

# Software Engineering en Gedistribueerde Applicaties

## Eindverslag

Team F.A.C.H.T.

Chris Bovenschen, Harm Dermois,  
Alexandra Moraga Pizarro, Tamara Ockhuijsen en Fredo Tan

6104096, 0527963, 6129544, 6060374, 6132421

Universiteit van Amsterdam

29 juni 2011

# Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>3</b>
<b>2</b>	<b>Implementatie</b>	<b>4</b>
2.1	Low-level . . . . .	4
2.1.1	Listener . . . . .	4
2.2	Sensormodules . . . . .	4
2.2.1	Odometry . . . . .	4
2.2.2	Range scanner . . . . .	4
2.2.3	Sonar . . . . .	5
2.3	Mid-level . . . . .	5
2.3.1	Collision avoider . . . . .	5
2.3.2	Wall search . . . . .	5
2.3.3	Wall follow . . . . .	5
2.4	High-level . . . . .	6
2.4.1	Map maker . . . . .	6
2.4.2	Path finding . . . . .	6
2.5	Libraries . . . . .	6
2.5.1	Communicator . . . . .	6
2.5.2	Movements . . . . .	7
<b>3</b>	<b>Tests</b>	<b>7</b>
3.1	Listener . . . . .	7
3.2	Odometry . . . . .	7
3.3	Range scanner . . . . .	7
3.4	Sonar . . . . .	7
3.5	Collision avoider . . . . .	7
3.6	Wall search . . . . .	7
3.7	Wall follow . . . . .	8
3.8	Wall combo . . . . .	8
3.9	Map maker . . . . .	8
3.10	Path finding . . . . .	8
<b>4</b>	<b>Experiment</b>	<b>8</b>
<b>5</b>	<b>Resultaten</b>	<b>8</b>
<b>6</b>	<b>Conclusie</b>	<b>8</b>
<b>7</b>	<b>Discussie</b>	<b>8</b>
<b>8</b>	<b>Bronvermelding</b>	<b>8</b>
<b>9</b>	<b>Appendix</b>	<b>9</b>

# 1 Inleiding

Dit verslag beschrijft het ontwerpen, implementeren en testen van een gedistribueerde robotapplicatie. De robot die hiervoor wordt gebruikt, is de gesimuleerde P2DX. Deze heeft twee aangedreven wielen en een zwenkwiel. Het platform voor de robotsimulatie is USARSim. Het doel is om met behulp van de sensoren op de robot in een vrij volle ruimte de volgende acties uit te kunnen voeren:

- een pad langs een wand te rijden
- botsingen te vermijden
- zonodig een wand op te zoeken
- uit doodlopende stukken te ontsnappen
- bijhouden waar hij is en welk pad hij heeft gevolgd
- de omgeving in kaart te brengen

Het project is op te delen in verschillende fases. Aan het begin is er nagedacht over een ontwerp met een model, een planning en een specificatie van het eindproduct. Hieruit is een voorlopig werkplan ontstaan. In het definitieve werkplan is een uitgebreidere beschrijving van de realisatie van verschillende componenten met bijbehorende tests te vinden samen met de eerste inleverbare versie van de code. Het grootste deel van de tijd heeft gezeten in de implementatie en tests. Als laatst zijn de componenten geïntegreerd tot een geheel, getest en geoptimaliseerd.

Dit verslag behandelt eerst de implementatie van de componenten met de bijbehorende tests en de opzet van het experiment. Daarna worden de resultaten besproken en hieruit volgt een conclusie.

## 2 Implementatie

De modules zijn verdeeld in low-level, sensor, mid-level en high-level componenten. De listener is een low-level component, de sonar, laser en odometry zijn sensormodules, de wall search, wall follow en collision avoider zijn mid-level componenten en de map maker en path finding zijn high-level componenten. De movements en communicator zijn libraries die door andere modules kunnen worden gebruikt.

De listener is verbonden met de server via een TCP-verbinding en stuurt de data direct door naar de sensormodules. De sensormodules halen de juiste waarden uit de data die voor hun bestemd zijn en sturen deze door naar de mid-level modules en op aanvraag naar de map maker en path finding. Berekenende kaarten van de map maker kunnen worden gebruikt voor de path finding. Path finding gebruikt de movements library om het pad te kunnen volgen. Als de robot aan het rijden is, staat de collision avoider te alle tijden aan.

### 2.1 Low-level

De low-level component verkrijgt rechtstreeks data van de robot en stuurt de ontvangen data van de robot direct door naar de sensormodules.

