijnen gevonden moeten worden uit de ruwe punten data die de laserscan aanlevert. Om deze lijnen te kunnen detecteren wordt gebruik gemaakt van het split & merge algoritme. Het split en merge algoritme is in stappen uitgelegd in listing 1.

Listing 1: Split & merge algorithm

```
1 Initially: set s1 consists of all N points. Insert s1 to the list L. Set index i=1
2 Fit a line to the next set si in L
3 Detect the point P with the maximum distance D to the line
4 if D is less than a threshold then continue to step 2
5 else split si at P into si1 and si2, replace si , in L, by si1 and si2. Continue to step 2
6 When all sets (segments) in L have been checked, merge collinear segments.
```

Het fitten van een lijn wordt hierbij gedaan door het zoeken naar een lijn zodanig dat de som van de afstanden in het kwadraat van de punten tot de lijn minimaal is. De lijn wordt hierbij uitgedrukt in polaire coördinaten r en α (zie figuur 1). Dit is het geval wanneer de formule $D^2 = \sum_{i=1}^{n} r - x_i cos(\alpha) - y_i sin(\alpha)$ minimaal is. Om de waardes van r en α te kunnen vinden kan gebruik gemaakt worden van de afgeleiden, $\frac{d(D^2)}{dr} = 0$ en $\frac{d(D^2)}{d\alpha} = 0$. Hieruit worden de volgende vergelijkingen verkregen:

$$nom = -2 \sum_{i=1}^{n} (x_i - x_c)(y_i - y_c)$$

$$dennom = \sum_{i=1}^{n} (y_i - y_c)^2 - (x_i - x_c)^2$$

$$\alpha = 0.5 \text{ atan2}(nom, denom)$$

Hierbij zijn x_c en y_c de coördinaten van het middelpunt van alle punten. Om deze coördinaten te vinden wordt simpelweg het gemiddelde van de x en het gemiddelde van de y waarden van alle punten genomen. Aan de hand van de α parameter kan de r parameter worden gevonden. Het is namelijk zo dat een punt op een lijn uitgedrukt in polaire coördinaten voldoet aan de vergelijking $xcos(\alpha) + ysin(\alpha) = r$. De r parameter kan worden gevonden als we van een punt zeker weten dat

Tom Peerdeman - 10266186 René Aparicio Saéz - 10214054

het op de lijn ligt. Dit punt valt niet exact te bepalen, daarom is gekozen om gebruik te maken van de gemiddelde waardes van x en y om daaruit de waarde voor r te verkrijgen. De gemiddelde waardes zullen vrij dicht in de buurt komen van een punt op de lijn.

Als nu naar de punten wordt gekeken die het verste op de lijn liggen kunnen de uiteindes van dit lijnsegment bepalen worden bepaald. Het wordt nu ook makkelijker om hoeken te zoeken door alleen naar deze uiteindes te kijken en te bekijken of ze een hoek vormen met een ander lijnsegment.

