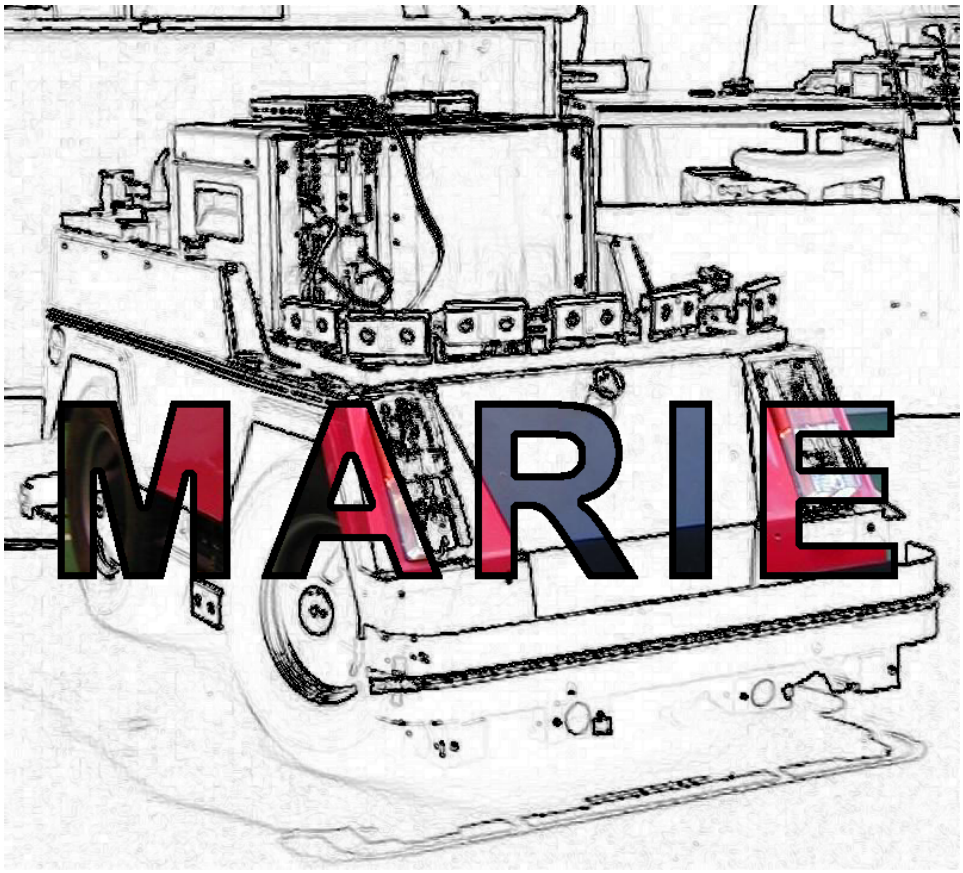
Een on-line planner voor



studie: kunstmatige intelligentie
specialisatie: Autonome systemen

Frank Terpstra
begeleider: Arnoud Visser

1-3-2001
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam

Samenvatting

In deze scriptie wordt de ontwikkeling beschreven van een on-line planner voor de MARIE robot. Een on-line planner is een programma dat tijdens de uitvoering van een plan dit plan kan aanpassen. Hiervoor is gekeken naar verschillende planners uit het verleden, zowel theoretische planners voor het STRIPS domein als planners op echte robots. Als doel is gesteld een beter betrouwbaarheid en robuustheid van MARIE. Hiervoor is een on-line planner geïmplementeerd die als basis heuristiek random kiezen heeft, met daar aan toegevoegd enkele elementen van planners uit het verleden. De on-line planner beschikt over een aantal "macro's", stukjes plan informatie. Wanneer het nodig is kiest de on-line planner één van de op dat moment (volgens de huidige status) toegestane "macro's" uit. Dit is getest door MARIE een opdracht te geven waarvoor de on-line planner een redelijk complex plan moet genereren. Uit deze test is gebleken dat de on-line planner inderdaad werkt en zorgt voor een grotere betrouwbaarheid en robuustheid van MARIE.

