



P&O Computerwetenschappen, Hoofdrichting

Sheephunt

een spel gebaseerd op plaatsbepaling m.b.v. WiFi-signalen

Groep H3

verslag

16 mei 2007

Adriaensen Chris¹

Balog Daniel

Cuypers Cedric

Dellaert Philippe²

Maerien Jef

Van Den Broeck Guy

Van Ranst Wouter

¹ secretaris

² coördinator

Inhoudsopgave

Inleiding	3
Sheephunt.....	5
2.1. Het spel	5
2.2. Spelverloop op de server	7
2.3. Spelverloop op de PDA.....	7
Graphical User Interface.....	9
3.1. ClientGUI	9
3.1.1. <i>Technische specificaties</i>	9
3.1.2. <i>Eerste versie</i>	10
3.1.3. <i>Ondervraagmethodiek</i>	11
3.1.4. <i>Opmerkingen van testgroep op eerste versie</i>	12
3.1.5. <i>Tweede versie</i>	13
3.1.6. <i>Opmerkingen van testgroep op tweede versie</i>	14
3.2. ServerGUI.....	14
Plaatsbepaling	15
4.1. Fingerprinting methode	15
4.1.1. <i>Radio Map</i>	16
4.1.2. <i>Fingerprint</i>	17
4.1.3. <i>Scan</i>	17
4.2. K-nearest-neighbours methode	17
4.2.1. <i>Uitbreiding: P-previous-k-nearest-neighbours methode</i>	18
4.3. Particle Filter.....	19
4.3.1. <i>Particle</i>	19
4.3.2. <i>Algoritme</i>	19
4.4. Experimenten.....	25
Ontwerp.....	29
5.1. Machine niveau.....	29
5.2. Conceptueel niveau	30
5.3. Implementatie niveau	32
5.3.1. <i>PositioningSystem</i>	32
5.3.2. <i>GUI</i>	32

5.3.3. <i>Communication</i>	34
5.3.4. <i>Server</i>	36
5.3.5. <i>Game</i>	39
Proces	41
6.1. Moeilijkheden	41
6.2. Wat hebben we ervan geleerd?	42
6.3. Werken in team	43
Dankwoord	45
Bronnen	46
Appendix A Werkbelasting	47
Appendix B Afstandsfuncties	50
B.1. Afstand tussen fingerprint en scan voor 1 WAP	50
<i>B.1.1. Verschil van gemiddelden</i>	50
<i>B.1.2. Verschil van medianen</i>	51
<i>B.1.3. Correlatie tussen de verdelingen</i>	51
<i>B.1.4. Hypothesetesten</i>	54
<i>B.1.5. Niet-lineaire transformatie: exponentiël 1</i>	55
<i>B.1.6. Niet-lineaire transformatie: exponentiël 2</i>	55
B.2. Afstand tussen fingerprint en scan	55
<i>B.2.1. Oneindig-norm</i>	56
<i>B.2.2. Gemiddelde van afwijkingen</i>	56
<i>B.2.3. 2-norm</i>	56
<i>B.2.4. Gewogen 2-norm</i>	57
Appendix C Update methodes	58
C.1. PerMeasurement	58
C.2. PerMeasurementInterpolation	58
C.3. PerMeasurementAdditive	59
C.4. Knearest	59
Appendix D Testresultaten	60

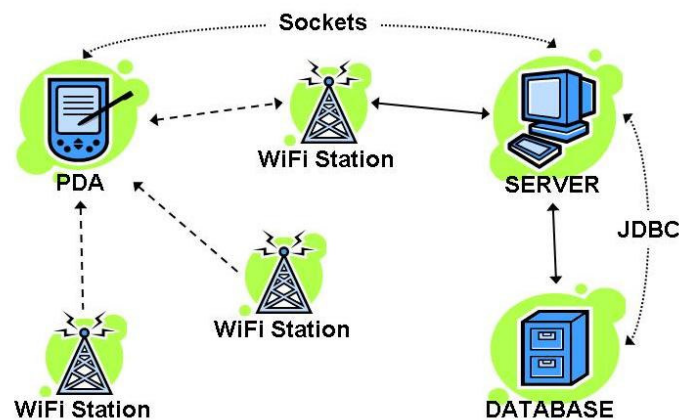
Hoofdstuk 1

Inleiding

Plaatsbepaling m.b.v. WiFi-signalen geeft ons de mogelijkheid om binnenshuis de positie van een ontvanger te bepalen. De rol van ontvanger wordt in onze toepassing ingevuld door een *Personal Digital Assistant* (PDA).

Meerdere spelers proberen in *Sheephunt* een losgeslagen schaap in een bepaald gebouw te vangen. Wordt dit schaap gevangen dan verschijnt er weer een nieuw schaap ergens in het gebouw. De speler die het eerst een vooraf bepaald aantal schapen kan vangen wint het spel.

De gebruiker interageert met het spel d.m.v. een gebruiksvriendelijke *Graphical User Interface* (GUI). De communicatie tussen de PDA en de server verloopt via *Sockets*. Op deze of een andere server bevindt zich een database waarin alle nuttige informatie wordt opgeslagen. De communicatie hiermee verloopt via *Java Database Connectivity* (JDBC).



Figuur 1 Opstelling met één enkele PDA.
(Gestreepte verbindingen zijn draadloos.)

Door gebruik te maken van een complexe plaatsbepalingsmethode, gebaseerd op een *particle filter* die gebruik maakt van de *p-previous-k-nearest-neighbours* methode, kan de server de positie van de PDA tot op ongeveer 2 à 3 meter nauwkeurig bepalen en dit tegen een maximale frequentie van 20 Hz¹ (1 keer / 50 msec).

In Hoofdstuk 2 bekijken we de spelwerking van *Sheephunt* in meer detail. Hoofdstuk 3 beschrijft de evolutie van de GUI's tot het huidig ontwerp. De gebruikte plaatsbepalingsmethode, de bijhorende experimenten en optimalisaties worden in Hoofdstuk 4 in detail bekeken.

¹ Deze frequentie houdt enkel rekening met de nodige rekentijd voor de plaatsbepalingmethode op de server. De server is hierbij een desktop machine uitgerust met een Intel® Pentium® 4 2,8 Ghz processor en 1 Gb RAM-geheugen. De totale tijd vanaf de meting op de PDA tot het einde van de plaatsbepaling bedraagt in onze experimenten ongeveer 600 à 700 msec (1,7 à 1,4 Hz) afhankelijk van de omstandigheden (zie Hoofdstuk 4 voor meer informatie hieromtrent).

Hoofdstuk 2

Sheephunt

In dit hoofdstuk bekijken we de spelwerking van *Sheephunt*. Zoals reeds vermeld in de inleiding, proberen meerdere spelers in *Sheephunt* (zie Figuur 2) een losgeslagen schaap te vangen. Wordt dit schaap gevangen dan verschijnt er weer een nieuw schaap ergens in het gebouw. De speler die het eerst een vooraf bepaald aantal schapen kan vangen wint het spel. We bekijken eerst in 2.1 de spelwerking algemeen, daarna specifiek in 2.2 op de server en in 2.3 op de PDA.

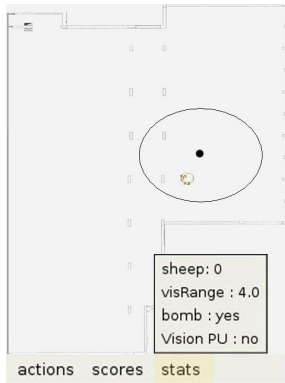


Figuur 2 *Sheephunt*-logo.

2.1. Het spel

Een speler probeert zoveel mogelijk schapen te vangen d.m.v. een map van het gebouw die getoond wordt op zijn PDA. De speler kan hierop enkel objecten zien die binnen zijn gezichtsveld liggen. Een dergelijk object kan zijn: het gezochte schaap (zie Figuur 3), een schatkist (zie Figuur 4) of een bom die door de speler zelf is geplaatst (zie Figuur 5). We bekijken deze objecten verder in detail.

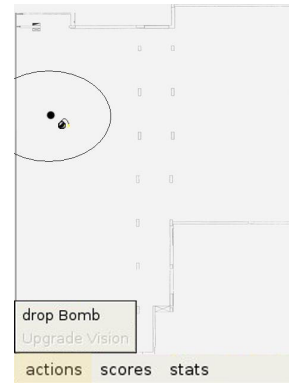
Als een speler dicht genoeg bij het schaap kan komen zal hij het vangen en een punt krijgen. Indien deze speler dan het vooraf bepaald aantal punten heeft vergaard wint hij het spel en eindigt tevens het spel voor alle spelers.



Figuur 3 Speler en schaap.



Figuur 4 Speler, schaap en schatkist.
(Speler heeft een VPU.)



Figuur 5 Speler en bom.

Een speler kan ook een schatkist tegenkomen, hier is de vangtechniek identiek als bij die van een schaap. De schatkisten bevatten willekeurige objecten. Voorlopig zijn er 2 mogelijke items geïmplementeerd: bommen en *Vision Power Up's*. Spelers kunnen maximaal 1 van elk item bij zich hebben. De spelers bepalen dan zelf wanneer ze deze items gebruiken d.m.v. het ACTIONS-menu (zie Figuur 5).

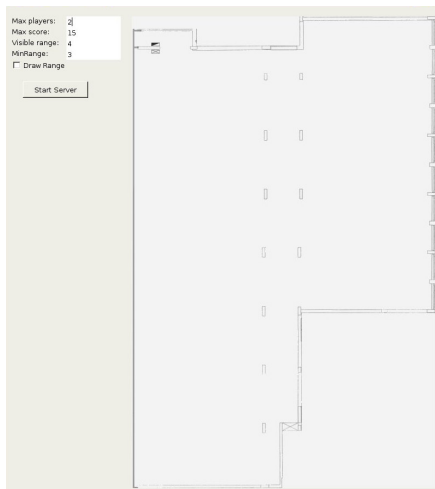
Vision Power Up's (VPU's) vergroten tijdelijk het gezichtsveld van een speler met een random factor zodat deze bv. het schaap vanop grotere afstand kan zien op de map (zie Figuur 4). Eenmaal de *Vision Power Up* zijn kracht verloren heeft, krijgt de speler de volgende boodschap 'VISIONPU FADED'.

Bommen kunnen door een speler op de map geplaatst worden tijdens het spel. Als een andere speler dan in de buurt komt van zo'n bom, zal de bom ontploffen en verliest deze speler een punt.

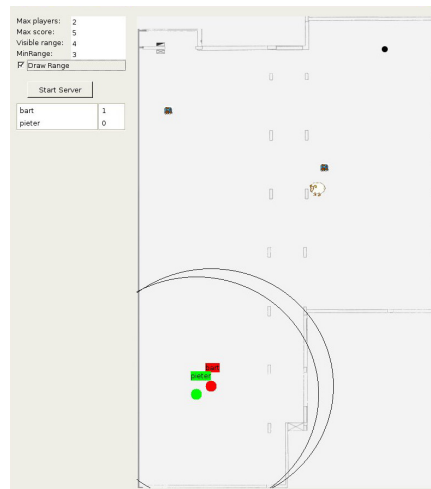
We bekijken nu het spelverloop op de server en de PDA.

2.2. Spelverloop op de server

De spelbeheerder die zich achter de computer bevindt waarop de server draait, krijgt bij het opstarten een scherm te zien waarin hij een nieuw spel kan aanmaken met bepaalde parameters die hij zelf kan instellen (zie Figuur 6), nl. het aantal spelers in het nieuwe spel, de benodigde score om het spel te winnen, de standaard straal voor het gezichtsveld van de spelers en de straal waarbinnen een speler moet geraken om een object te kunnen vangen / oppikken.



Figuur 6 ServerGUI bij het opstarten.



Figuur 7 ServerGUI tijdens een spel.

Eenmaal een spel aangemaakt is, wordt er weergegeven welke spelers al reeds zijn ingelogd. Als er voldoende spelers zijn ingelogd start het spel. Het spel kan dan vervolgens continue gevolgd worden vanop de server (zie Figuur 7).

2.3. Spelverloop op de PDA

In een eerste stap moet de speler inloggen op de server. De spelers krijgen een login-scherm te zien waarop ze hun naam als identificatie moeten ingeven (zie Figuur 9). De server controleert vervolgens of de naam al in gebruik is. Is de naam nog vrij dan wordt de speler ingelogd. Hij krijgt dan een lobby-scherm te

zien met de reeds ingelogde spelers. Als er voldoende spelers ingelogd zijn, wordt er op de PDA een wacht-scherf getoond met de spelers die zullen deelnemen aan het spel en een boodschap dat het spel weldra van start gaat.

