Particle filter

Vanpaemel Simon

25 september 2015

Hoofdstuk 1

Particle filter

1.1 beschrijving particle filter

De particle filter (PF) is net als de KF een specifieke implementatie van de Bayes filter. Het verschil tussen de KF en de PF is dat deze laatste de mogelijkheid heeft om met arbitraire kansverdelingen voor $bel(x_t)$ te werken. De KF was immers beperkt tot het gebruik van normaalverdelingen (gausscurve). Door een arbitraire kansverdeling $bel(x_t)$ te bemonsteren wordt een groep monsters verkregen die deze kansverdeling benaderen. In de plaats van de kansverdeling continu voor te stellen, stelt men hier de kansverdeling voor door een set monsters. Dit zorgt weliswaar voor een benadering, maar dit zorgt ervoor dat de verdeling die wordt verkregen niet-parametrisch is en er dus veel meer verdelingen kunnen worden voorgesteld.

Een monster $x_t^{[m]}$ $(1 \leqslant m \leqslant M)$, met M het aantal monsters, is een bepaalde toestand waarin de robot zou kunnen verkeren op tijdstip t. Dit is dus een hypothese en niet de werkelijke toestand van de robot. Een toestand van de robot is bijvoorbeeld de positie waar de robot zich op dat moment bevindt. Deze monsters kunnen gegroepeerd worden in een set

$$\chi_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{1.1}$$

Deze set χ_t wordt ook wel de particle set genoemd, waarbij een particle dus een ander benaming is voor een monster. In de tekst die volgt worden deze benamingen dan ook door elkaar gebruikt.

De kans dat een hypothese x_t deel zal uitmaken van χ_t is gelinkt aan de bayes filter kansuitdrukking $bel(x_t)$:

$$x_t^{[m]} \sim p(x_t|z_{1:t}, u_{1:t}) \tag{1.2}$$

Dit zorgt ervoor dat hoe meer particles min of meer dezelfde hypothetische toestand hebben, hoe groter de kans is dat de werkelijke toestand ongeveer die toestand zal zijn.

De particle set wordt opgebouwd op een recursieve manier, χ_t is gebaseerd op χ_{t-1}. Het algoritme dat hiervoor instaat wordt weergegeven op figuur1.1. Dit algoritme is een speciale vorm van de PF, het is namelijk het algoritme met MCL aanpassing. Het verschil met het gewone algoritme wordt uitgelegd in sectie 1.2

1: **Algorithm Particle_filter**($\mathcal{X}_{t-1}, u_t, z_t$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: for $m = 1$ to M do
4: sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: endfor
8: for $m = 1$ to M do
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to $\mathcal{X}_t$
11: endfor
12: return $\mathcal{X}_t$

Figuur 1.1: Algoritme voor opbouwen particle set

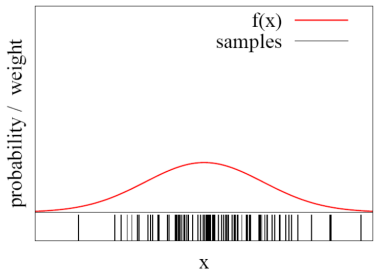
Het algoritme genereert nieuwe mogelijke toestanden (dus nieuwe particles) voor één particle op basis van de controle signalen u_t en de vorige hypothetische toestand van het particle (motion model). Anders gezegd neemt men de positie van het particle van de vorige tijdstap dan past men er de controlesignalen op toe en dan worden nieuwe samples gegenereerd rond deze nieuwe positie. Daarna wordt bekeken of de nieuwe hypothetische toestand overeenkomt met de werkelijke toestand. Dit wordt verwezenlijkt door de werkelijke observatie te vergelijken met de observatie die de robot ziet indien hij zich in de hypothetische toestand bevindt. Indien deze 2 observaties overeenkomen, is het plausibel dat de nieuwe hypothetische toestand zich dicht bij de werkelijke toestand bevindt (observation model). Afhankelijk van hoe plausibel deze toestand is voor de werkelijkheid, wordt een gewicht $w_t^{[m]}$ gegeven aan het particle. Dit gewicht is een maat voor de belangrijkheid van het particle, als de toestand van het particle goed lijkt overeen te komen met de werkelijkheid krijgt het particle een hoog gewicht. De particles, samen met hun gewicht, worden voorlopig opgeslaan in $\overline{\chi_t}$. Deze set monsters geeft eigenlijk $\overline{bel}(x_t)$ weer (prediction step).Daarna worden particles getrokken uit $\overline{\chi_t}$, deze actie wordt herbemonsteren genoemd. Hoe groter het gewicht van de particles, hoe groter de kans dat de particles getrokken worden. De getrokken particles worden daarna in de definitieve particle set χ_t gezet. Deze particle set geeft na het herbemonsteren $bel(x_t)$ weer (correction step), aangezien hier de observaties (gerelateerd met gewicht) worden meegenomen in de keuze van de samples.