Voorwoord

Het heeft wat langer geduurt dan ik oorspronkelijk gehoopt had, maar nu is mijn scriptie dan toch klaar. Ik ben blij dat er uiteindelijk een echt werkend programma uit is gekomen. Dit heb ik natuurlijk niet alleen voor elkaar kunnen krijgen. Als eerste wil ik Arnoud bedanken voor alle hulp die hij gegeven heeft. Ik heb het hierbij niet alleen over de adviesen en tips bij het programmeren en schrijven, maar ook over de vele uren die hij besteed heeft aan het werkend krijgen van de wall-sensor. Hakan wil ik bedanken voor zijn hulp bij allerlei kleine en grote programmeer en systeem problemen. MARIE is een project waar al sinds 1989 aan gewerkt wordt. Ik waardeer dus ook zeer de informatie en tips die ik van Dick van Albada heb gekregen, aangezien hij jarenlang bij het MARIE project betrokken is geweest. Dankzij Nikkos Massios heb ik een bouwtekening van kruislaan 403 kunnen gebruiken voor het maken van het wereld model van MARIE, hetgeen mij een hoop kruipen met centimeters over gangen heeft bespaard. Verder wil ik Roel, Martijn, Tommy, Maurits en Joost bedanken voor het lezen en corrigeren van voorlopige versies van mijn scriptie. Tommy wil ik in het bijzonder bedanken voor het lenen van zijn digitale video camera. Ook Giel verdient een vermelding voor het lenen van een digitale (in dit geval foto) camera, verder wil ik hem ook bedanken voor het photoshoppen van de ingescande plaatjes (Frank & Meie ook bedankt voor het lenen van de scanner). Tenslotte wil ik mijn ouders bedanken voor alle steun en goede adviezen.

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 De Opdracht

Deze scriptie zal gaan over het ontwikkelen van een on-line planner voor MARIE. In dit inleidende gedeelte zal een uitleg van een aantal begrippen en concepten volgen die duidelijker moeten maken wat dit (on-line planner voor MARIE) precies betekent. Om te beginnen de opdracht die de aanleiding was voor deze scriptie.

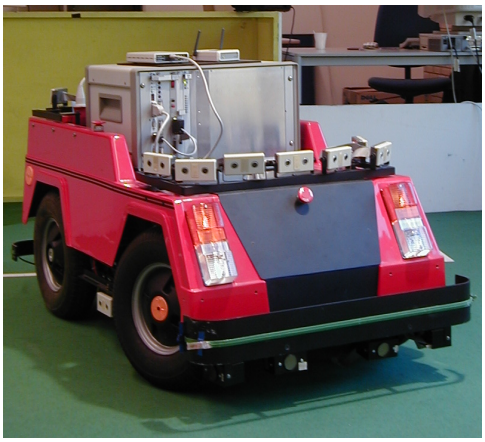
titel	On-line plan-generation for the MARIE-robot
omschrijving	In the MARIE-architecture a task is specified as an AND/OR graph. The AND-nodes are used to specify sequences of required activities, OR-nodes are used to specify alternatives. Alternative branches in a plan are a way to make the execution of a task more robust: depending on the conditions different activities are performed. Yet, fully specified alternatives make the graph quite large. This is not only a disadvantage concerning processing-time and memory usage, but also makes it difficult to verify a full graph. The task for the student will be to replace the fully specified alternatives by commands for an on-line plan-generator. The candidate has to come with a design for such an on-line plan-generator, and has to show which efficiency is gained by this approach, and how many flexibility is lost. The task can be decomposed into the following steps: 1. Acquire a good overview of the state of the art in on-line plan-generation in autonomous systems. 2. Review the current alternative branches in the MARIE-plans. 3. Design an on-line plan-generator, including its command-interface. 4. Implement and test this plan-generator. 5. Evaluate the approach, compared with the old implementation and the state of the art. It is strongly recommended that the candidate has finished the course OOAS ('Ontwikkeling en Ontwerp van Autonome Systemen') succesfully.

Om deze opdracht beter te begrijpen moet men eerst weten wie of wat de MARIE robot is, verder volgt een uitleg van de begrippen robot, autonoom en on-line plannen. Daarna wordt uitgelegd wat Kunstmatige Intelligentie hiermee te maken heeft. Als laatste komen de verwachtingen voor het uiteindelijke resultaat aan bod.

1.2 Wie of wat is MARIE

MARIE staat voor "Mobile Autonomous Robot in an Industrial Environment". Ze komt voort uit het Esprit-MARIE project dat in 1989 begon en

onderzoek deed naar de combinatie van lage niveau aansturing en Kunstmatige Intelligentie in autonome robots. Het doel van dit onderzoek was om een robot te maken die in echte complexe en veranderende omgevingen zijn taak kon uitvoeren. Eén van de test robots voor dit project was een mobiele robot gebaseerd op een oranje invalide karretje. Dit is de robot die nu bekend is onder de naam MARIE en ook de robot die gebruikt is voor de experimenten die in deze scriptie beschreven worden.