#### 2.1.1 Listener

De listener zit direct aan de server vast. Hij haalt informatie binnen die de robot verstuurt en stuurt ook commando's naar de robot toe. De binnenkomende data worden meteen doorgestuurd naar de sensormodules. Eerst controleerde de listener de data ook op geldigheid en stuurde deze in het juiste formaat door naar de bijbehorende sensormodules. Echter werd de listener overwelmd door de data, waardoor deze nu slechts een doorgever is geworden.

Zie appendix A

### 2.2 Sensormodules

De sensormodules ontvangen data van de listener. Ze kijken of de data voor hun bestemd zijn en anders negeren ze deze. Ze verkrijgen de sensorwaarden uit de data en sturen ze door naar de mid-level modules. De sensormodules kunnen door de high-level modules worden aangeroepen voor de meest recente data. Zodra er ongeldige data binnenkomen, wordt de robot gestopt.

#### 2.2.1 Odometry

Odometry haalt de drie waarden op die nodig zijn voor de bepaling van de positie van de robot. Dit zijn de x, y positie en de theta hoek in verhouding tot de startpositie en de richting van de robot. Deze waarden worden doorgestuurd naar de mid-level modules.

Zie appendix B

#### 2.2.2 Range scanner

De range scanner bevindt zich op een bepaalde hoogte boven de robot. Hij scant de omgeving horizontaal door middel van het roteren van de laser in een

bepaald interval. De range scanner leest alle 181 laserwaarden uit en stuurt ze door naar de mid-level modules.

Zie appendix C

### **2.2.3 Sonar**

De sonar leest alle 8 sonarwaarden uit en stuurt ze door naar de mid-level modules. De sonar stuurt een geluidsgolf, waardoor obstakels die op de grond liggen ook te detecteren zijn.

Zie appendix D

## **2.3 Mid-level**

De mid-level componenten verzorgen het vermijden van botsingen en het zoeken en volgen van een muur.

### **2.3.1 Collision avoider**

De collision avoider kijkt alleen naar de waarden van de sonar, omdat die in tegenstelling tot de laser wel objecten op de grond kan detecteren. Als de kleinste waarde van de sonar kleiner is dan 0,315 stopt de robot met rijden. Bij deze waarde zijn objecten op de grond nog zichtbaar. Zodra deze waarde te klein wordt, kan de sonar het object niet meer zien. Tevens wordt er met behulp van de snelheid uitgerekend na hoeveel seconden de botsing plaats zal vinden als de robot door blijft rijden. Deze berekening wordt gedaan op basis van de snelheid van de wielen, en kan dus in sommige gevallen afwijken.

Zie appendix E

### **2.3.2 Wall search**

Bij het zoeken naar een muur draait de robot eerst 360 graden om te kijken waar de dichtstbijzijnde muur is. Zodra de kleinste sensorwaarde is gevonden, roteert de robot nog een keer totdat hij de juiste positie heeft gevonden. Hij rijdt recht op de muur af totdat hij op een bepaalde afstand van de muur af staat en dan gaat hij over op wall follow.

Zie appendix F

### **2.3.3 Wall follow**

De muur wordt gevolgd die door wall search is gevonden. Als de muur zich meer aan de linkerkant van de robot bevindt, draait hij naar rechts en als de muur zich meer aan de rechterkant van de robot bevindt, draait hij naar links. De kleinste laserwaarde bevindt zich nu aan de rechter- of linkerkant. Hij blijft rechtdoor rijden als de kleinste laserwaarde niet kleiner en groter wordt dan een ingestelde waarde. Als hij te ver van de muur afwijkt, stuurt hij bij richting de muur. Als hij te dicht bij de muur komt, stuurt hij bij van de muur af. Als er geen muur meer wordt waargenomen, wordt de wall.continued functie binnen de wall search aangeroepen. Deze functie kijkt of er in de buurt nog een muur is te vinden. Als dit zo is, gaat hij daarheen. Als er zich in de buurt nog een andere muur bevindt, zoekt wall search naar de nieuwe dichtstbijzijnde muur.

Elke keer wanneer de logica klaar is, wordt er nieuwe informatie opgevraagd van de sensormodules.

Zie appendix G

## 2.4 High-level

De high-level componenten houden bij waar de robot is geweest en kunnen hem naar een ander punt laten rijden. Dit zijn de modules die meer tijd nodig hebben dan de mid-level modules.