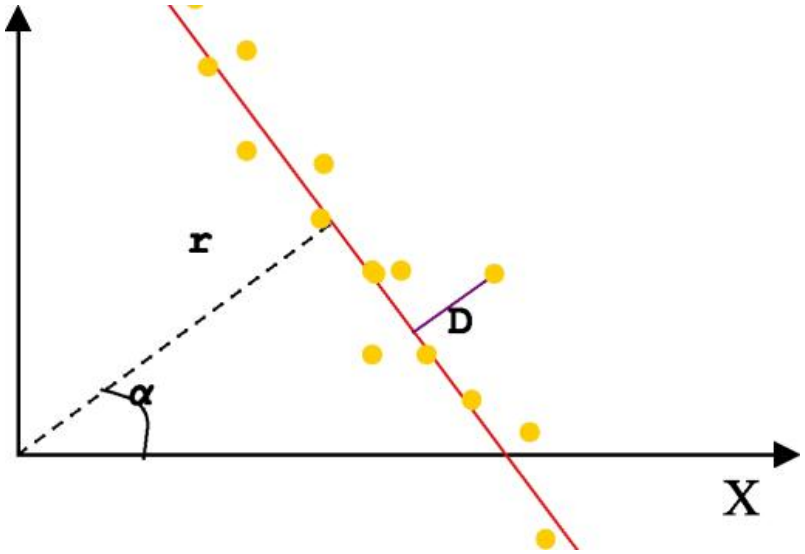

Figure 1: Fitten van de lijn door de som van D kwadraat te minimaliseren

4 fastSLAM algoritme

Het fastSLAM algoritme maakt gebruik van particles. Hierbij kan elk particle voorgesteld worden als een kans dat de robot aanwezig is op de positie van de robot, de stand van de robot maakt hierbij ook deel uit van de positie. Het algoritme werkt met twee stappen.
Stap 1 is de odometrie update. In deze stap wordt gekeken naar de verplaatsing van de robot. Aan de hand van de encoderwaardes kan worden uitgerekend welke afstand gereden is. Aangezien de kans op een verschil tussen de gemeten waarde en de echt afgelegde afstand groeit kan worden voorspeld dat de particles zich met een bepaalde error zullen voortbewegen. Er wordt aangenomen dat deze error gaussisch is. De nieuwe positie van een particle kan dan worden voorspeld met de vergelijking

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}^{t+1} = \begin{bmatrix} (x_t + dr_n cos(dth_n + \theta_t)) \\ (y_t + dr_n sin(dth_n + \theta_t)) \\ pi2pi(pi2pi(\theta) + dth) \end{bmatrix}$$

Hierbij is $pi2pi$ een functie die hoeken omzet van de range $[02\pi]$ naar $[0, pi], (-pi, 0]$. dth_n is de toegevoegde noise voor de afgelegde hoek dth. dr_n is de toegevoegde noise voor de afgelegde afstand.
Het is belangrijk te vermelden dat er meerdere odometrie updates kunnen voorkomen voordat stap 2 uitgevoerd wordt. Hierdoor kan de error in de particle positie vrij hoog oplopen. Deze error moet dat naar beneden gebracht worden door de data uit de laserscan (stap 2).

Stap 2 is de laserscan update. In de laserscan update wordt de error verkleind door een aantal particles weg te gooien. De particles die worden weggegooit hebben daarbij de kleinste kans dat de robot op de positie van deze particles is. Dit wordt gedaan door de ruwe laserscan data om te zetten in features (hoeken in dit geval) zoals is uitgewerkt in sectie 3. Als de gedetecteerde features al eens eerder gedetecteerd waren kunnen een aantal particles worden weggegooit aangezien

de positie van deze features al bekend is. Door voor elk particle te voorspellen waar de feature zich zou moeten bevinden als de robot op die positie zou zijn, en dit te vergelijken met de gedetecteerde positie kan elke particle een gewicht krijgen. Dit gewicht is hoger naar mate de gedetecteerde positie meer overeenkomt met de voorspelde positie. Aan de hand van deze weights gaat het algoritme de particle set resamplen. De particles met de laagste weights worden hierbij verwijderd. Aangezien de overgebleven particles weer een error met een bepaalde distributie voorstellen kunnen uit deze distibutie nieuwe posities worden getrokken om zo het aantal particles gelijk te houden aan het aantal van voor de resampling.

5 Schatten van SLAM parameters

SLAM maakt gebruik van enkele parameters. Deze parameters moeten worden geschat, om het beste resultaat te verkrijgen. Met behulp van een meegeleverde test log, kunnen enkele experimenten met verschillende waarden voor deze parameters worden gedaan. De te schatten parameters zijn, $NPARTICLES$, $sigmaX$, $sigmaTH$, $sigmaR$ en $sigmaB$. $NPARTICLES$ geeft aan hoeveel deeltjes (locaties) er moeten worden weergegeven waar de robot momenteel kan zijn. $sigmaX$ en $sigmaTH$ geven de variantie in de odometrie aan, de afwijking van de gegeven waarden voor respectievelijk de gereden afstand en de hoek. $sigmaR$ en $sigmaB$ geven de variantie aan die de laserscan als afstanden tot lijnen teruggeeft (de rangesensor).