1.2 Sampling

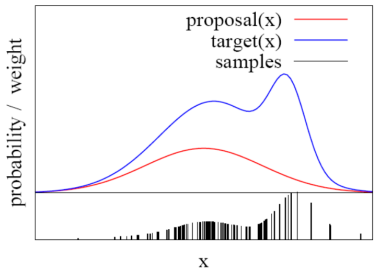
Aangezien de kern van de PF nu net gaat over monsters van een kansverdeling, moeten er dus bemonsterd worden om een set monsters te bekomen. Samplen van een arbitraire distributie, target distribution f genoemd, is echter niet altijd mogelijk. Als oplossing wordt een andere kansverdeling bemonsterd die bemonsterbaar is, de proposal distribution g genoemd. Voor deze verdeling wordt veelal de gausscurve gebruikt. Waarbij f overeenkomt met $bel(x_t)$ en g overeenkomt met $\overline{bel}(x_t)$. Door voor elk monster de kans van de target distribution te delen door de kans van de proposal distribution wordt een gewicht bekomen:

$$w^{[m]} = \frac{f(x_{[m]})}{g(x^{[m]})} \tag{1.3}$$

Op deze manier kan er dus worden gesampled en tegelijk een gewicht meegegeven worden aan de samples, dit wordt geïllustreerd in figuren 1.2 en 1.3. Bij MCL, een speciale vorm van de PF, wordt voor de proposal distribution het motion model gekozen. Dit is te zien op lijn vier van het algoritme op figuur1.1. Indien als proposal het motion model wordt gekozen, kan aangetoond worden dat de gewichten evenredig zijn volgens het observation model, dit is te zien op lijn vijf van het algoritme op figuur 1.1. Dit laatste wordt aangetoond in sectie1.3.



Figuur 1.2: Bemonstering van proposal distribution



Figuur 1.3: Bemonstering van target distribution

1.3 Afleiding gewichten bij MCL particle filter

In deze sectie wordt de bewering bewezen die in sectie1.2 werd gemaakt:"Indien de proposal distribution gelijk wordt genomen aan het motion model, dan de gewichten evenredig zijn met het observation model."

Ieder particle is een geschiedenis van hypothetische toestanden waarin de robot zou zijn geweest. Deze toestanden zijn, zoals bovenvermeld, in MCL de posities van de robot. Een particle bevat dus eigenlijk een hypothese van het traject dat de robot heeft doorlopen. De PF berekent dus eigenlijk $p(x_t|z_{1:t}, u_{1:t})$

maar dan over alle toestanden van de robot:

$$bel(x_{0:t}) = p(x_{0:t}|u_{1:t}, z_{1:t}) \tag{1.4}$$

Analoge afleiding van sectie (JENS ZIJN AFLEIDINGE) toegepast op $bel(x_{0:t})$ levert volgende uitdrukking op (met assumptie van oneindig veel particles):

$$bel(x_{0:t}) = p(x_{0:t}|u_{1:t}, z_{1:t}) = \eta p(z_t|x_t)p(x_t|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1}) \tag{1.5}$$

Zoals bovenvermeld komt de proposal overeen met $\overline{bel}(x_t)$, namelijk het motion model samen met een recurieve term:

$$\overline{bel}(x_t) = p(x_t|x_{t-1}, u_t)bel(x_{0:t-1}) = p(x_t|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1}) \tag{1.6}$$

vergelijkingen 1.5 en 1.6 invullen in vergelijking1.3 geeft

$$w_t^{[m]} = \frac{\eta p(z_t|x_t)p(x_t|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})}{p(x_t|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1})} \tag{1.7}$$

met als resultaat:

$$w_t^{[m]} = \eta p(z_t|x_t) \tag{1.8}$$

In vergelijking1.8 zien we dat de gewichten gelijk zijn aan het observation model. Merk op dat door te herbemonsteren met gewichten $w_t^{[m]}$, de overblijvende monsters in χ_t verdeeld zijn volgens het product van de proposal en de gewichten:

$$bel(x_{0:t}) = \eta w_t^{[m]} p(x_t|x_{t-1}, u_t)p(x_{0:t-1}|z_{1:t-1}, u_{1:t-1}) \tag{1.9}$$