### 2.4.1 Map maker

De map maker maakt een kaart van de constante stroom van laser en odometry waarden die hij binnenkrijgt. Voor het maken van een kaart wordt de Simultaneous Localization And Mapping (SLAM) techniek gebruikt. De kaart heeft de vorm van een matrix. De bibliotheek die wordt gebruikt om deze map te maken is tinySLAM. Dit is een zeer simpel algoritme dat niet meer dan 200 regels code bevat. Het enige wat het elke keer doet, is de gevonden lijnen tekenen. Dit algoritme heeft nog heel wat uitbreidingen, maar alleen de basis van dit algoritme wordt gebruikt. De originele code is geschreven in C, maar met behulp van een gevonden versie in Python en een paar aanpassingen is deze bruikbaar geworden.

Zie appendix H

### 2.4.2 Path finding

Path finding kan alleen gebruikt worden binnen de al ontdekte gebieden en kan dus alleen een route plannen binnen een bekend domein. Hierin probeert hij een zo kort mogelijk pad te vinden naar de bestemming. Er wordt gebruik gemaakt van het A ster zoekalgoritme om het efficiëntste pad te vinden tussen twee punten.

Zie appendix I

## 2.5 Libraries

Alle modules kunnen gebruik maken van de beschikbare libraries. Ze kunnen deze libraries importeren en hun functies gebruiken.

### 2.5.1 Communicator

De communicator library leest een bestand in dat de lokaties van de modules bevat. Via deze data maakt hij verbindingen aan met andere modules. De communicator importeert twee thread klassen en een derde klasse. De ene thread luistert naar de gemaakte verbindingen en de andere luistert naar een poort om te bepalen of er nieuwe verbindingen gemaakt moeten worden. De thread die luistert, handelt de berichten af volgens het communicatieprotocol. Hierin staan alle mogelijke berichten en tags die gebruikt worden bij de communicatie en er staat ook in wat hun doel is. Hierna handelt elke module zijn ontvangen data anders af.

Zie appendix K

### **2.5.2 Movements**

Dit is een library waarin alle bewegingen zijn gedefinieerd. Deze library wordt gebruikt voor de modules die de robot willen besturen. In de movements staan functies gedefinieerd voor het vooruit, naar links, naar rechts en achteruit rijden. Tevens is het mogelijk om naar links en rechts te roteren, een spoor achter te laten, de camera te zien vanuit de robot en de robot te laten stoppen met rijden.

Zie appendix L

## **3 Tests**

Voor elke module is er een aparte test geschreven om er zeker van te zijn dat elke module op zichzelf goed werkt. Elke module wordt direct aan de simulator vastgezet en getest met de directe data die binnenkomen.

### **3.1 Listener**

De listener wordt getest door data te verkrijgen en deze uit te printen.

### **3.2 Odometry**

De odometry wordt getest door de robot te laten rijden, 100 regels data te verkrijgen van de simulator en hiervan de odometriewaarden te printen. De odometriewaarden worden getest op geldigheid.

### **3.3 Range scanner**

De range scanner wordt getest door de robot te laten rijden, 100 regels data te verkrijgen van de simulator en hiervan de 181 laserwaarden te printen. De laserwaarden worden getest op geldigheid.

### **3.4 Sonar**

De sonar wordt getest door de robot te laten rijden, 100 regels data te verkrijgen en hiervan de 8 sonarwaarden te printen. De sonarwaarden worden getest op geldigheid.

### **3.5 Collision avoider**

De collision avoider wordt getest door de robot vanuit verschillende startposities rond te laten rijden zonder dat er botsingen ontstaan.

### **3.6 Wall search**

De wall search wordt getest door de robot vanuit verschillende startposities de dichtstbijzijnde muur te laten zoeken.

### **3.7 Wall follow**

De wall follow wordt getest door de robot vanuit verschillende startposities een muur te laten volgen. Het is de bedoeling dat deze module wordt aangeroepen wanneer er al een muur is gevonden. Dan kan de robot de muur naar links en naar rechts toe volgen. Binnen deze gebruikte methode zijn stompe bochten ook mogelijk.

### **3.8 Wall combo**

De wall combo is geschreven om te testen of de wall search goed werkt in samenwerking met de wall follow. Deze test simuleert de werking in een gedistribueerd systeem.

### **3.9 Map maker**

De map maker wordt getest door hem een output bestand met laser- en odometriewaarden van de wall combo te laten lezen en de kaart ervan te tekenen.

### **3.10 Path finding**

De path finding wordt getest door een punt op te geven waar de robot naartoe moet rijden. Hij moet hierbij botsingen ontwijken en naar het punt toe rijden op een zo efficiënt mogelijke manier.

## **4 Experiment**

De robot wordt geïnitieerd op een bepaalde plaats. Hij zoekt een muur en blijft deze muur volgen. Zodra hij de opdracht krijgt om naar een punt toe te rijden, wordt path finding ingeschakeld. De map maker werkt continu de kaart bij en de collision avoider zorgt ervoor dat er geen botsingen plaatsvinden. Als hij geen nieuwe opdracht krijgt om naar een ander punt toe te rijden, gaat hij weer een muur zoeken en deze volgen.