Indien één van de spelers het nodig aantal punten heeft bereikt om te winnen, eindigt het spel en krijgen alle spelers hetzelfde eindscherf te zien met daarop een lijst van alle spelers samen met hun behaalde scores. Ook krijgen ze de mogelijkheid om het spel opnieuw te spelen (RESTART) of te beëindigen (END).

Hoofdstuk 3

Graphical User Interface

Opmerking [AC1]: Na te kijken door Daniel & Jef.

Opmerking [AC2]:
Onderdeel > 3 blz

In dit hoofdstuk bekijken we de evolutie van de 2 *Graphical User Interfaces* (GUI's) in *Sheephunt* tot hun finale vorm. De GUI op de PDA draagt de naam *ClientGUI* en bespreken we in 3.1, de GUI op de server draagt de naam *ServerGUI* en bespreken we in 3.2.

3.1. ClientGUI

ClientGUI is de interface tussen het spel en de speler op de PDA. Om deze en eender welke gebruiksvriendelijke GUI te ontwerpen moet je eerst voldoen aan 2 soorten specificaties: de technische specificaties en de gebruikersspecificaties. De technische specificaties worden door het spel zelf bepaald. Sommige dingen kunnen niet worden veranderd omdat ze een essentieel deel van het spel zijn. Zo moet er in het spel bv. een manier zijn om bommen te plaatsen op de map. Hoe dit gebeurt echter hangt af van de gebruikersspecificaties. De gebruikersspecificaties worden bepaald a.d.h.v. feedback van proefpersonen. Deze proefpersonen kunnen kritiek geven over een GUI prototype. Deze kritieken worden dan verzameld en verwerkt, om zo tot een betere GUI te komen. De uiteindelijke versie is niets anders dan de laatste iteratie van het oorspronkelijk prototype van de GUI.

3.1.1. Technische specificaties

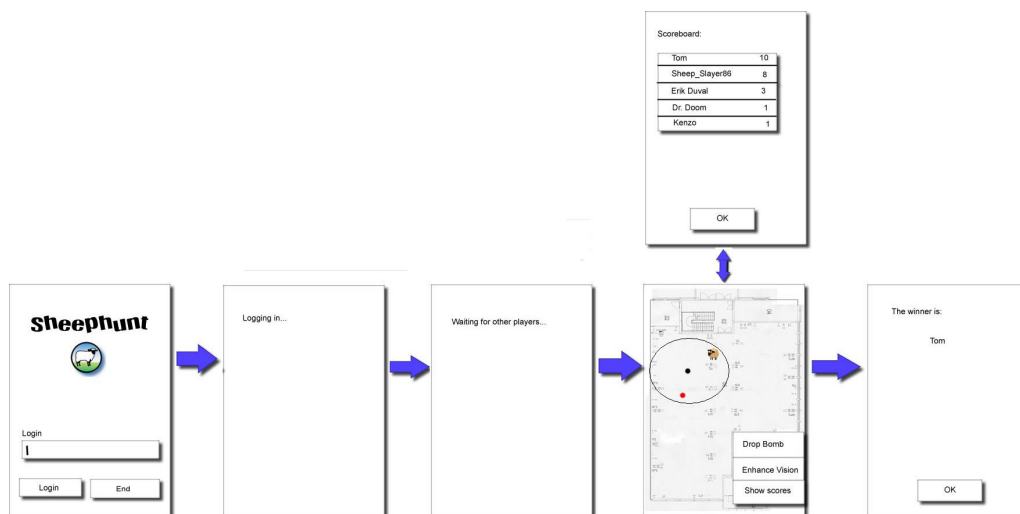
- Elke speler moet zich op voorhand inloggen met een naam. Om onduidelijkheden te vermijden moet in elk spel elke gebruiker een

unieke naam hebben. Een wachtwoord is niet nodig, aangezien er geen identificatie-bewijs moet geleverd worden omdat een naam niet uniek hoeft te zijn over verschillende spelen heen. Dit inloggen kan soms enkele seconden duren, het is dus aan te raden dit weer te geven aan de gebruiker.

- Nadat de speler ingelogd is, zal deze een tijd moeten wachten tot de andere gebruikers ook ingelogd zijn.
- Tijdens het spel loopt de speler rond op de map. Hij heeft de mogelijkheid om verschillende acties te ondernemen, nl. een bom plaatsen of een *Vision Power Up* activeren. Deze moeten hem vanzelfsprekend aangeboden worden.
- Op het einde van het spel moet de winnaar aangegeven worden.

3.1.2. Eerste versie

In een eerste versie is er gekozen voor een simpel login-scherm (zie Figuur 8). Het bevat het *Sheephunt*-logo, een tekstveldje voor de naam en twee knoppen voor in te loggen (LOGIN) en het spel te verlaten (END).



Figuur 8 Eerste versie van ClientGUI.

Wanneer de gebruiker zijn naam heeft ingevuld en de LOGIN-knop indrukt, wordt hij naar een 'logging in'-scherm gebracht. Wanneer vervolgens de speler een bevestiging van de server ontvangt, gaat hij naar een simpele wachtkamer met bijhorend wacht-scherm. Daar wordt de speler gevraagd om te wachten op de andere spelers.

Het spel start zodra alle spelers zijn ingelogd. Het spel-scherm wordt getoond met een map en 3 knoppen (DROP BOM, ENHANCE VISION, SHOW SCORES). De eerste twee knoppen dienen voor de mogelijke acties in het spel, de derde brengt de speler naar het score-scherm.

Er is geen mogelijkheid om het spel te verlaten voordat een speler gewonnen heeft. Indien een speler niet meer wilt meespelen, laat hij zijn PDA gewoon achter. Eenmaal het spel afgelopen is, gaat de speler naar een eind-scherm waarop de winnaar wordt bekendgemaakt. Met een OK-knop kan de speler dit scherm verlaten en de speler komt terug bij het login-scherm.

3.1.3. Ondervraagmethodiek

Om de proefpersonen te ondervragen werd er gebruik gemaakt van een papieren GUI prototype. Eerst werd er een uitleg gegeven over het spel zelf; dat ze m.b.v. een PDA uitgerust met plaatsbepaling een losgeslagen schaap moesten zoeken en vangen.

Aan alle proefpersonen werd gevraagd hun mening over de GUI te geven. Deze input werd dan gebruikt om een nieuw (en mogelijk beter) perspectief te bekomen op het geheel. Alle opmerkingen werden opgeschreven zodat ze later konden worden verwerkt (zie 3.1.4).

In tegenstelling tot onze toepassing in het eerste semester¹, is dit project minder leeftijdsgebonden. De zoektocht naar proefpersonen verliep daarom veel vlotter aangezien nu bijna iedereen een potentieel kandidaat was. Uiteindelijk werden er voornamelijk kotgenoten ondervraagd, met een gemiddelde leeftijd van 20 jaar.

3.1.4. Opmerkingen van testgroep op eerste versie

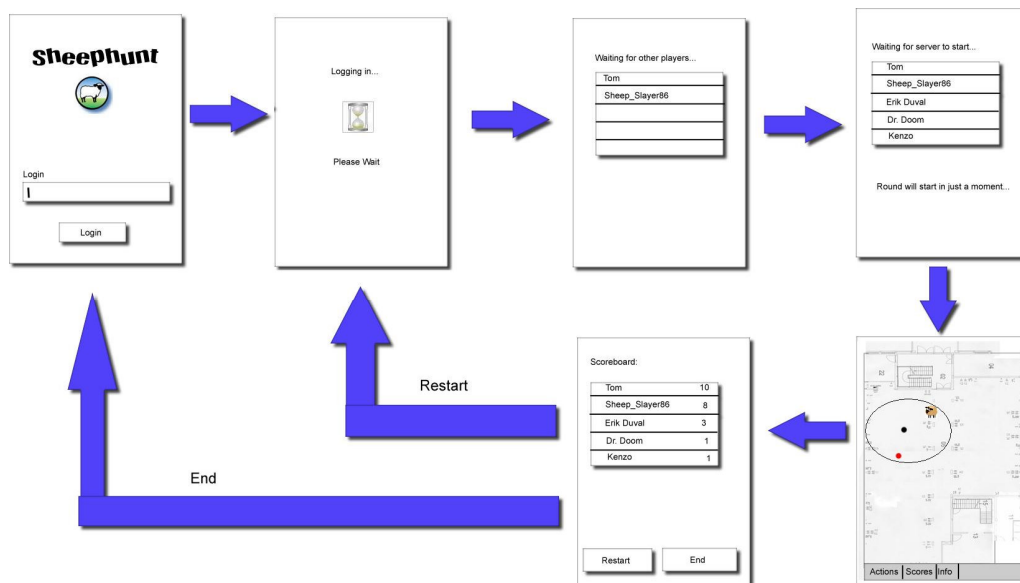
- END-knop is niet nodig in het login-scherm aangezien men op elk moment het spel kan sluiten met het kruisje (X) rechts bovenaan het scherm. (Aangezien de PDA toch enkel dient om het spel te spelen, is dit geen slechte opmerking.)
- Bepaalde proefpersonen vonden het een fijn idee om, net zoals in de huidige online-spelen, te tonen welke spelers er al ingelogd zijn op de server. Zo kan men beter inschatten hoe lang men nog moet wachten op de andere speler(s), en belangrijker nog: op welke speler(s).
- Knoppen op het spel-scherm nemen te veel plaats in op de map, en ze zijn storend.
- Een proefpersoon maakte de terechte opmerking dat het apart score-scherm het spelverloop verstoort.
- Het is voor de speler gewenst om op elk moment te weten of hij bommen of andere items heeft opgenomen van het speelveld.
- Er werd verzocht om i.p.v. het eind-scherm met enkel de winnaar, een eind-scherm met de statistieken van alle spelers te tonen. Deze statistieken kunnen dan bestaan uit o.a. de score.
- Wegens technische redenen is het nuttig om voordat het spel begint al reeds enkele metingen te kunnen maken. Zodoende kunnen de beginposities van de spelers nauwkeuriger bepaald worden. Daarom was het wenselijk om nog een scherm te voorzien tussen het lobby-scherm en het spel-scherm.

¹ In het eerste semester ontwikkelden we een quiz-tour die bv. kon gebruikt worden in musea. De doelgroep was toen vnl. leerlingen lager en secundair onderwijs.

3.1.5. Tweede versie

A.d.h.v. de opmerkingen van de testgroep werden dan de volgende wijzigingen aangebracht (zie Figuur 9):

- In het login-scherm werd de END-knop verwijderd.
- In het lobby-scherm worden de namen zichtbaar van de spelers die reeds ingelogd zijn op de server.
- Een extra scherm werd toegevoegd tussen het wacht- en spel-scherm.
- De knoppen in het spel-scherm zijn vervangen door een menubalk onderaan het scherm (zie ook Figuur 3). Ook de scores en de status van de speler kan men bekijken via deze menubalk.
- In het eind-scherm worden de scores van alle spelers getoond¹. Ook is er een RESTART-knop toegevoegd zodat een speler onmiddellijk een nieuw spel kan heropstarten.



Figuur 9 Tweede versie van ClientGUI.

¹ In het huidige ontwerp worden er op de server, buiten de score, geen andere statistieken bijgehouden.

3.1.6. Opmerkingen van testgroep op tweede versie

Tijdens deze test werd gebruik gemaakt van een geïmplementeerde versie van de tweede versie van *ClientGUI*. De reacties waren over het algemeen zeer positief.

Er kwam enkel nog de opmerking dat het niet duidelijk was te zien wanneer een speler een schaap had gevangen en niet bv. een speler die er vlak naast staat.

Daarom hebben we directe output van de gebeurtenissen toegevoegd. Dit d.m.v. enkele tekstlijnen bovenaan het scherm, waar we boodschappen van het spel aan de spelers kunnen tonen, zoals bv. de melding dat een speler een schaap heeft gevangen.

3.2. ServerGUI

ServerGUI is de interface tussen het spel en de spelbeheerder achter de machine waarop de server draait. Ze biedt functionaliteit aan om een spel aan te maken met bepaalde parameters (zie 2.2 en Figuur 6). Tijdens het spel zijn alle spelers zichtbaar op de bijhorende map (zie Figuur 7). Ook de locatie van het schaap, bommen en schatkisten zijn hierop zichtbaar. De score wordt voor elke speler op een tabel getoond naast de map¹.