5.1 Experimenten

In figuur 5, figuur 6 en figuur 7[1] zijn de resultaten te zien van de gemaakt experimenten. Duidelijk is te zien dat met het verhogen van het aantal $NPARTICLES$ de dikte van de lijn gestaag toeneemt. Door $NPARTICLES = 200$ te kiezen wordt ervoor gezorgd dat de lijn niet te dun wordt en niet te dik.
Zoals te verwachten viel, is met een lage variantie in de odometrie minder ruis voor de gedetecteerde muren zichtbaar. Deze variantie moet dus laag gehouden worden om nog een duidelijke kaart te kunnen opbouwen. Een variantie van $sigmaX = 0.00003m$ en $sigmaTH = 0.0002°$ zorgt voor een duidelijk resultaat.
Opvallend is het extreem dikker worden van de locatie van de robot zodra een lage variantie wordt gekozen voor de rangesensor. De rangesensor is blijkbaar niet gevoelig genoeg om met grote zekerheid de werkelijke afstand terug te geven, dus moet een grotere waarde voor de variantie worden gekozen. $sigmaR = 0.7$ en $sigmaB = 0.7$ levert een niet al te dikke locatielijn op, en een duidelijk resultaat voor de locatie van de muren.

6 Deel 2, Eigen data

6.1 Offline SLAM met behulp van een dataset

Er moet worden gewerkt met een offline dataset, in verband met tijdgebrek en het niet aanwezig zijn van voldoende robots. De meegeleverde dataset bestaat uit 61 fotos. Voordat begonnen kan worden met het opbouwen van een kaart moeten eerst de parameters voor de camera worden gecalibreerd. De fotos bevatten echter twee verschillende centers. Na het 40e plaatje veranderd de center enigszins (Picture 51.jpg tot het einde van de dataset). De centers zijn respectievelijk $centre_1 = [465; 343]$ voor de eerste 40 fotos en $centre_2 = [465; 335]$ voor de overige fotos. Alle fotos maken gebruik van $\alpha = 140$ en $Rmin = 100$ en $Rmax = 180$. Er is gebruik gemaakt van een BWtreshold van 100, behalve voor Picture 17.jpg tot Picture 22.jpg. Hierbij is de treshold verlaagd naar 90. Dit is gedaan omdat bij deze fotos de draad die de camera met de laptop verbind gemeten wordt. De draad beinvloed hiermee de data. In figuur 2 is te zien welke route gereden is. Deze kaart met bijbehorende route zal ongeveer overeen moeten komen met de opgebouwde kaart door het SLAM algoritme.

[1] deze figuren staan aan het einde van dit labreport, i.v.m. structuur behoudt van het labreport

Tom Peerdeman - 10266186 René Aparicio Saéz - 10214054

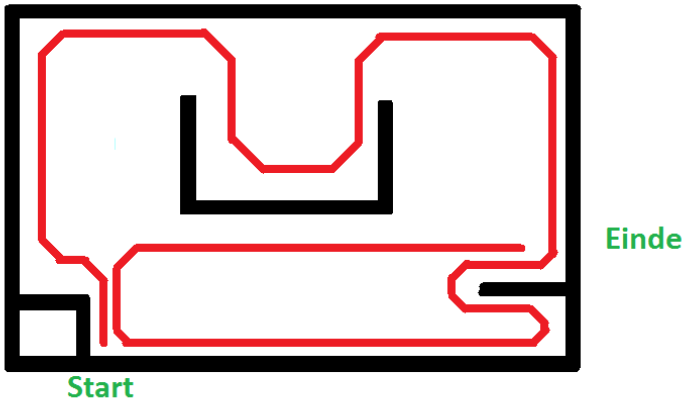

Figure 2: Afgelegde route

6.2 Odometrie bepalen

Omdat er gebruik gemaakt wordt van een dataset van fotos en niet van de meegeleverde GUI, moet de odometrie met de hand worden gemaakt. Aan de hand van de meegegeven layout, te zien in figuur 3, en het bekijken van de locatie van de robot aan de hand van de fotos kan een schatting gemaakt worden van de afgelegde afstanden. Deze schatting kan aan de hand van experimenten