## **5 Resultaten**

## **6 Conclusie**

## **7 Discussie**

## **8 Bronvermelding**



## 9 Appendix

### A. Listener

De listener stuurt berichten van de robot door naar de modules odometry (ODO), range scanner (RSC) en sonar (SNR). Als een datapakket incompleet is, dat wil zeggen dat er geen `\n` aan het einde is, wordt er een nieuw datapakket achter geplakt.

### B. Odometry

De odometry module krijgt data binnen van de listener. De data worden gesplitst in arrays op elke `\r\n` die er in voorkomt. Vervolgens wordt de functie `re.findall('[^\\]*\\|S+', string[i])` gebruikt om elke array te splitsen in elementen die beginnen en eindigen met een accolade en overige elementen. Elke array bevat sensorinformatie als het eerste element van de array gelijk is aan `SEN`. Bij berichten van het type odometry staat er op de tweede plaats in de array het element `{Type Odometry}`. Er is echter tussen het eerste en tweede element het element tijd toegevoegd om te kijken of de odometriewaarden nieuw zijn of niet, waardoor het `{Type Odometry}` element nu op de derde plaats in de array staat. De odometry module gebruikt de array als dit laatste element op die plaats is gevonden. De tijd van het bericht wordt samen met de odometriewaarden op de vijfde plaats in de array als een string meegegeven aan de functie `odometry_module()`. De odometriewaarden komen als volgt binnen: `{Pose 0.0000,0.0000,0.0000}`, oftewel `x,y,theta`. De `odometry_module()` functie verkrijgt de drie getallen uit deze string, controleert of deze waarden geldig zijn door ze om te zetten in floats en test of de theta tussen -3.15 en 3.15 ligt. Alleen als de waarden geldig en nieuw zijn, worden ze in een string van het formaat `tijd+x,y,theta` doorgestuurd naar de mid-level modules.

### C. Range scanner

De range scanner module krijgt data binnen van de listener. De data worden gesplitst in arrays op elke `\r\n` die er in voorkomt. Vervolgens wordt de functie `re.findall('[^\\]*\\|S+', string[i])` gebruikt om elke array te splitsen in elementen die beginnen en eindigen met een accolade en overige elementen. Elke array bevat sensorinformatie als het eerste element van de array gelijk is aan `SEN`. Bij berichten van het type range scanner staat er op de tweede plaats in de array het element tijd en op de derde plaats in de array het element `{Type RangeScanner}`. De range scanner module gebruikt de array als dit laatste element op die plaats is gevonden. De tijd van het bericht wordt samen met de laserwaarden op de zevende plaats in de array als een string meegegeven aan de functie `range_module()`. De tijd en de laserwaarden worden gesplitst en gecontroleerd op geldigheid door ze om te zetten in floats. Alleen als de waarden geldig en nieuw zijn, worden ze in een string van het formaat `tijd+x,x,...,x` doorgestuurd naar de mid-level modules.

## D. Sonar

De sonar module krijgt data binnen van de listener. De data worden gesplitst in arrays op elke `\r\n` die er in voorkomt. Vervolgens wordt de functie `re.findall('[^\\]*\\S+', string[i])` gebruikt om elke array te splitsen in elementen die beginnen en eindigen met een accolade en overige elementen. Elke array bevat sensorinformatie als het eerste element van de array gelijk is aan `SEN`. Bij berichten van het type sonar staat er op de tweede plaats in de array het element tijd en op derde plaats in de array het element `{Type Sonar}`. De tijd van het bericht wordt samen met de sonarwaarden als een string meegegeven aan de functie `sonar_module()`. Deze functie splitst de tijd in de sonarwaarden en controleert ze op geldigheid door ze om te zetten in floats. Alleen als de waarden geldig en nieuw zijn, worden ze in een string van het formaat `tijd+1,2,3,4,5,6,7,8` doorgestuurd naar de mid-level modules.