ServerGUI werd ontworpen om gemakkelijker te kunnen debuggen en om snel een spel te kunnen aanmaken met bepaalde parameters. Aangezien het niet de bedoeling is deze beschikbaar te maken voor de spelers, is ze ook niet onderworpen aan proefpersonen.

¹ In het huidige ontwerp worden er op de server, buiten de score, geen andere statistieken bijgehouden.

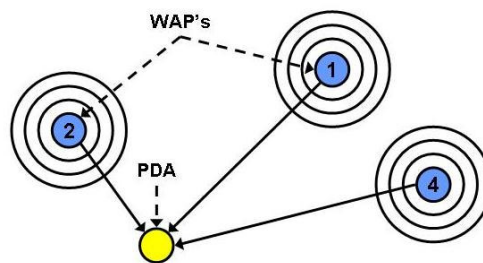
Hoofdstuk 4

Plaatsbepaling

In dit hoofdstuk bekijken we de gebruikte plaatsbepalingsmethode, die gebaseerd is op een *particle filter* die gebruik maakt van de *p-previous-k-nearest-neighbours* methode. Om deze complexe methode goed te begrijpen bekijken we eerst in 4.1 de *fingerprinting* methode waarop de *k-nearest-neighbours* methode is gebaseerd. In 4.2 bekijken we dan de stap naar de *k-nearest-neighbours* methode zelf en in 4.2.1 wordt de hiervan gebruikte uitbreiding toegelicht. Vervolgens bekijken we in 4.3 het meest belangrijke deel van de gebruikte methode, nl. de *particle filter*. Als laatste bekijken we in 4.4 de ondernomen experimenten en de daaruit volgende optimalisaties.

4.1. Fingerprinting methode

De fingerprinting methode is een signaal-gebaseerde methode, dit wil zeggen dat de methode enkel gebruik maakt van de ontvangen signaal-sterkten van de verschillende *Wireless Access Points* (WAP's) en een bijhorende *radio map*. Deze laatste is een map van het gebouw waarop bijkomende attributen zijn geplaatst, we bekijken de *radio map* in 4.1.1 in meer detail.



Figuur 10 Het maken van een *fingerprint / scan*.

In deze methode worden een aantal plaatsen van de *radio map* opgemeten. Onder dit opmeten verstaan we het opmeten van de ontvangen signaalsterkten van de verschillende WAP's op deze plaats. Deze signaalsterkten per WAP horende bij een bepaalde plaats noemt men een *fingerprint*, omdat ze deze plaats uniek kenmerkt (zie Figuur 10). De gezochte plaats wordt dan benaderd door de dichtstbijzijnde fingerprint. Hoe men de dichtstbijzijnde fingerprint bepaalt staat compleet vrij en leidt zodoende op verschillende interpretaties (zie Appendix B). Uit onze experimenten blijkt dat de afstandfunctie gedefiniëerd door de *niet-lineaire transformatie: exponentiël 2* samen met de 2- of *oneindig-norm* de beste is. We bekijken in 4.1.2 de *fingerprint* in meer detail en in 4.1.3 wat we verstaan onder een *scan* van de omgeving.

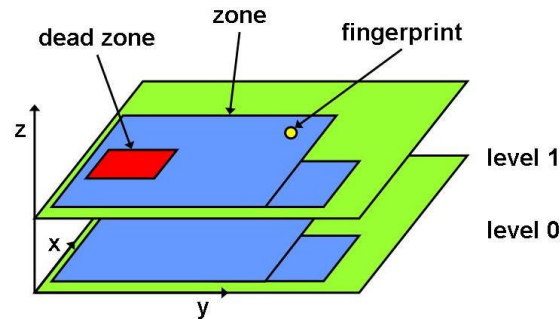
4.1.1. Radio Map

Onder de *radio map* verstaan we het volledige gebied waarbinnen we geïnteresseerd zijn om aan plaatsbepaling te doen. In ons ontwerp hebben we hiervoor een flexibele en commerciële structuur opgebouwd. Hierin bestaat de *radio map* uit *levels*¹ die vervolgens weer zijn opgebouwd uit *zones*² (zie Figuur 11). Een *zone* is een gebied waar de speler zich kan bevinden en kan ook *dead zones* bevatten; zones waar een speler zich onmogelijk kan bevinden omwille van obstakels (bv. tafels, kasten, ..). Op de *radio map* bevinden zich ook attributen, zoals bv. *fingerprints*³.

¹ In de huidige spel-opstelling

² In het huidige ontwerp zijn enkel rechthoekige *zones* uitgewerkt.

³ In het huidige ontwerp bevinden zich op de *radio map* enkel *fingerprints*.



Figuur 11 Structuur van de *radio map*¹.

4.1.2. Fingerprint

Een *fingerprint* bevindt zich op de radio map en heeft bijgevolg coördinaten (x, y, z) . Een *fingerprint* bevat ook voor elke omliggende WAP een set van metingen¹. In plaats van de volledige sets bij te houden distilleren we hieruit een gaussische verdeling waarvan enkel de parameters worden opgeslagen.

4.1.3. Scan

Een *scan* wordt gemaakt wanneer men de plaats van de PDA wil bepalen. De *scan* maakt voor elke omliggende WAP een set van metingen^{1,2}. Hiervan worden enkel de parameters van de gaussische verdeling doorgestuurd naar de server. Deze zoekt dan vervolgens naar de dichtstbijzijnde *fingerprint* op de *radio map* en geeft de coördinaten van deze *fingerprint* als oplossing terug.

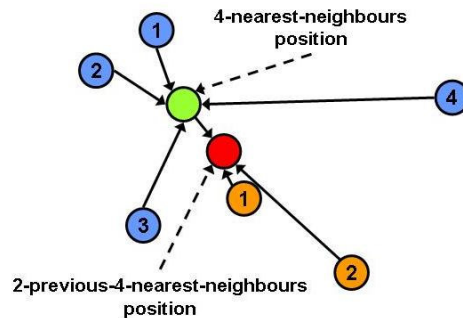
4.2. K-nearest-neighbours methode

Het verschil van *k-nearest-neighbours* methode met de *fingerprinting* methode is dat we nu niet enkel de dichtstbijzijnde fingerprint maar de *k* dichtstbijzijnde

¹ Indien een WAP niet opgemeten wordt wordt de signaal-sterkte ervan op -90 dB gezet.

² In het huidig ontwerp wordt er bij de scan eig. maar 1 meting genomen om de scansnelheid hoog te houden. In dit geval is de enige zinvolle parameter het gemiddelde van de verdeling dat gelijk is aan de meting zelf.

fingerprints nemen. De gezochte plaats wordt dan benaderd door het gewogen midden van de locaties van deze *fingerprints* (zie Figuur 12).



Figuur 12 P-previous-k-nearest-neighbours methode met $p = 2$ en $k = 4$.

4.2.1. Uitbreiding: P-previous-k-nearest-neighbours methode

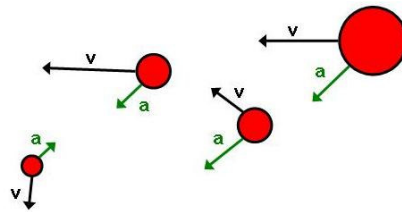
Een uitbreiding op de *k-nearest-neighbours* methode, de *p-previous-k-nearest-neighbours* methode die in het huidige ontwerp gebruikt wordt, maakt ook bijkomend gebruik van de p voorgaande locaties. De gezochte plaats wordt dan benaderd door het gewogen midden van de voorgaande locaties met de plaats berekend a.d.v. de *k-nearest-neighbours* methode (zie Figuur 12).

4.3. Particle Filter

Het belangrijkste deel in de plaatsbepalingsmethode is de *particle filter*. Kort samengevat stelt deze een discretisatie van de kansdichtheidsfunctie van de plaats van de speler over de *radio map* op. In 4.3.1 bekijken we wat we verstaan onder een *particle* en in 4.3.2 nemen we het algoritme onder de loep.

4.3.1. Particle

Een *particle* is een gemodelleerde speler op de *radio map* die een min of meer realistische beweging volgt (zie Figuur 13). Bovendien heeft een *particle* een eigen gewicht. Dat gewicht drukt uit wat de kans is dat het particle zich op de plaats van de speler bevindt. De locatie van elke *particle* wordt aangepast tijdens de predictie-fase d.m.v. een predictor en het gewicht ervan wordt geüpdate in de update-fase.



Figuur 13 *Particles* in beweging met bijhorend gewicht.
(Beweging is voorgesteld door de snelheids- en versnellings-vector.
Gewichten zijn voorgesteld via de groottes van de *particles*.)

4.3.2. Algoritme

De structuur van het algoritme bestaat uit 4 fasen: een resampling-, een predictie-, een update- en normalisatie-fase (zie Figuur 14). We bespreken elk van de fasen verder in detail.

```

while (playing) {
    /* Als het aantal particles met een relevant gewicht te klein is geworden moeten
    we resampelen.*/
    if (getEffectiveSampleSize() < minimumSampleSize)
        resample()

    /*Predictie-fase, de positie van elk particle wordt aangepast.*/
    for each Particle in particle[]
        particle[i].position = prediction(particle[i])

    /*Er komt een nieuwe meting binnen.*/
    measurement = takeMeasurement()

    /*Het gewicht van elk particle wordt aangepast gegeven de nieuwe meting.*/
    for each Particle in particle[]
        particle[i].weight = updateWeight(particle[i], measurement)

    /*Normalisatie-fase, de som van alle kansen moet sommeren tot 1.*/
    for each Particle in particle[]
        particle[i].weight = particle[i].weight / getSumOfAllWeights()
}

```

Figuur 14 Pseudocode voor de *particle filter*.

• Systematic Resampling

Het basisidee van resampelen is het elimineren van *particles* met kleine gewichten en focussen op *particles* met grotere gewichten. We gebruiken in ons ontwerp *Systematic Resampling* (). Het is niet-deterministisch, eenvoudig te implementeren en vergt maar $O(M)$ werk met M het aantal *particles*.

Het is niet zinvol om elke iteratie te resampelen. Om te bepalen wanneer dit moet gebeuren berekenen we de *Coëfficiënt of Variation* (cv_t^2 , zie Vergelijking 1) d.m.v. $w_t(i)$, het gewicht van het i^{de} *particle*. Hiermee wordt dan de *Effective Sample Size* (ESS, zie Vergelijking 2) berekent.

$$cv_t^2 = \frac{\text{var}(w_t(i))}{E^2(w_t(i))} = \frac{1}{M} \sum_{i=1}^M (Mw_t(i) - 1)^2$$

Vergelijking 1 *Coëfficiënt of Variation*.

$$ESS_t = \frac{M}{1 + cv_t^2}$$

Vergelijking 2 *Effective Sample Size*.

Uit onze experimenten is gebleken dat we best resamplen op het moment dat de ESS 50% bedraagt.

- **Predictor**

Zonder een realistisch bewegingsmodel is de *particle filter* niet volledig. De beweging van de speler zorgt ervoor dat de kansdichtheidsfunctie mee moet bewegen. Wanneer een speler duidelijk in een bepaalde richting aan het lopen is dan moet de functie ook in die richting meebewegen. Men zou kunnen zeggen dat random bewegen voldoende is omdat er toch overal wel een *particle* zal zijn. We willen echter dat het particle met het grootste gewicht zich naar de plaats van de volgende meting verplaatst zodat we meer zekerheid omtrend de plaats hebben. Onze experimenten tonen aan dat een realistisch bewegingsmodel verantwoordelijk is voor een (beperkte) extra nauwkeurigheid.

In concreto heeft elk *particle* zoals beschreven in 4.3.1 een positie en een snelheidsvector (zie ook Figuur 13). Naargelang de snelheid van het *particle* zal er een andere kansverdeling zijn voor de versnelling. Uit onze experimenten blijkt dat *particles* met een heel kleine en heel grote snelheid best een heel willekeurige versnelling krijgen. *Particles* met een gemiddelde snelheid krijgen best een kleine versnelling. Dit is realistisch in de zin dat personen die op dit moment stilstaan een grote kans vertonen om plots te vertrekken. Daarentegen personen die zeer snel vooruitgaan zullen waarschijnlijk gaan afremmen. Personen die tegen een matig tempo wandelen zullen waarschijnlijk dat tempo aanhouden. Alle *particles* hebben een lichte voorkeur om af te remmen.

Ook de tijd tussen verschillende metingen wordt bijgehouden en in rekening gebracht. Wanneer door een delay op het netwerk plots enkele seconden geen

meting meer doorkomt spreekt het voor zich dat de speler verder kan zijn verplaatst dan wanneer er zoals gewoonlijk elke 600 à 700 ms een nieuwe meting binnenkomt.