1.4 Voor-en nadelen van de particle filter

1.4.1 Voordelen

- Er kunnen niet-parametrische target distributions worden gebruikt.
- Er kan geen foute data-associatie gebeuren, wat bij de KF wel kan. De reden hiervoor is dat indien de robot bijvoorbeeld 2 of meer plausibele toestanden van de werkelijkheid heeft, er tussen deze toestanden niet wordt gekozen. Ze worden immers allebei gekozen, door beide toestanden te bemonsteren. Wanneer men verdergaat in de tijd en de robot dus heeft bewogen, zullen er bepaalde toestanden minder waarschijnlijk worden en zullen deze verdwijnen. Vergelijk het met dat er op de wereld 2 exact dezelfde straten zouden zijn in een verschillende stad , dan worden deze 2 straten bemonsterd. Wanneer de robot beweegt doorheen de straat en uiteindelijk in een andere straat terechtkomt, zal in 1 van de twee straten de observatie niet overeenkomen met de map die reeds werd gemaakt, dan weet men dat men zich in de andere straat bevindt.

1.4.2 Nadelen

- Er zijn veel particles nodig zodat deze monsters de gewenste target distribution goed weergeven.
- Wanneer men bij MCL met sensoren werkt die zeer accuraat zijn ($p(z|x)$ is bijna een dirac-distributie), dan kan het zijn dat het motion model (de proposal) geen samples genereert op de plaatsen in het gebied waar $p(z|x) > 0$. Dit zorgt ervoor dat alle gewichten nul zijn, en geeft dus problemen wanneer er moet herbemonsterd worden.
- Werkt het best als de toestandsvector laag-dimensioneel is. Aangezien een particle een hypothese is voor een toestand, zijn er minder particles nodig voor een toestand te beschrijven met een weinig aantal dimensies dan een toestand met veel dimensies.

1.5 Mapping mbv Rao-Blackwellization

Indien er naast de positie van de robot ook de posities van de landmarks zouden worden meegenomen in de toestandsvector, dan zou de toestandsvector hoog-dimensioneel worden. Dit is zoals gezegd in sectie 1.4.2 niet gewenst. De oplossing hiervoor is om een verband te zien tussen de positie van de robot en de landmarks, namelijk eens je de positie van de robot weet, is het makkelijk om de omgeving in kaart te brengen. Want eenmaal je het traject van de robot weet, kan je de positie van de landmarks updaten met de verkregen observaties van de sensoren. De idee is dus om met de PF de poses van de robot te bepalen. Elk sample heeft een eigen traject, en dus een eigen map die verschillend is van de andere particles. Aan ieder traject hangt dus een bepaalde map en die map is het resultaat van het traject die werd afgelegd.

Het principe van Rao-Blackwellization kan men als volgt samenvatten in formules:

$$p(a, b) = p(b|a)p(a) \tag{1.10}$$

Waarbij a het traject van het particle voorstelt en b de map voorstelt. De eerste factor $p(b|a)$ kan efficiënt berekent worden voor elk sample en p(a) stelt de samples voor, namelijk $bel(x_{0:t})$. Anders gezegd kunnen we vergelijking 1.10 ook voorstellen als:

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) = p(m_{1:M}|x_{0:t}, z_{1:t})p(x_{0:t}|z_{1:t}, u_{1:t}) \tag{1.11}$$

Waarbij dus de eerste factor $p(m_{1:M}|x_{0:t}, z_{1:t})$ de map opbouwd op basis van het traject en de observaties en de tweede factor $p(x_{0:t}|z_{1:t}, u_{1:t})$ het traject voorstelt, merk op dat $m_{1:M}$ hier de map voorstelt op basis van landmarks, met M het aantal landmarks. Om de PF computationeel mogelijk te maken is het de bedoeling dat wanneer er een traject van het particle dus voorhanden is er efficiënt een map van dat particle kan worden opgebouwd. Alle landmarks zijn onafhankelijk van elkaar als je het traject van de robot kent (CYRILL