Figuur 1.1: MARIE

Voor MARIE was een uitgebreid systeem ontwikkeld dat alle mogelijke "exceptions" ofwel onvoorziene situaties probeerde op te vangen. Dit gebeurde op meerdere niveaus, de opdracht van deze scriptie gaat over het hoogste niveau van "exception handling".

1.3 Wat is een robot

Volgens van Dale is een robot een mechanisme dat al of niet de gedaante van een mens heeft en verrichtingen of arbeid uit kan voeren.

Deze definitie geeft al aan dat veel mensen een robot zien als iets dat op een mens lijkt. Nu lijkt het dus op het eerste gezicht logisch dat de meeste robots op mensen lijken omdat we ze werk willen laten verrichten dat oorspronkelijk door mensen gedaan werd. Dit is echter niet zo vanzelfsprekend aangezien het meeste werk dat we robots willen laten doen werk is waar de mens niet erg goed in is of werk is waar de mens niet zoveel zin in heeft. Feitelijk is de menselijke vorm voor deze taken dus bijzonder ongeschikt. Een robot moet dus ontworpen zijn om zijn taak zo goed mogelijk uit te kunnen voeren en hiervoor zijn vaak de meeste menselijke eigenschappen overbodig. Erger nog de meeste menselijke eigenschappen zitten een ook maar enigszins gespecialiseerde robot behoorlijk in de weg.

Citaat uit the hitchhikers guide to the galaxy:

> Modern elevators are strange and complex entities. The ancient electric winch and "maximum-capacity-eight-persons" jobs bear as much relation to a Sirius Cybernetics Corporation Happy Vertical People Transporter as a packet of mixed nuts does to the entire west wing of the Sirian State Mental Hospital.
>
> This is because they operate on the curious principle of "defocused temporal perception". In other words they have the capacity to see dimly into the immediate future, which enables the elevator to be on the right floor to pick you up even before you knew you wanted it, thus eliminating all the tedious chatting, relaxing, and making friends that people were previously forced to do whist waiting for elevators.
>
> Not unnaturally, many elevators imbued with intelligence and precognition became terribly frustrated with the mindless business of going up and down, up and down, experimented briefly with the notion of going sideways, as a sort of existential protest, demanded participation in the decision-making process and finally took to squatting in basements sulking.
>
> An impoverished hitch-hiker visiting any planets in the Sirius star system these days can pick up easy money working as a counsellor for neurotic elevators.

Voor de meeste robots zijn de overeenkomsten met de mens bijna afwezig. Ze beperken zich tot de een gering aantal functies die nodig zijn voor het uitvoeren van het specialisme van de robot. En zelfs die functies zijn meestal niet alleen specifiek voor de mens maar ook voor vele andere organismen en mechanismen. Hierdoor kan je gerust stellen dat de meeste robots in de verste verte niet op mensen lijken.

Dit is ook het geval bij MARIE alhoewel ze een menselijke naam heeft en daardoor vaak menselijke eigenschappen wordt toegedicht lijkt ze in de verste verte niet op een mens.

1.4 Wat is een autonoom systeem

Een Autonoom systeem is een systeem dat zelfstandig beslissingen kan nemen met betrekking tot de acties die het uit moet voeren. Dit betekent dus dat het een zekere mate van zelfbeschikking heeft. Hetgeen zeer nuttig is in situaties waarin de mens een systeem niet constant in de gaten kan of wil houden.

MARIE is een voorbeeld van een autonoom systeem. Ze kan bijvoorbeeld zelfstandig obstakels ontwijken, muren volgen, deze kwijtraken en weer terug vinden. Het uitbreiden van MARIE met een on-line planner moet deze

autonomie verder vergroten. Een on-line planner geeft MARIE de mogelijkheid haar plan zelfstandig te veranderen, dit geeft haar een grotere keuze vrijheid aan mogelijke acties en daarmee meer autonomie.

1.5 Welke rol speelt kunstmatige intelligentie

Het on-line plannen gaat gebeuren met één of meerdere technieken uit de kunstmatige intelligentie. Wat hopelijk gebeurt is dat MARIE meer intelligent gedrag laat zien dan tot nu toe het geval was. In het hoofdstuk planners dat volgt op deze inleiding worden meerdere van de technieken uit de kunstmatige intelligentie besproken en wordt bekeken hoe toepasbaar ze zijn op MARIE.