We hebben pogingen ondernomen om een adaptieve predictor te gebruiken die zijn bewegingsmodel aanpast aan de speler. Coëfficiënten worden dan op een genetische manier per *particle* aangepast en enkel de realistische *particles* overleven. In onze experimenten zijn de resultaten echter niet beter dan met het gewone model.

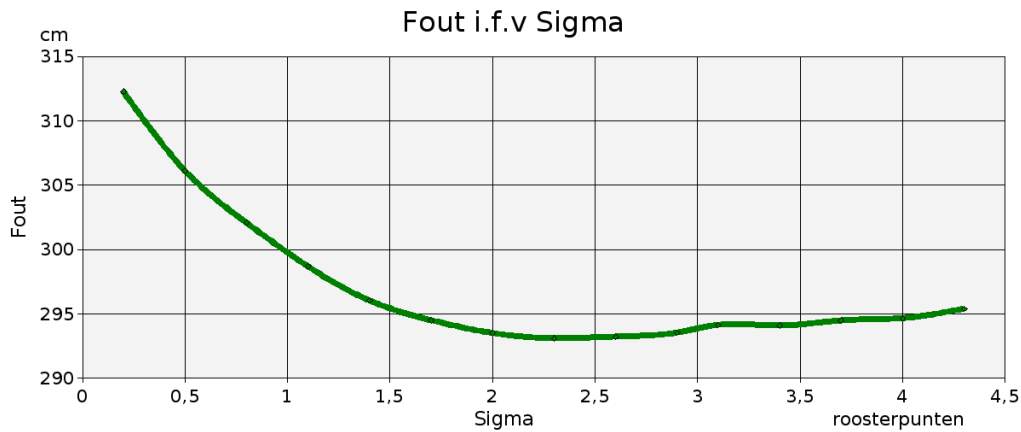
- **Update**

De taak van de update methode is te zorgen dat de waarschijnlijkheid die aan elke *particle* wordt gegeven zo realistisch mogelijk is. De update methode gebruikt de plaatsbepaling van de *p-previous-k-nearest-neighbours* methode. Rond die plaats draait de klokcurve van de normaalverdeling (zie Vergelijking 3).

$$P(x^{k+1} | z) = \frac{1}{\sqrt{2\pi}\sigma_d} e^{\frac{-(d_i)^2}{2\sigma_d^2}}$$

Vergelijking 3 Gebruikte normaalverdeling.

Hierin is x^{k+1} de plaats van het particle na de predictie fase en z is de nieuwe plaats bepaald met de *p-previous-k-nearest-neighbours* methode meting. d_i is de afstand tussen deze 2 plaatsen. De parameter σ_d hebben we experimenteel geoptimaliseerd op 2,5 roosterpunten (4,5m, zie Figuur 15).



Figuur 15 Optimalisatie van parameter σ_d .

Particles die terecht komen buiten de kaart of op een *dead zone* (zie 4.1.1) krijgen gewicht nul. We hebben geprobeerd deze onmogelijke plaatsen in het bewegingsmodel mee op te nemen. Uit de experimenten blijkt dat de resultaten beter zijn wanneer we alle beweging toelaten en de gewichten navenant aanpassen.

Voor de correcte *particles* wordt de berekende waarschijnlijkheid vermenigvuldigd met het huidige gewicht van het particle.

- **Normalisatie**

De normalisator berekent de som van alle gewichten van de *particles* en deelt het gewicht van elk *particle* door deze som. Dit voorkomt over- of onderloop.

- **Estimator**

Uit een gigantische collectie *particles* (20.000 in ons huidig ontwerp) moet nog 1 bepaalde plaats worden gevonden. Er zijn een aantal mogelijkheden om dit te doen, nl het beste *particle* (het particle met het grootste gewicht, zie Vergelijking 4), een gewogen gemiddelde van alle *particles* (zie Vergelijking 4), een robuust

gemiddelde (een gewogen gemiddelde van alle *particles* binnen een cirkel rond het beste *particle*, zie Vergelijking 4).

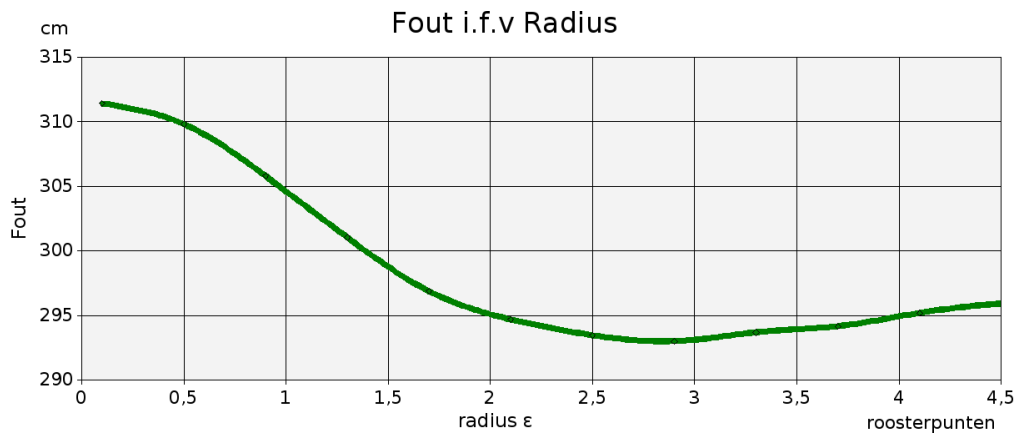
$$\text{best_particle} : \bar{x}_s = x_s^{\max}$$

$$\text{average} : \bar{x}_s = \sum_{i=1}^M x_{s,i} w_i$$

$$\text{limited_average} : \bar{x}_s = \sum_{i=1}^M x_{s,i} w_i : |x_{s,i} - x_s^{\max}| \leq \varepsilon$$

Vergelijking 4 Verschillende implementaties voor de estimator.

Deze laatste is de beste keuze en is met een radius ε van 2,4 rasterpunten (4,32m) optimaal (zie Figuur 16).



Figuur 16 Optimalisatie van de radius ε .

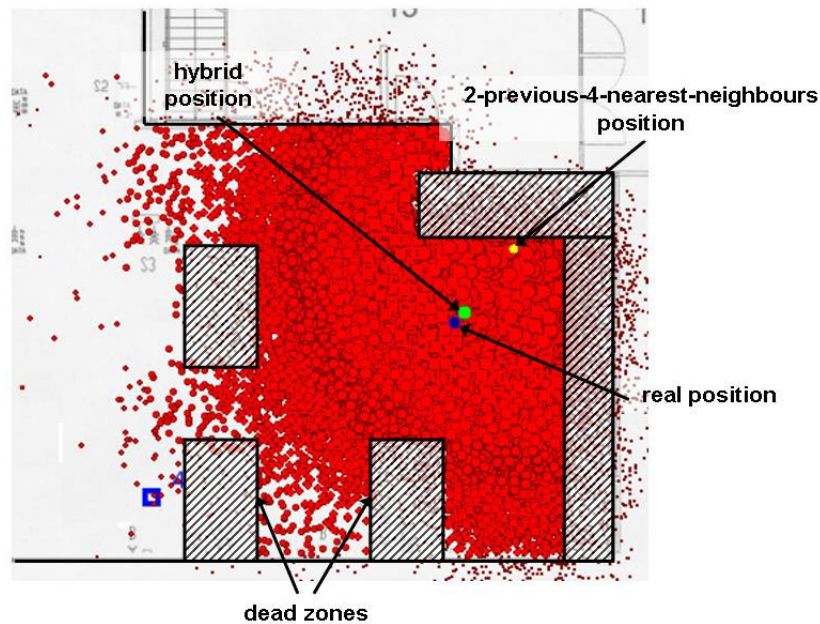
Het is belangrijk niet alleen de gemiddelde fout te minimaliseren maar ook de radius klein genoeg te houden. Als deze te groot is zal het moeilijker zijn om plaatsen op de rand van de *radio map* te kunnen bereiken. Daarom verkleinen we de radius ook at runtime wanneer het grootste particle zich in de buurt van de randen van het gebouw bevindt.

4.4. Experimenten

In dit onderdeel lichten we onze keuze voor de plaatsbepalingsmethode toe met de resultaten van een uitgebreid arsenaal van experimenten. We hebben reeds al enkele resultaten bekeken bij de *particle filter* (zie 4.3).

- **Ideale experimenten**

Vorig semester konden we reeds met tests aantonen dat onze toenmalige plaatsbepaling in het ideale geval (met gegevens uit de metingen van de fingerprints) een gemiddelde fout had van 2,2 meter. De tests 'in het ideale geval' gebeuren nog steeds met de signaalsterktes uit de metingen van de fingerprints, maar nu volgens een parcours dat de speler realistisch had kunnen volgen. Voor de oude methode is de fout nog steeds 2,2 meter. Met een particle filter echter hebben de de gemiddelde ideale fout kunnen terugbrengen tot gemiddeld 70cm! Spijtig genoeg wordt dit theoretische minimum nooit bereikt in realiteit.



Figuur 17 Grafische voorstelling van psuedo-realistische test.

Om toch realistisch te kunnen testen hebben we een pseudo-realistische test ontwikkeld. Door op een GUI op de PDA een bepaald parcours te volgen in de zaal (langs randen, snel, traag, met veel variatie, ...) kan de PDA een meting linken aan een exacte plaats in de zaal. Hiermee kunnen we de gemiddelde fout echt gaan minimaliseren en er zeker van zijn dat het in realiteit een betere plaatsbepaling gaat geven.

Om de tests mooi te visualiseren voorzien we ook een GUI die alle particles kan afbeelden samen met de plaats van de meting het resultaat van de k-nearest methode (zie Figuur 17).

De grootte van de particles duidt hun gewicht aan. Het is duidelijk dat particles die buiten de muren of op tafels gaan gewicht nul krijgen.

De *p-previous-k-nearest-neighbour* methode is zeer snel en kan op 4ms aan plaatsbepaling doen. De *particle filter* is zo traag als het aantal *particles* dat je gebruikt. Voor 20.000 particles zit je al aan de grens van het zinvolle en duurt de plaatsbepaling nog maar 50ms. Uitvoeringstijd is dus geen probleem.

Het optimaliseren van de plaatsbepaling was een iteratief proces van lange adem. Op tientallen servers in de PC klas hebben we tientallen uren simulaties gedaan van plaatsbepaling met alle mogelijke coëfficiënten, instellingen en methodes. Enkele resultaten werden reeds besproken. Ook de gewichten van de *p-previous-k-nearest-neighbours* methode en de gewichten van de vorige locaties werden zo geoptimaliseerd. Ter illustratie tonen we één van de berekende tabellen in Appendix D.

Hier werden 256 verschillende combinaties van gewichten getest. Kennelijk geeft een combinatie van gewichten voor de burens met index 5 en gewichten van de vorige posities met index 5 de beste plaatsbepaling. Dit is maar één van 20 tabellen die we zo hebben opgesteld.

Ook het bewegenismodel, het aantal particles, de minimum Sample Size en nog veel meer instellingen zijn zo geoptimaliseerd. In totaal hebben we zo onze gemiddelde fout in de tabellen van 3 tot 5 meter kunnen terugdringen tot 2,9 meter. Hiervan mag je nog eens een kleine meter aftrekken door de kleine fout die wordt gemaakt tijdens het aanmaken van de testdata. We kunnen dus met vrij grote zekerheid en gesteund door berekeningen zeggen dat voor onze plaatsbepaling de gemiddelde fout in realiteit ongeveer 2 tot 2,5 meter bedraagt. Dit veronderstelt dat de gebruiker stilstaat of tegen een rustig tempo wandelt. Wanneer de gebruiker snel loopt mag je hier een paar meter bijtellen.

Bovendien zijn wij er van overtuigd dat we de grenzen hebben bereikt van wat mogelijk is met WiFi signalen. Door het grondig optimaliseren zijn we zeker dat het onmogelijk is een kleinere fout te halen wanneer je een particle filter of een k-

nearest-neighbour methode gebruikt. We vermoeden zelfs dat een fout van 2 meter inherent is aan het probleem.