STACHNISS ZEGT DA, LECTURE 12 minuut 20:08). Dit zorgt ervoor dat je de factor die de map met landmarks berekent kunt opsplitsen in een product dat één enkel landmark berekent. Hierbij stellen de factoren $p(m_i|x_{0:t}, z_{1:t})$ onafhankelijke gaussische distributies voor, dit geeft:

$$p(x_{0:t}, m_{1:M}|z_{1:t}, u_{1:t}) = \prod_{i=1}^{M} p(m_i|x_{0:t}, z_{1:t}) p(x_{0:t}|z_{1:t}, u_{1:t}) \tag{1.12}$$

De map wordt nog steeds gemaakt zoals in de methode van de EKF, dit komt erop neer dat er in de plaats van een hoog dimensionele EKF, moet er een kleine 2x2 EKF worden opgelost voor elk landmark. Elk particle heeft dus een 2x2 EKF dat het moet oplossen voor de bepaling van ieder landmark, wat zeer efficiënt is. De keerzijde is dat dit wel voor ieder particle moet gebeuren, maar toch is dit nog efficiënter dan indien er een hoog dimensionele EKF zou moeten worden opgelost. (DA BENK IER NIE GJIL ZEKER, zie cyrill 22:24 les 12)

1.5.1 Implementatie: FastSLAM

FastSLAM 1.0

FastSLAM is de implementatie van de PF in combinatie met de mapping adh van Rao-Blackwellization. Hier wordt wel niet het volledige traject meegenomen maar enkel de huidige positie. Dit komt omdat wanneer de robot verder beweegt in de tijd we nooit terugkeren naar een bepaalde toestand vroeger om bijvoorbeeld de positie ervan te corrigeren. De PF kijkt altijd in de toekomst om de positie in de volgende tijdstap te berekenen. Het is de bedoeling dat er wel online, dus voor iedere tijdstap een huidige map wordt gemaakt die ergens wordt weggeschreven. Een particle houdt dus de positie bij waarop het zich op dat moment bevindt en ieder landmark wordt voorgesteld door een 2x2 EKF, zie figuur1.1.

Figuur 1.4: Particles bevatten hun positie en 2x2 EKF voor de voorstelling van ieder landmark

FastSLAM 2.0

Het verschil met Fastslam 1.0 is om in de proposal distribution niet enkel de odometry te gebruiken voor het bepalen van de volgende toestand maar nu ook rekening te houden met sensor informatie. Dit zorgt ervoor dat meer samples beter zullen aansluiten bij de target distribution. Dit zorgt er voor dat er minder particles nodig zijn en dat de map nauwkeuriger zal zijn.

1.5.2 Grid-Based FastSLAM

Dit is een particle gebaseerd SLAM algoritme voor het maken van grid-maps. Het voordeel is dat er geen feature moet worden ingebouwd om data-associatie te verwezenlijken tussen de landmarks en er dus geen foute data-associatie kan gebeuren. Grid-Based SLAM maakt gebruikt van het principe van FastSLAM 2.0 omdat er hier landmarks zijn waarbij de robot deze bij de measurement update kan corrigeren. Een grid map waarbij het traject niet heel correct is, is minder goed dan een landmark gebaseerde map voor navigatie. De observatie die wordt meegenomen in de proposal distribution is een zogenaamde scan-matcher. Hierbij wordt de kans gemaximaliseerd dat deze huidige map en positie relatief overeenkomt met de vorige map en positie. Anders gezegd neemt men de map van de vorige positie en de map van de huidige positie en verschuift men (roteren,transleren) de map zodat de huidige map het best overlapt met de vorige map. In formule wordt dit als volgt weergegeven:

$$x_t^* = \underset{x_t}{argmax}\{p(z_t|x_t, m_{t-1})p(x_t|u_{t-1}, x_{t-1}^*)\} \tag{1.13}$$

Mathematisch gezien is dit niet volledig correct, aangezien de observatie al reeds wordt gebruikt in de proposal distribution. Dit zorgt ervoor dat de berekening voor de gewichten zou moeten worden aangepast. Als oplossing hiervoor wordt niet alles gescanmatched, maar enkel bepaalde blokken van bijvoorbeeld 100 posities. Als resultaat krijg je voor die 100 posities een locale map die nauwkeurig is, en dan gebruik je de scanmatcher om deze lokale mappen te laten overlappen.