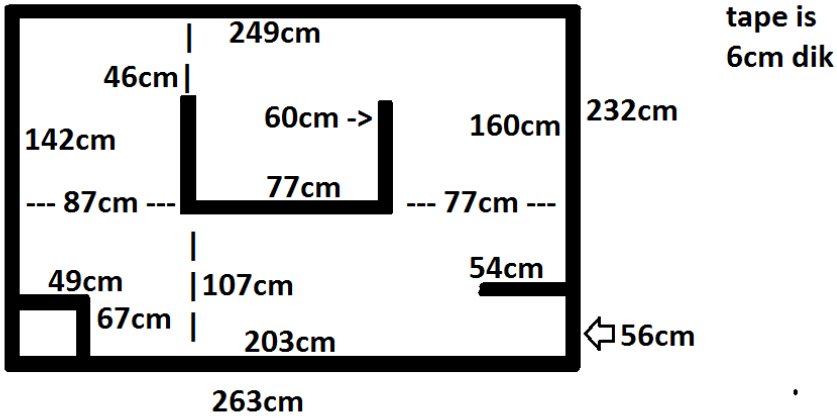

Figure 3: Layout van de kaart

worden geoptimaliseerd. Deze optimalisatie wordt gedaan door afgelegde afstanden te vergroten of te verkleinen. In een enkel geval moet de draaihoek worden aangepast. De meeste bochten die gemaakt worden bestaan uit twee korte bochten van elk 45°. Er zijn echter twee bochten waarbij eerst een draaiing wordt gemaakt van 60° gevolgd door een draaiing van 30°.

6.3 Resultaten

Het resultaat met geoptimaliseerde odometrie is zichtbaar in figuur 4. Er is duidelijk zichtbaar dat de gevonden kaart grotendeels overeenkomt met de layout uit figuur 2. De gemaakte kaart is een kwartslag gedraaid, maar dit maakt voor de metingen niet uit. De gevonden afstanden voor de lengtes van de lijnen klopt in grote lijnen.

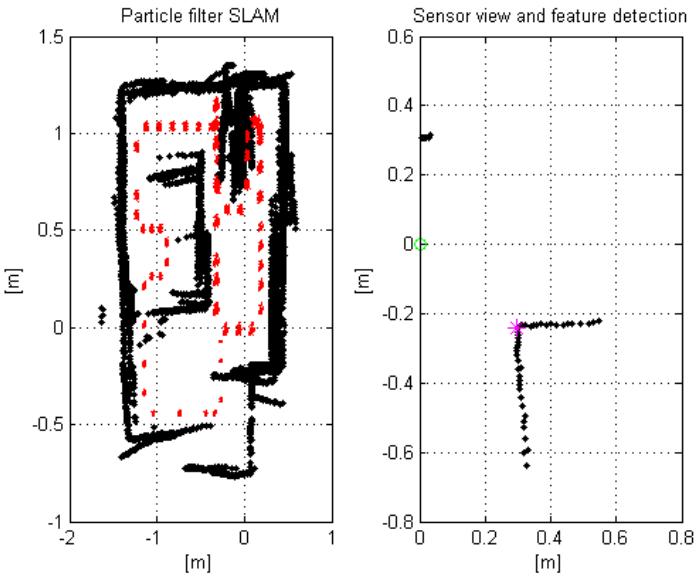

Figure 4: Uiteindelijk resultaat

6.4 Discussie

Er is ruis zichtbaar in het resultaat. Dit heeft te maken met het mogelijk nog meer te optimaliseren van de odometrie of door het optimaliseren van de parameters. Omdat de dataset twee verschillende centers heeft, ontstaat ruis vanaf plaatje 41. Dit zorgt voor meer ruis (de dikkere muur) rechtsboven in figuur 4.
De gevonden afstanden kloppen grotendeels. Ze zijn niet volledig accuraat vanwege meerdere factoren. Zo speelt de ruis een grote rol, maar ook de omzetting van het gebolde beeld naar een 2D-beeld zorgt voor een significante afwijking. Omdat de bolling van de gebruikte gloeilamp niet een perfecte bolling is, is het moeiilijk om een goede alpha te kiezen waarbij de werkelijke afstand klopt ten opzichte van de pixelwaarden, zonder dat het beeld vervormd raakt. Dit zorgt voor een kleine afwijking in werkelijke afstanden.

6.5 Verbeteringen

De odometrie houdt momenteel alleen rekening met de afgelegde afstand in de huidige rijrichting van de robot. Het is hierdoor niet mogelijk om met behulp van fotos een daadwerkelijke bocht aan te geven. Het algoritme denkt nu dat er een rechte lijn wordt gereden en vervolgens wordt gedraaid, in plaats van een continue draaiing over de af te leggen afstand uit te voeren. Daarnaast zullen er betere resultaten worden verkregen als er per locatie meerdere metingen worden gemaakt en de afstand tussen de metingen verkleind wordt. Daar komt bij dat tijdens het maken van de dataset het aan te raden is de afgelegde afstanden en draaihoeken worden opgeschreven, zodat deze niet achteraf geschat hoeven te worden. Tot slot zou een verbeterde omni-directionele camera, die op een juiste manier de pixelafstanden om kan zetten naar werkelijke afstanden een uitkomst bieden op betere resultaten.