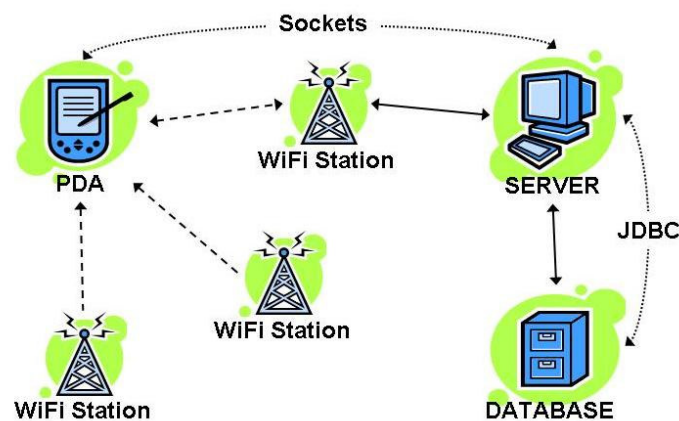
Hoofdstuk 5

Ontwerp

In dit hoofdstuk bekijken we het ontwerp van *Sheephunt* op meerdere niveaus, van een abstract in 5.1 naar een gedetailleerd niveau in 5.3, waarin we de implementaties van de verschillende onderdelen toelichten.

5.1. Machine niveau

Het eerste niveau waarop we *Sheephunt* bekijken is op het machine niveau (zie Figuur 18). De PDA's van de spelers nemen zo snel mogelijk metingen van de signaal-sterkten van de nabijgelegen WAP's¹. Deze metingen worden doorgegeven aan de server via *Sockets* over een draadloze verbinding tussen de PDA's en een WAP. De server bepaalt hieruit de plaatsen van PDA's. De database wordt bij de initialisatie van de server eenmalig ingeladen m.b.v. JDBC.



Figuur 18 Machine niveau van *Sheephunt* voor één enkele PDA.

¹ In het huidige ontwerp wordt er eig. maar 1 meting genomen om de scansnelheid hoog te houden.

5.2. Conceptueel niveau

Het hierop volgende niveau dat we bekijken is het conceptuele niveau (zie Figuur 19). Deze toont de verschillende onderdelen in de machines. Hier volgt een klein overzicht van de onderdelen en hun bijhorende functie-omschrijving.

- **ClientGUI**

Is de interface tussen het spel en de speler op de PDA. Deze is verantwoordelijk voor de gebruiksvriendelijke grafische communicatie van het spel met de spelers.

- **ClientCore**

Is het centrale onderdeel op de PDA. Deze is verantwoordelijk voor de correcte werking van de onderdelen op de PDA.

- **WiFiScanner**

Neemt metingen van de signaal-sterkten van de nabijgelegen WAP's.

- **Com. (Communication)**

Staat in voor de correcte communicatie tussen de PDA's en server.

- **ServerCore**

Is het centrale onderdeel op de server. Deze is verantwoordelijk voor de correcte werking van de onderdelen op de server.

- **Game**

Is het centrale onderdeel voor *Sheephunt*. Deze is verantwoordelijk voor een correct spelverloop.

- **PositioningSystem**

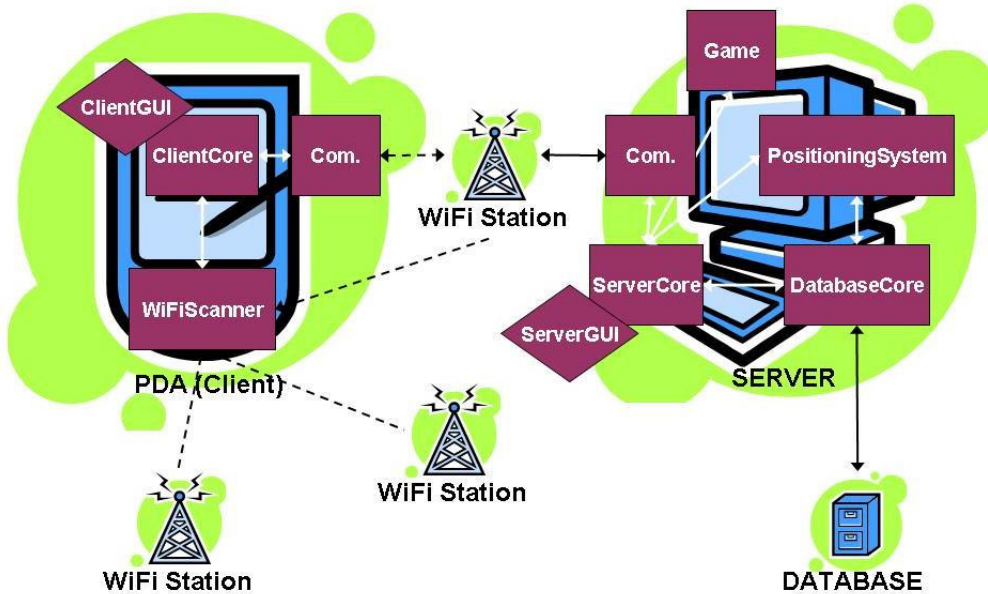
Bepaalt a.d.h.v. de metingen en bijhorende informatie in de database de plaats van de PDA op de *radio map*.

- **DatabaseCore**

Deze staat in voor de correcte communicatie met de database.

- **ServerGUI**

Is de interface tussen het spel en de spelbeheerder achter de machine waarop de sever draait. Deze is verantwoordelijk voor de grafische communicatie met de spelbeheerder.



Figuur 19 Conceptueel niveau van *Sheephunt* voor één enkele PDA.

Aan de hand van dit niveau delen we *Sheephunter* op in 5 onderdelen. We bespreken deze later in detail.

- **PositioningSystem:** *PositioningSystem*.
- **GUI:** *ClientGUI*, *ClientCore* en *ServerGUI*.
- **Communication:** *Communication* en de bijhorende communicatie tussen de PDA's en de server via *Sockets*.
- **Server:** *ServerCore*, *DatabaseCore* en de database zelf.
- **Game:** *Game*.

5.3. Implementatie niveau

Het laatste, meest gedetailleerde niveau dat we bekijken is dat van het *Unified Modeling Language* (UML) diagram, nl. het implementatie niveau. Deze toont de implementatie van de conceptuele onderdelen. We bespreken de implementatie van de onderdelen zoals hierboven beschreven.

5.3.1. PositioningSystem

De plaatsbepaling is volledig in *Java* geïmplementeerd. Om de verscheidene plaatsbepalingsmethodes gemakkelijk door elkaar te kunnen gebruiken hebben we uitvoerig gebruik gemaakt van *interfaces*. Dit maakt het zeer eenvoudig om een nieuw idee snel in de praktijk te kunnen brengen. In *Sheephunt* heeft iedere speler zijn eigen plaatsbepalingsmodule.

De structuur is ontworpen met het oog op orde, uitbreidbaarheid en efficiëntie. Er is een grote hoeveelheid klassen die alternatieve plaatsbepalingsmethodes implementeren die in het huidige ontwerp niet meer in gebruik zijn. We hebben niet veel gebruik gemaakt van *design patterns* en op bepaalde plaatsen is er geen goede afscherming van de variabelen, dit omwille van efficiëntie-redenen.

5.3.2. GUI

Het GUI onderdeel bestaat uit 2 grote delen, nl. de client (PDA) en de server. We bespreken hieronder elk deel in detail.

- **Client (PDA)**

De client is opgebouwd uit 3 delen, nl *ClientCore*, de *Updater* en *ClientGUI*.

ClientCore is de logica achter de client, hij stuurt alle data stromen naar de juiste ontvanger, en stuurt ook de *ClientGUI* aan. Eigenlijk is *ClientCore* niet meer als een eindige toestandsmachine. Hij kent 4 basistoestanden: login, waiting to start game, game en End. *ClientGUI* wordt vanzelfsprekend direct opgestart vanaf het programma begint. De *Updater* wordt opgestart zodra er een verbinding is met de server. *ClientGUI* draait in dezelfde thread als de core. De *Updater* wordt in een aparte thread opgestart.

De *Updater* start op vanaf er een verbinding met de server is verwezenlijkt. Zijn taak is de wifi meeting te nemen en deze meting samen met eventuele acties van de gebruiker door te sturen naar de server en de locatie van de speler samen met game-informatie terug te ontvangen en door te spelen naar de core, die deze dan verder verwerkt. Deze werkt in een aparte thread omdat dit de meest tijdsconsumerende taak is op de pda. Door deze in een aparte thread te draaien, kunnen we in de gui een veel hogere refresh rate verwezenlijken. Een update van een wifi-meting en update duurt ongeveer 400 tot 600 ms. Dit zou betekenen dat als we dit in dezelfde thread als de gui lieten verlopen de gui een refresh rate zou hebben van ongeveer 2 hz. Dit is natuurlijk te traag om een vlotte gui te bouwen. Ook het feit dat soms bij zwaar netwerk verkeer de update nog langer duurt, is ook een argument om de updater in een aparte thread te laten verlopen. Met deze strategie hebben we kunnen verwezenlijken dat de gui gerefreshed wordt met een rate van ongeveer 10hz. Dit is nog altijd niet veel, maar uit gebruik blijkt dat dit voldoende is om een aangename ervaring te verwezenlijken.

De gui is natuurlijk de aansturing van het gebruikersscherm. Hiervoor werd de swt toolkit gebruikt. Voor elke toestand in de core, komt er 1 scherm in de gui overeen. Tussen de login en de wait toestand is er nog een connect scherm. We hebben gekozen om deze in dezelfde thread te laten draaien als de core, aangezien er geen nood was voor meerdere threads. Het was perfect mogelijk om een thread te houden, en toch alle gewenste functionaliteit te hebben. Het bijkomende voordeel van niet met threads te werken was de eenvoud van het

uiteindelijke programma. Door de enkele thread moesten we geen rekening houden met eventuele synchronisatie problemen, wat het coderen veel vlotter liet verlopen.

Server gui implementatie:

De servergui bestaat uit één groot scherm met één map en een knop om de server te starten en enkele veldjes om de server te configureren voordat hij wordt opgestart. We hebben deze geïmplementeerd, dat als er op de knop wordt gedrukt, de server zelf wordt opgestart. De server draait in dezelfde thread omdat er eigenlijk vanaf de server is opgestart, geen gebruikersinteractie meer is. De servergui wordt volledig aangestuurd vanuit de servercore. Telkens als deze een update ontvangt van een van de clients, wordt deze verwerkt in de game, en vervolgens wordt de gui verwittigd, die dan de nodige informatie uit de core opvraagt en het scherm update.

5.3.3. Communication

Op het einde van vorig semester werd gesproken over de mogelijkheid om over te schakelen van *Remote Method Invocation* (RMI) naar *Simple Object Access Protocol* (SOAP). Dit omdat RMI eigenlijk te uitgebreid was voor onze doeleinden en dat we het moeilijk werkende kregen (en te hielden) tussen een PDA en een desktop machine op een ander netwerk met een *firewall* ertussen. De communicatie in *Sheephunt* dient er immers enkel toe om strings en arrays hiervan (die de gemaakte meting(en) op de PDA voorstellen) door te sturen. Hiervoor is een krachtig framework als RMI overkill.

Dit semester hebben we besloten om SOAP te implementeren. XML-RPC, een afgeslankte SOAP versie, blijkt uitstekend te voldoen aan onze specifieke noden en is ook een lichtere versie dan SOAP zelf en bijgevolg ook meer geschikt voor gebruik op de PDA.

De communicatie-structuur is ook aangepast. Vorig semester werden met RMI de methodes aangeroepen over het netwerk; wat een mooi concept was, maar in de praktijk heel moeilijk werkte. Voor XML-RPC is een protocol geschreven dat 2 klassen heen en weer stuurt.

- Er wordt tegen maximale snelheid nieuwe metingen gemaakt van de signaalsterktes van de omliggende WAP's op de PDA's. Deze informatie wordt dan gecomprimeerd, zoals reeds uitgelegd in 4.1.3, tot een instantie van *StochasticScan*. Deze wordt dan d.m.v. een parser omgezet in een array van strings en vervolgens doorgestuurd naar de server.
- Het antwoord van de server is een instantie van de klasse *Info*. Deze bevat volgende informatie voor de speler op de PDA:
 - de spelstatus (nl. GAME STARTED, WAITING FOR PLAYERS, GAME BUSY of GAME ENDED),
 - een lijst van de spelers met hun bijhorende score,
 - de huidige plaats op de *radio map* en
 - een lijst met alle zichtbare objecten op de *radio map* met hun bijhorende locatie.

Deze wordt dan opnieuw door een parser omgezet en doorgestuurd.