## E. Collision avoider

De collision avoider module maakt gebruik van een semafoor 'flag'. Zodra de flag gelijk is aan nul, zal de robot vooruit rijden en wordt `getdata()` aangeroepen. BLABLABLA. Zodra het type sensor gelijk is aan sonar wordt er gekeken of de lengte van de data groter is dan 9. Hierin wordt de `min_sonar_val()` functie aangeroepen met als parameter de meegegeven sonarwaarden. Hierbij wordt de `string_to_float()` functie gebruikt. Elke sonarwaarde wordt omgezet in een float en deze floats worden vervolgens in een array gestopt en geretourneerd. Deze worden gesorteerd en er wordt 1 bij de eerste indexwaarde opgeteld, zodat er van 1 tot 8 wordt geteld in plaats van 0 tot 7. De `min_sonar_val()` retourneert twee waarden, waarvan de eerste de minimale waarde is en de tweede de indexwaarde waar zich de minimale waarde bevindt. Voor elke sonarwaarde wordt er gekeken of de float van elke waarde kleiner is dan de zogenaamde level, die als .25 is gegeven. Dit zal de robot genoeg ruimte geven om te kunnen roteren zonder dat deze in aanraking komt met een object. Als één van de twee meest rechter sonarwaarden kleiner zijn dan de meest linker sonarwaarde, wordt de robot naar links gedraaid. Zijn de twee meest linker sonarwaarden kleiner dan de meest rechter sonarwaarde, dan wordt de robot naar rechts gedraaid. Zodra de minimale waarde bij de voorste twee indexwaarden ligt en de flag gelijk is aan nul, wordt de `calc_collision()` functie aangeroepen voor een mogelijke botsing. Door middel van de `calc_collision()` en de `calc_speed()` kan de tijd in secondes worden berekend voordat de robot in aanraking komt met een object. Als de meest rechter sonarwaarde eveneens kleiner is dan de meest linkerwaarde zal de robot naar rechts draaien, en andersom. Tenslotte wordt er een 1 geretourneerd.

- F. Wall search
- G. Wall follow
- H. Map maker
- I. Path finding
- J. Communicatieprotocol

```

TAG:
REQ: data sturen
RCV: data ontvangen
CMD: movement commando's
NEX: go to the next module

```

```

data TAG:
SNR: sonar
ODO: odometry
RSC: rangescanner
MAP: map

```

```

module TAG:
ODO: Odometry
RSC: Rangescanner
SNR: Sonar
WFW: Wall Follow
CAV: Collision Avoider
WSC: Wall Search
MAP: Map Maker
PLA: Route Planner
LIS: Listener

```

```

dividers:
#: eindetransmissie
!: transmissie divider

```

```

Vorm:
REQ:
REQ!"own module TAG"#
example: REQ!WFW# this send to a sensor module will return the data of that module.
RCV!"data TAG!"data"#
example: RCV!SNR!x,x,x,x,x,x,x# this will send the data to a module.
CMD!"commando"# CMD!drive# will send a drive command.
NEX!"side"# example: NEX!1# 1 if wall is on the right side.

```

```

data:

```

```

SNR:
naar sensor module:
{Name F1 Range "x"} {Name F2 Range "x"} {Name F3 Range "x"} {Name F4 Range "x"}
{Name F5 Range "x"} {Name F6 Range "x"} {Name F7 Range "x"} {Name F8 Range "x"}
uit sensormodule:
{Time}+x,x,x,x,x,x,x,x

RSC:
naar sensor module:
{Range "x","x",...,"x"}
uit sensormodule:
{Time}+x,x,x,x,x,x,x,x,x,x,...,x,x,x

ODO:
naar sensor module:
{Pose x,y,theta}
uit sensormodule:
{Time}+x,y,theta

MAP:
"# of characters" "type" "# of characters" "type" _..._ "# of characters" "type"

```

## K. Communicator

## L. Movements

Elke module kan een commando type aanroepen in de movements library via de `handle_movements()` functie. Hierin staan alle mogelijke bewegingen die behandeld kunnen worden. `go_drive()` zorgt ervoor dat de robot vooruit en achteruit kan rijden, bochten en rotaties kan maken. Hierbij wordt voor zowel het linker- als het rechterwiel dezelfde parameter meegegeven. Bij `go_rotate_right()` en `go_rotate_left()` wordt een rotatie uitgevoerd om de as van de robot. De meegegeven parameter bepaalt de snelheid van de rotatie. Om naar rechts te draaien moet de parameter van het rechterwiel groter zijn dan die van het linkerwiel en omgekeerd. De functies `go_right()` en `go_left()` worden hierbij gebruikt. De robot remt bij het gebruik van `go_brake()`, hierbij staan beide wielen op een snelheid van 0. Door de wielen een gelijke negatieve waarde mee te geven in de functie `go_reverse()`, zal de robot achteruit rijden. De functie `go_camera()` zorgt ervoor dat door indrukken van de linkermuisknop het beeld zichtbaar is vanuit de camera van de robot. Om hier vervolgens uit te komen, dient er op de rechtermuisknop gedrukt te worden.