7 Parameter experimenten figuren

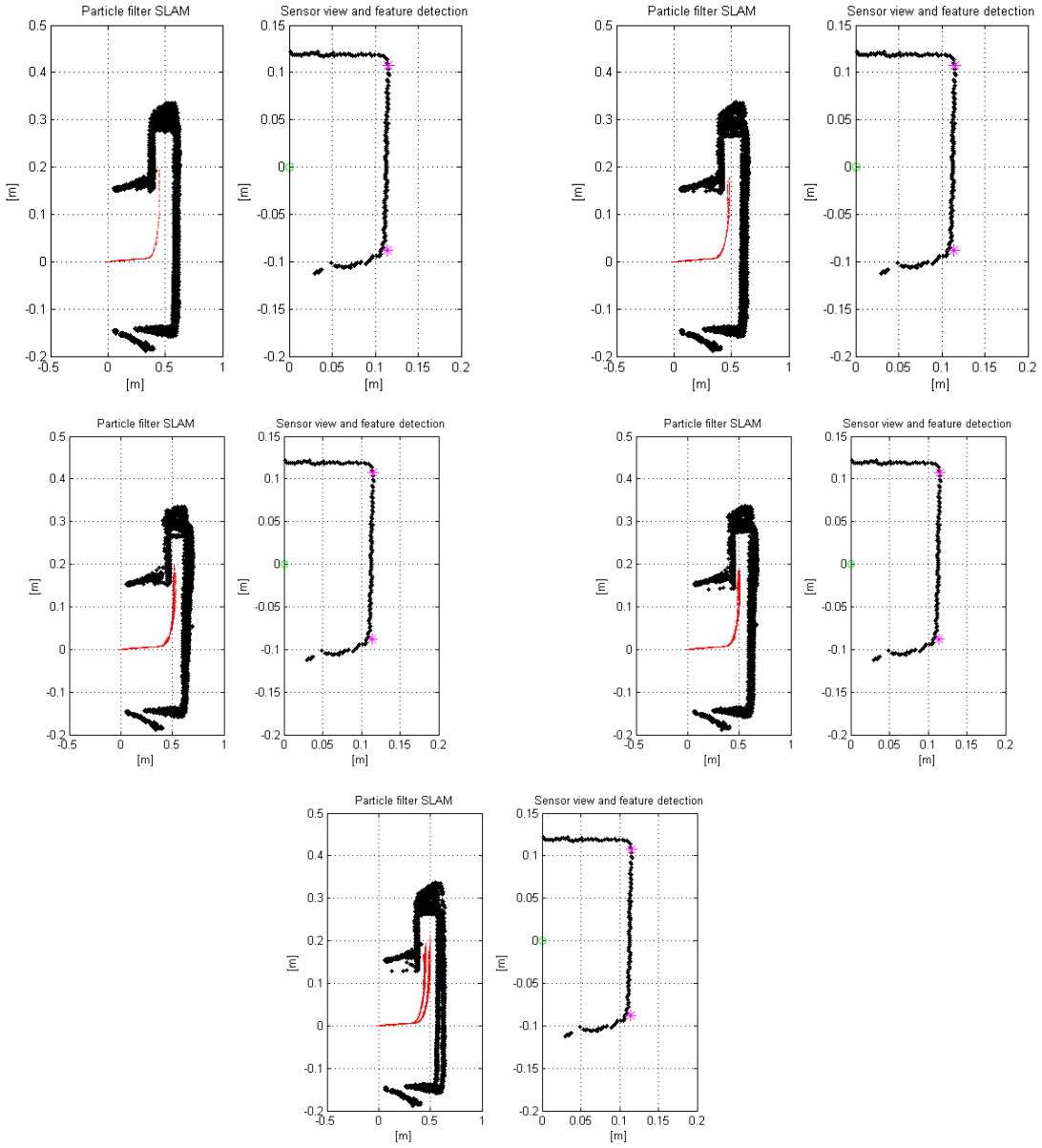

Figure 5: Het aantal $NPARTICLES$,
linksboven $NPARTICLES = 5$, rechtsboven $NPARTICLES = 20$
linksmidden $NPARTICLES = 100$, rechtsmidden $NPARTICLES = 200$
Onderaan $NPARTICLES = 1000$

8 Code snippets

Hieronder zijn enkele code snippets te vinden van files die flink zijn aangepast ten opzichte van hoe ze zijn geleverd.
FitLine.m bevat de code om de hoek en de afstand tot een lijn te bepalen.