In een poging om het doorsturen van de metingen te versnellen werd er een buffer geschreven die de verschillende update-requests tijdelijk bijhoudt en doorstuurt van zodra de vorige communicatie afgelopen was. Zo kan er een accuratere plaatsbepaling gebeuren doordat er meer regelmatig WiFi metingen kunnen genomen worden, met intervallen onafhankelijk van de communicatie.

Omdat er 2 processen uit deze buffer lezen en schrijven was er nood aan thread-synchronisatie.

Er zijn tests gedaan met meerdere threads. De metingen gebeurden maar even snel als de communicatie ze kon doorsturen. Dit werkte echter niet efficiënt omdat er te weinig van thread werd gewisseld. Communicatie stuurde ineens

alles uit de buffer door en wachtte dan zeer lang. Een verklaring hiervoor kan zijn dat threads of synchronisatie niet efficiënt is geïmplementeerd in J9, of dat de thread die naar de sockets schrijft een blocking thread is en dat het netwerk tijdens de tests overbelast was en maar af en toe communicatie toeliet van de PDA. De eindconclusie is dat het sequentieel meten en doorsturen het beste is voor onze toepassing, met een meting die binnenkomt op de server gemiddelde om de 600-900 ms.

Omdat de XML-RPC implementatie uiteindelijk toch maar herleid was naar het heen en weer sturen van arrays van strings, besloten we van nog een niveau lager te gaan, en gewoon met Sockets te werken. Dit bleek heel simpel en elegant. Sockets zijn ook de enige vorm van netwerkcommunicatie die al werden behandeld in de opleiding, namelijk in de labo's van het vak computernetwerken. Bij het begin van het tweede semester werd er als bijkomende opgave gegeven dat onze applicatie robuust moest zijn en de communicatie met sockets heeft zijn robuustheid zeker bewezen en werkt voorlopig feilloos.

Het feit dat we 2 keer van communicatieprotocol zijn veranderd zonder één regel te moeten veranderen aan de rest van het project bewijst dat de hele applicatie goed modulair is opgebouwd. De communicatie is dan wel niet objectgeoriënteerd maar het ontwerp van heel de applicatie is er op vooruit gegaan. Het enige nadeel dat we zien is dat de communicatieklassen actief moeten worden onderhouden wanneer er toevoegingen aan het spel gebeuren. Het toevoegen van bijvoorbeeld bommen naar het einde van het project toonde aan dat dit niet zo moeilijk is.

5.3.4. Server

De server bestaat uit twee belangrijke pakketten, namelijk DatabaseCore en ServerCore. Deze laatste zorgt voor de communicatie met de database waarin alle informatie voor de plaats bepaling opgeslagen is, alsook de informatie over

de spelers en de verschillende spelen die gespeeld zijn of bezig zijn. Daarnaast is er ook nog een GUI, zodat men op de server goed het spelverloop kan volgen en de nodige aanpassingen kan maken.

De servercore klasse is de spil van heel de server, alle andere zaken worden hier aangemaakt en gebruikt. Zo zal servercore een databasecore object aanmaken om met de database te kunnen interageren om spelers op te vragen, een spel te starten of te stoppen en om een positioncalculator object aan te maken dat zijn data uit de database haalt. De servercore zal ook nog user objecten aanmaken zodra spelers zich aanloggen en communicatie objecten om met deze speler te kunnen communiceren. Als laatste maakt de servercore ook een spel aan zodra het juiste aantal spelers is verbonden.

Op dit moment is het zo dat dit servercore object aangemaakt wordt door een servergui object. Misschien is dit niet echt de beste manier en zou het beter zijn om dit omgekeerd te doen aangezien het logischer is dat een servercore zijn eigen gui beheerst. Aangezien servergui een extra is die wij op het einde toe hebben ingebouwd, hebben we hier niet lang genoeg bij stilgestaan en hebben we gekozen voor de op dat moment gemakkelijkste oplossing.

- **Database**

Tegenover vorig semester zijn er toch enkele ingrijpende veranderingen gebeurd in verband met de database. Om te beginnen wordt er dit semester geen gebruik meer gemaakt van de tables die verband houden met de applicatie van vorig semester. Meer specifiek zijn dit de tables LOCATIONS, QUESTIONS, ANSWERS, USER_ANSWERS. De tables USER, FINGERPRINTS, FP_MEASUREMENTS worden nog wel gebruikt. De eerste voor het bijhouden van de verschillende spelers, de laatste twee voor de plaatsbepaling.

Het systeem van deze plaatsbepaling is aangepast om nu ook gebruik te maken van radiomaps, radiomap levels en radiomap zones. Hiervoor zijn dan ook drie

tables aangemaakt om deze informatie in de database bij te houden (RADIO_MAPS, RADIO_MAP_LEVELS, RADIO_MAP_ZONES). Als de server nu de plaatsbepaling initialiseert, zal de lokale radiomap worden opgevraagd aan de database, die dan de juiste zones, levels en geassocieerde fingerprints zal inladen in deze radiomap.

Daarnaast zijn er nog twee tables bijgekomen, genaamd GAMES en USER_SCORES. In de GAMES table zal elk spel dat gestart wordt opgeslagen worden. Bij het starten zal de begin tijd worden ingesteld en op het einde zal de eindtijd worden opgeslagen. In de USER_SCORES wordt de score per game voor elke user opgeslagen zodra het spel beeindigt is.

Uiteraard wordt er doorheen heel de database gewerkt met foreign keys, zodat alles in relatie staat met de juiste zaken. Op die manier wordt er ook consistentie behouden in de database. Er kan bijvoorbeeld geen score in de USER_SCORES table staan voor een user of game die niet meer bestaat. Hetzelfde geldt voor radiomap zones, levels en fingerprints.

- **DatabaseCore**

Aan de klasse DatabaseCore is als gevolg van de hierboven vermelde veranderingen, ook een aantal veranderingen aangebracht. De belangrijkste zijn uiteraard de nieuwe manier waarop de data van de plaatsbepaling wordt opgevraagd. Men vraagt nu gewoon een radiomap op en de database zorgt ervoor dat deze radiomap alle benodigde informatie heeft.

Ook de functies die aanwezig zijn voor het spel zijn veranderd. Er zijn functies aanwezig om

een speler op te vragen, een spel te starten en een spel te beeindigen. De eerste functie zal een nieuwe speler aanmaken in de database als deze nog niet bestaat. De functie om een spel te starten zal een nieuw spel in de database

aanmaken en de starttijd instellen. Als laatste is er de functie om een spel te beëindigen en hierbij wordt de eindtijd ingesteld van een spel en de verschillende scores van de spelers opgeslagen.

5.3.5. Game

De game logic bevat alle klassen die het spel en de daarin voorkomende objecten voorstellen en beheren. De centrale Game-klasse stelt een spel voor met daarin alle nodige informatie over de gebruikte map, het huidige schaap, de aanwezige spelers en hun scores, de geplaatste bommen, de actieve VisionPowerUps en de schatkisten in het spel.

Een schaap wordt voorgesteld door de Sheep-klasse die overerft van de PositionObject-klasse. Een schaap heeft enkel een positie karakteristiek en krijgt zijn coördinaat van een random position generator (RandomPositionGenerator-klasse) die door de Game-klasse wordt bijgehouden.

Elke speler wordt voorgesteld als een object van de User-klasse, die de coördinaat-karakteristiek overerft van de PositionObject-klasse en die de Owner-interface implementeert zodat hij eigenaar kan zijn van objecten van de ObjectWithOwner-klasse (bom, power-up,...) . Elke speler heeft ook een info-object dat informatie bijhoudt over het spel en de speler zelf (score, bom, power-up, ...) en dient om de communicatie te verzorgen tussen server en client.

Bommen zijn objecten van de Bomb-klasse die over erft van de ObjectWithOwner-klasse. Deze omvat de coördinaat-karakteristiek via de PositionObject-klasse en definieert ook objecten die een geldige eigenaar moeten hebben (momenteel user of schatkist). Zo kunnen later makkelijk andere power-ups of items toegevoegd worden. Daarnaast houden bommen ook informatie bij over het feit of ze al dan niet ontploft zijn.

Ook de VisionPowerUp-klasse erft van de ObjectWithOwner-klasse en heeft dus dezelfde coördinaat en eigenaars-karakteristiek als bommen. Ook bevat deze klasse informatie over de kracht van de power-up en over het feit of de power-up al dan niet gebruikt werd.

Schatkisten zitten in de TreasureChest-klasse die net zoals de User-klasse de coördinaat-karakteristiek via de PositionObject-klasse en de eigenaar-karakteristiek via de Owner-interface krijgt. Verder houdt een schatkist ook informatie bij over zijn inhoud waarvan hij eigenaar is. Als de inhoud uit de schatkist genomen wordt (en dus overgedragen aan een user), wordt er een nieuwe inhoud voor de schatkist gegenereerd via de singleton-klasse TreasureChestContentGenerator die random een nieuw ObjectWithOwner-object genereert.

Door het gebruik van de Owner-interface en de ObjectWithOwner-klasse kunnen eenvoudig nieuwe eigenaars bv een pc-speler met een zekere intelligentie of nieuwe items bv VisionPowerDown's toegevoegd worden in de toekomst.

Het feit dat de TreasureChestContentGenerator-klasse een singleton klasse is, zorgt ervoor dat doorheen heel de server en verschillende games er maar 1 enkele instantiatie gebruikt wordt om een nieuwe inhoud te genereren voor geopende schatkisten. (de RandomPositionGenerator-klasse is afhankelijk van de gebruikte RadioMap van het spel en kan dus niet geconverteerd worden tot een singleton klasse aangezien er op 1 server verschillende games op een verschillende locatie kunnen gespeeld worden.)

Hoofdstuk 6

Proces

In dit hoofdstuk tonen we het proces dat bij het ontwerp van *Sheephunt* is gevolgd. In 6.1 bekijken we de moeilijkheden die we zijn tegenkomen; wat we hebben bijgeleerd van dit project wordt besproken in 6.2 en in 6.3 blikken we terug op het team-gebeuren.

6.1. Moeilijkheden

Als men werkt aan een groot project zoals dit, komt men uiteraard moeilijkheden tegen. Deze komen niet alleen voor doordat men werkt in een groot team, maar ook doordat men werkt met onbekende hardware. Een voorbeeld hiervan is de pda. Deze deed niet altijd hetgeen men zou verwachten, gaande weg leert men daarmee omgaan en dit te anticiperen.

Daarnaast waren er met deze PDA's ook problemen met de batterij, af en toe kwam het voor dat de batterij ofwel al helemaal leeg was ofwel bijna leeg was op het moment dat we deze in handen kregen. Daarnaast was het naar het einde toe soms lastig om genoeg PDA's te vinden aangezien elk groepje zoveel mogelijk tests wil doen. Op zo'n moment is het handig als er extra pda's aanwezig zouden zijn.

Een ander probleem dat we ondervonden hebben is het wifi netwerk. Af en toe ging dit ongeloofelijk traag doordat er een groot aantal personen met laptops aanwezig waren in de pc-klas. Dit zorgde voor een zware belasting op het netwerk waardoor de throughput van de verbinding tussen server en pda soms zeer laag was.

In het ontwerpen van de software moet men ook leren omgaan met bepaalde zaken. Zo was het nodig om bij SWT enkele dingen grondig te onderzoeken voor er verder kon gegaan worden met het implementeren. Dit komt natuurlijk voor bij alle nieuwe dingen die men probeert.

Ook met RMI hebben we de nodige problemen gehad, maar hier zijn we snel vanaf gestapt door eerst over te schakelen naar XML-RPC en daarna naar *Sockets*. Eenmaal dit geïmplementeerd was, was er voor de communicatie niet echt meer een probleem.

Moeilijkheden:

Moeilijkheid 1: Een pda werkt soms niet helemaal zoals je verwacht van een gewone pc. Bepaalde instructies herkent hij niet, en ook het opstarten verloopt anders dan normaal. Dit gaf in het begin wat puzzelproblemen om de juiste link file aan te maken, maar na een tijd werkte dit wel.

Moeilijkheid 2: Een gui schrijven is soms zeer complex. Swt werkt eigenlijk in een andere thread om de gui up te daten. Hierdoor is het soms moeilijk in te schatten welk effect bepaalde wijzigingen soms hebben op het vernieuwen van het scherm. Dit gaf als probleem dat het scherm soms zeer langzaam ververs werd. Dit hebben we opgelost door een methode op te roepen die de pas verder gaat als er geen openstaande gui-opdrachten meer zijn.

6.2. Wat hebben we ervan geleerd?

Doorheen het jaar hebben we geleerd om samen te werken in een groot team en dat communicatie hierbij van groot belang is. We hebben dan ook geprobeerd om zoveel mogelijk met elkaar te communiceren. Enerzijds door elke sessie te starten en te eindigen met een vergadering. Op die manier was iedereen op de hoogte van wat elk subteam deed. Tussen de verschillende subteams werd ook

veel gecommuniceerd om ervoor te zorgen dat we in de integratiefase zo weinig mogelijk problemen zouden hebben. Dit bleek van in het begin een zeer goede strategie te zijn aangezien we weinig problemen hebben ondervonden tijdens de integratie.

Dat er weinig problemen waren tijdens de integratie, is ook te wijten aan het feit dat elk onderdeel op zich goed functioneert. Dit is het gevolg van het uitvoerig testen van alle geschreven code. Dit is iets wat we hebben geleerd gedurende deze sessies, het is belangrijk om veel testen te schrijven en de code uitvoerig na te kijken. Dit kan alleen maar zorgen voor minder problemen achteraf.

Als laatste hebben we ook geleerd dat het inzetten van personen waar nodig is, ervoor kan zorgen dat alles snel klaar geraakt. In het begin werd aan iedereen een bepaalde taak gegeven binnen een bepaalde subgroep. Al snel bleek dat bepaalde taken sneller klaar waren dan anderen. Dan kan het zeer handig zijn om personen die klaar zijn met hun taak in te zetten bij andere taken. Door op die manier flexibel om te gaan met de taken en de personen, hebben we ervoor gezorgd dat alles sneller klaar was dan verwacht.

Deze zaken zullen ons zeker verder helpen bij volgende projecten en we zullen die dan ook later zeker nog veel toepassen.

6.3. Werken in team

Bij het werken in een groot team is het belangrijk dat het grote project wordt opgedeeld in kleinere taken waarvoor dan één of meerdere personen verantwoordelijk zijn. Hierdoor kan er sneller ontwikkeld worden.

Hierbij kunnen zich natuurlijk wel problemen voordoen. Zo is het soms onduidelijk wat een ander subteam aan het doen is. Dit kan voor problemen zorgen later.

Zoals eerder vermeld hebben we dit opgevangen door uitvoerig te communiceren met iedereen.

Daarnaast is het werken in een groot team ook goed voor de motivatie. Als een persoon het even niet ziet zitten, is er altijd iemand die klaarstaat om een motiverende uitleg te geven en ervoor te zorgen dat die persoon het weer ziet zitten. Daarnaast voelt ieder teamlid ook een verplichting tegenover het team om zo goed mogelijk werk te leveren.

Wat ook een sterke motivator is, is de sfeer in het team. Iedereen komt goed overeen met iedereen, en dit zorgt ervoor dat iedereen zich goed voelt binnen het team. Daardoor ziet er ook niemand tegenop om naar de sessies te komen, aangezien het toch altijd een leuke omgeving is.

Spijtig genoeg is het werken in team niet altijd zo positief. Af en toe kunnen er wel eens negatieve zaken naar boven komen. Zo kan het af en toe voorkomen dat je moet wachten op een ander subteam voor je echt verder kan. Het is dan wel mogelijk om allerlei test en dummy klassen te schrijven, maar uiteindelijk moet je toch wachten tot het andere subteam een bepaald punt heeft bereikt om je code in de echte omgeving te testen. Dit hebben we proberen opvangen door zoveel mogelijk de verschillende subteams op elkaar af te stemmen en door extra mensen in te zetten waar nodig.

Dankwoord

Bronnen

Appendix A

Werkbelasting

In deze appendix bekijken we hoe het werk efficiënt verdeelt is geworden over het team.

In eerste instantie zijn de grote onderdelen van *Sheephunt*, zoals ze beschreven worden in 5.2, toegewezen aan één of meerder teamleden.

- **PositioningSystem:** *Adriaensen Chris & Van Den Broeck Guy*
- **GUI:** *Balog Daniel & Maerien Jef*
- **Communication:** *Van Den Broeck Guy & Van Ranst Wouter*
- **Server:** *Dellaert Philippe*
- **Game:** *Cuypers Cedric*

Vervolgens bekijken we per teamlid de belangrijkste taken die deze heeft gedragen bij *Sheephunt*.

- ***Adriaensen Chris***
 - secretaris van het team
 - verantwoordelijk voor het op punt stellen van de *p-previous-k-nearest-neighbours* methode met het oog op uitbreidbaarheid
 - verantwoordelijk voor de uitwerking van het *walk*-testprogramma voor de plaatsbepalingsexperimenten
 - actief meegewerkt aan alle experimenten en optimalisaties voor plaatsbepaling
 - verantwoordelijk voor het verslag
- ***Balog Daniel***
 - verantwoordelijk voor de uitwerking van de client (PDA) en ServerGUI

- ***Cuypers Cedric***
 - verantwoordelijk voor de uitwerking van de game logica en implementatie ervan
 - verantwoordelijk voor de uitwerking van *ServerCore*

- ***Dellaert Phlippe***
 - coördinator van het team
 - verantwoordelijk voor de uitwerking van de database en server
 - verantwoordelijk voor de demonstratie
 - verantwoordelijk voor de presentatie

- ***Maerien Jef***
 - verantwoordelijk voor de uitwerking van de client (PDA) en ServerGUI

- ***Van Den Broeck Guy***
 - verantwoordelijk voor de uitwerking van XML-RPC
 - verantwoordelijk voor de uitwerking van *Sockets*
 - verantwoordelijk voor de uitwerking van de *particle filter*
 - verantwoordelijk voor alle experimenten en optimalisaties voor plaatsbepaling

- ***Van Ranst Wouter***
 - verantwoordelijk voor de uitwerking van XML-RPC
 - verantwoordelijk voor de uitwerking van *Sockets*
 - verantwoordelijk voor de poster

Als laatste tonen we de belasting van het team i.f.v de tijd (zie Tabel 1). Ook het totaal aantal uren belasting per teamlid is getoond.

Dag	Guy	Cedric	Daniel	Wouter	Philippe	Chris	Jef
13/02/07	5	5	5	5	5	5	5
14/02/07	4	4	2	3	5	5	2
20/02/07	5	5	5	5	5	5	5
23/02/07	2	2	0	0	0	2	0
27/02/07	5	5	5	5	5	5	5
03/03/07	6	0	0	0	0	4	2
06/03/07	5	5	5	5	5	5	5
09/03/07	3	0	2	0	0	4	2
13/03/07	5	5	5	5	5	5	5
15/03/07	2	2	0	0	0	5	2
20/03/07	5	5	5	5	5	5	5
22/03/07	4	2	0	0	0	4	2
27/03/07	5	5	5	5	5	5	5
Paasvakantie	20	4	3	2	3	5	3
17/04/07	5	5	5	5	5	5	5
19/04/07	6	3	3	0	0	0	0
24/04/07	5	5	5	5	5	5	5
26/04/07	4	2	4	2	4	3	5
01/05/07	5	5	5	5	6	5	5
03/05/07	5	2	0	0	0	0	0
04/05/07	5	2	0	0	0	0	0
08/05/07	5	5	5	5	5	5	5
11/05/07	4	3	5	5	5	4	2
13/05/07	6	3	2	4	6	5	3
15/05/07	5	5	5	5	5	5	5
16/05/07	4	3	4	3	3	8	5
Totaal	135	92	85	79	87	109	88

Tabel 1 Belasting van het team.

Appendix B

Afstandsfuncties

Men kan de *fingerprinting* methode op vele manieren implementeren aangezien men vrij is in de definitie van de dichtstbijzijnde *fingerprint*. In deze appendix bekijken we verschillende definities voor een afstandsfunctie op de *radio map*, waarmee we dan de dichtstbijzijnde *fingerprint* kunnen berekenen.

B.1. Afstand tussen fingerprint en scan voor 1 WAP

In de eerste plaats moeten we de afstand tussen een *fingerprint* en een scan definiëren voor één enkel WAP.

B.1.1. Verschil van gemiddelden

De simpelste mogelijkheid is ongetwijfeld gewoon het verschil nemen tussen de gemiddelde waarden van het ontvangen signaal van dat bepaald WAP op de *fingerprint* en op de plaats waar we nu staan. Het voordeel is dat dit een zeer simpele methode is. Nadeel is dat de variaties van de verschillende signalen in deze methode niet worden mee ingerekend. Ook wordt er hier geen rekening gehouden met het feit dat het verschil tussen de ontvangen signalen duidt op een niet-lineair verband van afstanden. Als er slecht 10 dB verschil is tussen de signalen duidt dit op een verschil van ongeveer 3.16 meter is. Een verschil van 20 dB verschil duidt op ongeveer 10 meter verschil en een verschil van 30 dB duidt op 30 meter verschil.

B.1.2. Verschil van medianen

De volgende mogelijkheid van afstand is de mediaan waarde. Een mediaan heeft dezelfde voor en nadelen als het gemiddelde, met het kleine verschil dat het minder gevoelig zal zijn aan eventuele grote afwijkingen. Dit zal geen groot verschil geven aangezien we toch maar over een relatief korte tijdsperiode meten. Enkele afwijkingen zullen waarschijnlijk toch bijna onmogelijk gedetecteerd worden aangezien de hoeveelheid meetgegevens te beperkt is.

B.1.3. Correlatie tussen de verdelingen

De correlatie stelt de 'gelijkheid' voor tussen verdelingen. De formule voor de correlatie tussen 2 verdelingen met dichtheidfunctie $f_1(x)$ resp. $f_2(x)$ is

$$corr_{12} = \int_{-\infty}^{\infty} f_1(x)f_2(x)dx$$

of

$$corr_{12} = \int_{-\infty}^{\infty} f_{corr_{12}}(x)dx$$

met

$$f_{corr_{12}}(x) = f_1(x)f_2(x)$$

Als de verdelingen gaussisch zijn verkrijgen we mits een kleine vereenvoudiging

$$f_{corr_{12}}(x) = \frac{1}{2\pi\sigma_1\sigma_2} e^{-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2} e^{-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2}$$

We nemen voor de afstand tussen de 2 sets van metingen de inverse van de correlatie van die 2 sets.

$$distance_{12} = \frac{1}{corr_{12}}$$

De integraal is niet analytisch op te lossen en moet dus numeriek opgelost worden. Een probleem dat zich dan stelt is het feit dat we niet over een oneindig interval kunnen integreren. We kunnen het interval zo inkorten dat we geen belangrijk verschil in de waarde van de integraal ondervinden. We gebruiken voor de numerieke integratie een automatisch integratie met de regel van Simpson. Dit algoritme gaat in elk deelinterval de functie benaderen door een 2de graads veelterm, hiervoor worden de uiteinden en het midden van het interval gebruikt.

We bekijken een voorbeeld met 3 sets van metingen.

$$set_1 : \mu_1 = 0 \text{ en } \sigma_1 = 1$$

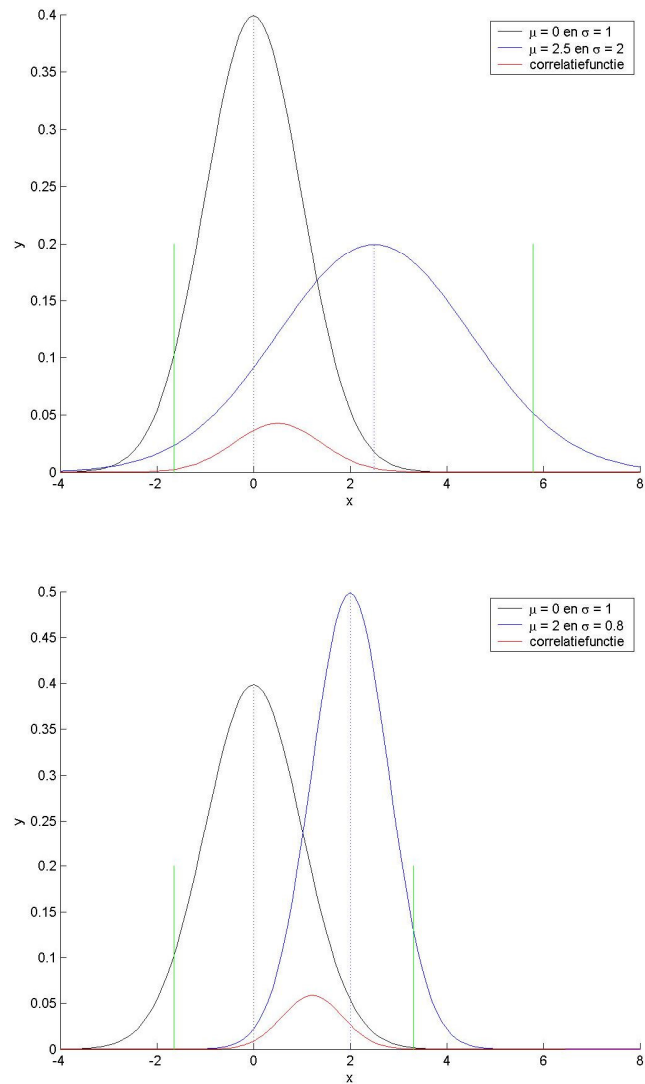
$$set_2 : \mu_2 = 2.5 \text{ en } \sigma_1 = 2$$

$$set_3 : \mu_2 = 2 \text{ en } \sigma_1 = 0.8$$

Berekenen we de correlatie tussen set_1 en set_2 en de correlatie tussen set_1 en set_3 dan zien we dat desondanks het gemiddelde van set_3 het dichtst bij het gemiddelde van set_1 ligt (zie B.1.1), met de correlatie set_2 het dichtst bij set_1 ligt (zie Figuur 20).

$$corr_{12} = 0.0473.. \text{ dus } distance_{12} = 21.11..$$

$$corr_{13} = 0.0459.. \text{ dus } distance_{13} = 21.74..$$



Figuur 20 Gaussische verdelingen van de 3 sets en de correlatiefuncties.