Listing 2: Fitline.m

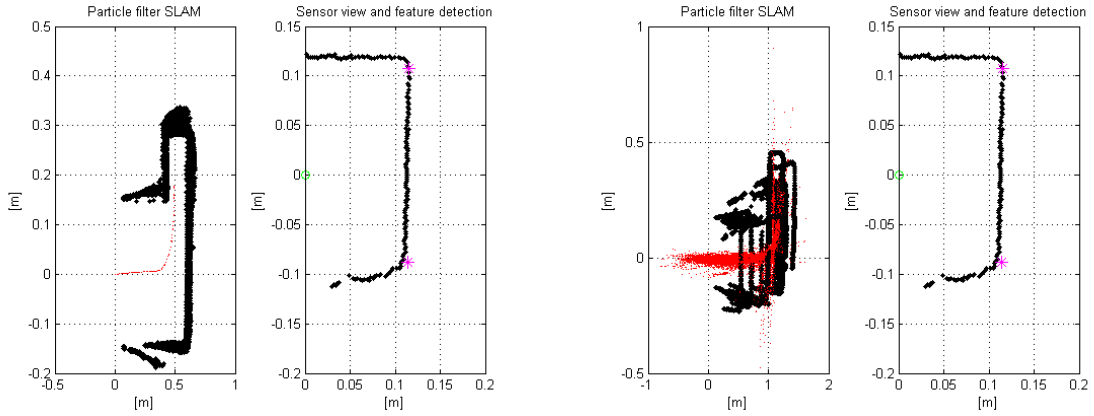

Figure 6: Odometrie variantie,
links $sigmaX = 0.00003m$ en $sigmaTH = 0.0002°$
rechts $sigmaX = 0.3m$ en $sigmaTH = 0.2°$

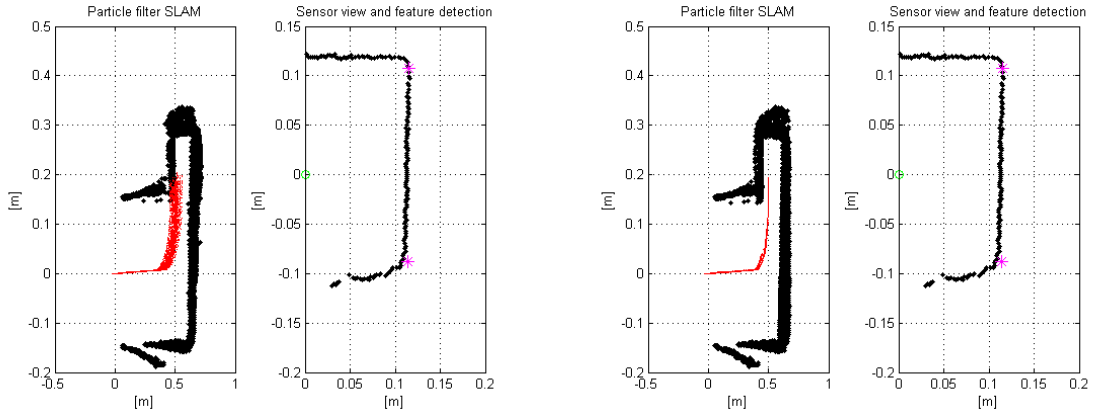

Figure 7: Rangesensor variantie,
links $sigmaR = 0.0001$ en $sigmaB = 0.0001$
rechts $sigmaR = 0.7$ en $sigmaB = 0.7$

```
%-------------------------------------------------------------------------
% This function computes the parameters (r, alpha) of a line passing
% through input points that minimize the total-least-square error.
%
% Input:  XY - [2,N] : Input points
%
% Output: r, alpha: paramters of the fitted line

function [r, alpha] = FitLine(XY)

  % Compute the centroid of the point set (xmw, ymw) considering that
  % the centroid of a finite set of points can be computed as
  % the arithmetic mean of each coordinate of the points.

  % XY(1,:) contains x position of the points
  % XY(2,:) contains y position of the points