Voordeel van deze methode is dat het wel alle informatie gebruikt die er in de metingen ter beschikking is. Nadeel is dat het meer rekenwerk vraagt dan het simpel gemiddelde en dat het geen rekening houdt met het niet-lineaire verband tussen signaalsterkte en afstand.

B.1.4. Hypothesetesten

Twee andere methodes halen we uit de statistiek. We gaan een hypothesetest ondernemen om te bepalen wat de kans is dat een uitspraak waar is. Hoe groter de kans dat de uitspraak waar is (de gemiddeldes van de sterktes van de twee signalen zijn gelijk) hoe groter de kans dat we op de betreffende *fingerprint* zijn. Dit staat ook bekend als de *Student-t Test* en wordt vaak gebruikt om na te gaan of 2 reeksen metingen uit dezelfde populatie komen. In beide gevallen is de nulhypothese (H0) dat de gemiddeldes van de populaties gelijk zijn en de alternatieve hypothese (H1) dat de gemiddeldes niet gelijk zijn..

De eerste relatief simpelste methode is ervan uit gaan dat ook de standaard deviaties gelijk zijn.

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1}{n_1} + \frac{\sigma_2}{n_2}}}$$

Een tweede methode gaat ervan uit dat de standaarddeviaties verschillend zijn.

$$t = \frac{\mu_1 - \mu_2}{s_{\mu_1 - \mu_2}}$$

met

$$s_{\mu_1 - \mu_2} = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{(n_1 + n_2 - 2)} \left(\frac{1}{n_1} + \frac{1}{n_2} \right)}$$

Beiden zijn *Student-t* verdeeld. Omdat het aantal metingen voor de *fingerprint* zo hoog is, benadert de verdeling (met 100^{den} vrijheidsgraden) de normaalverdeling. We benaderen de bovenkwantiel-kans met een lineair stuksgewijze interpolatie. Als totaal resultaat krijgen we dus de kans in percent dat de gemiddeldes niet gelijk zijn.

Voordeel van deze methodes is dat het ook weer alle data gebruikt dat ter beschikking is. Mogelijk nadeel is dat het werkt op de ontvangen signaalsterktes in plaats van de afstand.

B.1.5. Niet-lineaire transformatie: exponentieel 1

Deze methode berekent het verschil in dB tussen de sterkte van de ontvangen signalen en de signalen van de *fingerprint*. Dit verschil gebruiken we dan om de vermoedelijke minimum afstand te berekenen tussen de gebruiker en de *fingerprint*. Eigenschappen hiervan zijn dat het wel rekening houdt met het feit dat dB niet lineair evolueert als we verder van een WAP gaan. Nadeel is dat de ruis ook exponentieel wordt getransformeerd.

B.1.6. Niet-lineaire transformatie: exponentieel 2

Een alternatief dat we ook uitproberen bestaat erin eerst de niet-lineaire transformatie naar afstand te maken, en vervolgens het verschil in afstand als afstandsmaat te gebruiken.

B.2. Afstand tussen fingerprint en scan

Na een van deze functies te hebben toegepast, krijgen we voor elke *fingerprint* een array van afstanden. We moeten nu deze afstanden voor elk WAP herleiden naar een enkele afstand. Een transformatie dus van een n-dimensionale vectorruimte naar een 1-dimensionaal geordend veld.

B.2.1. Oneindig-norm

De oneindig norm neemt het grootste verschil. Dit zegt echter heel weinig aangezien het zeer sterk beïnvloed kan worden door een enkel WAP. Ook is het zeer waarschijnlijk dat een beetje ruis zelfs onze metingen in de war kunnen sturen.

B.2.2. Gemiddelde van afwijkingen

Het gemiddelde van de afwijkingen zal minder beïnvloed worden door eventuele piekwaarden door omgevingsvariabelen, maar zal door eventuele positieve en negatieve verschillen in de signalen zelfs voor grote verschillen uitgemiddeld worden.

Dit kan opgelost worden door de absolute waarde van deze afwijkingen te nemen (wat dus eigenlijk overeen komt met de 1 norm). Nadeel is hier ook weer dat 1 of enkele relatief grote afwijkingen het volledig resultaat negatief kan beïnvloeden. Het maakt ook niet uit of deze afwijking op grote of kleine afstand gebeurt, alle afwijkingen tellen even hard door.

B.2.3. 2-norm

Als we normen nemen, denken we automatisch ook aan de euclidische norm, met deze norm zullen we de grootste afwijkingen ook weer iets meer laten doorwegen, wat zowel goed als slecht zou kunnen zijn.

B.2.4. Gewogen 2-norm

We kunnen bij de 2-norm (en ook de 1-norm) gewichten aan de waarden toekennen, maar dit zou ons in eerste instantie te ver brengen. We zullen een simpele toekenning testen. We vermenigvuldigen elk verschil van signaal tot *fingerprint* met de gemiddelde waarde van de *fingerprint* voor dat WAP, en delen dit door de som van de sterktes van alle WAP's voor die *fingerprint*.

Appendix C

Update methodes

In deze appendix bekijken we een aantal verschillende update-methodes voor een *particle filter*.

C.1. PerMeasurement

Deze update methode zoek voor elke *particle* de *fingerprint* die er het dichtste bij ligt. Deze *fingerprint* heeft voor elk WAP een gemiddelde en standaarddeviatie. Uit deze gegevens kunnen we berekenen wat de kans is dat we een bepaalde meting zouden ontvangen. We veronderstellen hier een normaalverdeling van de signaalsterkte. Dit doen we voor elk WAP en uiteindelijk vermenigvuldigen we al die kansen.

Omdat er meer *particles* dan *fingerprints* zijn is het inefficiënt om telkens opnieuw de kansberekening te doen. Daarom gebruiken we een cache die de zaken sterk versnelt.

C.2. PerMeasurementInterpolation

In plaats van de dichtstbijzijnde *fingerprint* te nemen zoals bij *PerMeasurement* nemen we nu een lineaire interpolatie van de rondomgelegen *fingerprints*. Uit de experimenten blijkt dat die veel meer rekenkracht vraagt en geen winst oplevert qua accuraatheid.

C.3. PerMeasurementAdditive

In plaats van de kansen van de verschillende WAP's te vermenigvuldigen tellen we ze nu op. Dit minimaliseert de invloed van 1 slecht signaal op het resultaat. In de experimenten is deze methode de tweede beste, na de methode beschreven hieronder.

C.4. Knearest

Deze methode lijkt op de methode die we gebruiken in het huidig ontwerp maar presteert toch opmerkelijk slechter. Deze methode vraagt niet het resultaat op van de *p-previous-k-nearest-neighbours* methode maar wel de plaatsen van de *neighbours* zelf. Door eigen gewichten toe te kennen en een soort normaalverdeling op te bouwen rond elke *neighbour* berekent de *Knearest updater* zijn kansen. De resultaten van deze methode zijn ondermaats.

Appendix D

Testresultaten

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	avg
0	386,81	360,5	352,89	349,98	348,79	348,87	366,34	361,13	359,72	358,29	372,61	367,13	366,75	376,46	375,33	382,28	364,62
1	387,76	360,38	352,99	350,41	348,48	348,53	366,23	361,25	359,19	359,19	371,77	368,24	366,56	376,88	375,25	382,78	364,74
2	370,78	350,08	342,54	340,1	338,87	338,33	355,19	350,31	349,46	348,39	359,57	356,62	354,93	363,98	361,15	368,14	353,03
3	371,58	349,98	342,4	340,02	338,42	338,27	355,35	350,27	348,4	347,61	359,76	356,68	354,79	363,52	363,54	367,36	353
4	370,92	349,53	342,9	339,34	338,18	337,79	355,03	349,42	348,13	347,08	359,36	355,48	354,19	362,15	361,06	366,34	352,31
5	370,96	349,91	343,15	339,74	338,41	337,67	354,97	349,7	347,5	347,5	359,57	354,91	354,7	362,65	361,09	366,53	352,44
6	387,5	360,77	353,12	349,78	348,99	348,73	366,93	361,57	359,16	358,08	371,77	368,54	366,05	376,95	374,17	381,63	364,61
7	376,56	353,07	345,44	343,29	342,41	341,78	359,14	353,53	351,54	350,65	363,36	358,2	358,7	366,53	365,63	371,38	356,33
8	375,93	353,45	345,6	343,16	341,66	341,4	358,74	353,15	351,01	350,35	362,89	358,84	358,25	367,53	365,05	373,35	356,27
9	375,63	352,98	345,92	343,34	342,07	341,4	358,49	353,3	351,37	350,37	362,58	359,01	358,3	366,81	365,47	371,3	356,15
10	387,51	360,76	352,73	350,28	348,15	349,19	366,79	362,25	359,27	358,9	371,5	367,86	367,23	376,4	374,45	382,05	364,71
11	378,11	354,2	347,62	345,08	343,18	342,56	360,47	355,11	353,26	352,18	364,75	360,68	359,65	369,31	367,79	373,67	357,98
12	378,75	354,64	347,12	344,98	343,34	343,01	360,53	354,76	353,38	352,24	364,89	361,12	360,8	369,45	368,16	372,69	358,12
13	387,46	360,64	352,93	349,84	348,82	348,82	366,89	361,5	358,79	357,71	372,02	368	366,91	377,07	375,69	381,43	364,66
14	384,71	358,66	351,17	348,16	347,11	346,57	364,6	358,78	357,04	355,72	369,44	364,88	364,3	373,99	372,31	378,02	362,22
15	387	360,89	352,6	350,55	349,46	348,63	366,81	361,44	359,47	357,72	371,98	367,68	366,5	376,9	374,82	381,03	364,59
avg	379,87	355,65	348,2	345,5	344,15	343,85	361,41	356,09	354,17	353,25	366,11	362,12	361,16	370,41	368,81	375	

Tabel 2 Gemiddelde fout (in cm) voor de pseudo-realistische test voor verschillende combinaties van coëfficiënten voor de *p-previous-k-nearest-neighbours* methode die de *particle filter* gebruikt. (De beste resultaten staan in het groen gemarkeerd, met de allerbeste in het donkergroen. Zie ook Tabel 3 voor de betekenis van de coëfficiënten.)

Code	$p = 0 / k = 1$	$p = 1 / k = 2$	$p = 2 / k = 3$	$p = 3 / k = 4$	$p = 4 / k = 5$	$p = 5 / k = 6$
0	1.00					
1	0.50	0.50				
2	0.50	0.25	0.25			
3	0.50	0.25	0.125	0.125		
4	0.50	0.25	0.125	0.0625	0.0625	
5	0.50	0.25	0.125	0.0625	0.03125	0.03125
6	0.60	0.40				
7	0.60	0.24	0.16			
8	0.60	0.24	0.096	0.064		
9	0.60	0.24	0.096	0.0384	0.0256	
10	0.70	0.30				
11	0.70	0.18	0.12			
12	0.70	0.18	0.084	0.036		
13	0.80	0.20				
14	0.80	0.16	0.04			
15	0.90	0.10				

Tabel 3 Bijhorende coëfficiënten voor de *p*-previous-k-nearest-neighbours methode voor Tabel 2.
(De betekenis verschilt afh. het voor p of k wordt gebruikt, zie hiervoor 4.2.1)