```

```
  xmw = mean(XY(1,:));
  ymw = mean(XY(2,:));

  % compute parameter alpha (see exercise pages)
  nom = 0;
  denom = 0;
  for i=1:size(XY,2)
      nom = nom + ((XY(1,i) - xmw) * (XY(2,i) - ymw));
      denom = denom + ((XY(2,i) - ymw)^2 - (XY(1,i) - xmw)^2);
  end

  nom = -2 * nom;

  alpha = 0.5 * atan2(nom, denom);

  % compute parameter r (see exercise pages)
  r = xmw * cos(alpha) + ymw * sin(alpha);

  % Eliminate negative radii
  if r < 0,
    alpha = alpha + pi;
    if alpha > pi, alpha = alpha - 2 * pi; end
    r = -r;
  end;

  return % function FitLine
```

predict_odo.m bevat de code die de odometrie voorspelt aan de hand van meegegeven variantie parameters.

Listing 3: predict_odo.m

```
function particle = predict_odo(particle, dr, dth, Q)
% Q  cov matrix, noise odometry
% dr and dth is retrieved from the encoders (that is delta_r and delta_theta) in dt
% time.

Dmv = multivariate_gauss([dr; dth], Q, 1);
dr_n = Dmv(1);
dth_n = Dmv(2);


% predict state
xv= particle.xv;

% xv contains the *LAST* position of the considered particle in world coordinate frame,
    such as:
% xv(1) = last x
% xv(2) = last y
% xv(3) = last theta

% In order to obtain the prediction step for the current particle 'particle'
% consider then the measurements of the odometry of the robot that are:
% dr_n = noise added measurement of dr using cov. matrix Q
% dth_n = noise added measurement of d_th using cov. matrix Q

% Hint:
% x_new = x_old + dn_rn*cos(dth_rn + theta_old)

particle.xv= [ (xv(1) + dr_n * cos(dth_n + xv(3))) ;
               (xv(2) + dr_n * sin(dth_n + xv(3))) ;
```

```
29              pi_to_pi(pi_to_pi(xv(3)) + dth)];
30
31
32  % pi_to_pi is a *safe* function that converts angles from [0,2*pi] into the
33  % standard convention {[0,pi], (-pi, 0]}
```

logger/MainOffline.m bevat de code voor het opbouwen van de log voor de gebruikte offline dataset.

Listing 4: logger/MainOffline.m

```
1   % ========================================================================
2   %
3   %  Main.m
4   %  Closed-loop position control for differential-drive robots.
5   %
6   %  The calling syntax is:
7   %       Main
8   %
9   %  Reference:
10  %  Introduction to: Autonomous Mobile Robots, Chapter 3.
11  %
12  %  This is an M-file for MATLAB.
13  %  Tested in: Matlab 7.1.0
14  %  Date: 12.04.07
15  %
16  % ========================================================================*/
17
18  clc; % clear console
19
20  path(path, './scans/');
21  path(path, './odometry/');
22
23  load('est_odo.mat');
24
25
26  %==================================%
27  %   PARAMETERS TO CHANGE         %
28  %==================================%
29
30  % NUMBER OF SCANS:
31  N = 360;
32
33  % TIME BETWEEN TWO SNAPSHOTS:
34  T_b_S = 0.5; %[s]
35
36  % FREQUENCY OF ODOMETRY UPDATE:
37  F_O_U = 5; %[Hz]
38
39  % NAME OF THE LOG FILE:
40  Log_name = 'log.txt';
41
42  %==================================%
43
44  % Open the log file for writing the data
45  FILE = fopen(Log_name, 'w')
46
47  tic;
48  for i=1:61
49      % Schat hier dx en dtheta
50      dx = est_odo(i, 1);
51      dtheta = deg2rad(est_odo(i, 2));
```

```
52      SaveEncoderData(FILE, toc, dx, dtheta, N);
53
54      disp(sprintf('Taking laser data %d', (i + 10)));
55      laser_scans = GetLaserScans(N, i+10);
56      SaveLaserData(FILE, toc, laser_scans);
57
58      pause(T_b_S);
59  end
60
61  fclose(FILE);
62
63  display('Execution Complete!');
```