1.5.1 Wat doet een planner

Een planner is een programma dat taken voor robots plant. Voor een planner is het volgende bekend: een (gedeeltelijke) beschrijving van de wereld, alle mogelijke acties van de robot, de begin toestand van de robot en de doel toestand van de robot. Met behulp van deze informatie moet de planner een serie acties kiezen die de robot succesvol van begin tot doel-toestand zal brengen. Een planner kan uitgevoerd worden door een computer programma maar natuurlijk net zo goed door een mens of een combinatie van beide.

1.5.2 Verschil tussen on en off-line planners

Een off-line planner werkt voordat de robot gestart is met acties uitvoeren oftewel wanneer deze off-line is. Van tevoren wordt er een compleet plan berekend dat vervolgens wordt uitgevoerd. Tijdens de uitvoering komt een off-line planner dus niet meer in actie. Een on-line planner daarentegen kan van tevoren wel een (gedeeltelijk) plan uitwerken maar kan dit plan tijdens de uitvoering aanpassen.

1.5.3 Voordelen van on-line plannen

On-line plannen heeft grote voordelen voor met name autonome robots. Autonome robots bewegen zich in (over het algemeen) dynamische omgevingen waarvan de robot geen volledige kennis heeft. Dit kan allerlei onverwachte situaties opleveren, zoals het tegenkomen van onbekende objecten of bestaande objecten die van plaats veranderd zijn. Daarnaast heeft een robot last van onnauwkeurigheid in zijn sensoren. Dit kan leiden tot situaties waarin uitvoering van het van tevoren bedachte plan onmogelijk wordt. Bijvoorbeeld, door een afwijking in de sensoren krijgt een module die een essentieel onderdeel is van het van tevoren bedachte plan zo'n rare waarde binnen dat hij zijn taak niet uitvoert. Dientengevolge faalt het gehele van

tevoren bedachte plan aangezien er geen rekening werd gehouden met het falen van dit onderdeel. Een on-line planner kan in deze gevallen zijn plan aanpassen om hetgeen dat uitvoering van zijn plan onmogelijk maakte te omzeilen. Bijvoorbeeld in het bovengenoemde geval kan een on-line planner dezelfde module nog een keer aanroepen na een bepaalde afstand verder te zijn gereden zodat de oorzaak van de vreemde sensor waarde er niet meer is.

1.6 Verwachte resultaat

Als je kijkt naar wat een on-line planner in theorie doet is er een aantal dingen dat je logischer wijs in de praktijk zou kunnen verwachten.

- Grotere betrouwbaarheid
- Grotere robuustheid

Met betrouwbaarheid wordt bedoelt dat je ervan op aan kan dat de robot consequent zijn doel zal bereiken. MARIE zou betrouwbaarder moeten worden doordat er meer mogelijkheden zijn om een plan tot een goed einde te brengen met een on-line planner dan zonder de on-line planner. Alhoewel de on-line planner dus minder betrouwbaar is wat betreft de reproduceerbaarheid van hetzelfde gedrag. Dit komt door de grotere diversiteit aan mogelijke oplossingen om het doel te bereiken die niet onder directe controle van de opdracht gever of operator liggen.

Grotere robuustheid betekent dat zelfs als een software of hardware onderdeel zou uitvallen de kans groter is dat MARIE een opdracht voltooid. Ook dit wordt gerealiseerd door de grotere mogelijke verscheidenheid aan manieren om een doel succesvol te bereiken.

Hoofdstuk 2

Planners

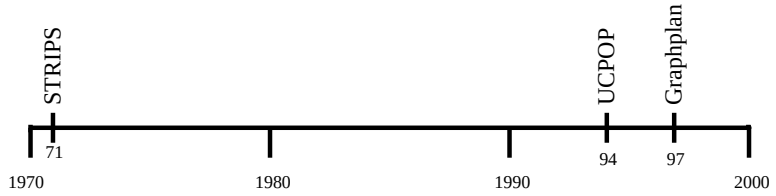
2.1 Inleiding

In dit hoofdstuk worden verschillende on en off-line planners uit de literatuur bekeken. Dit om te zien welke lessen hieruit te leren zijn en vooral welke elementen hiervan bruikbaar zijn voor MARIE. Er wordt hierbij gekeken naar twee soorten te weten de klassieke planners en de evolutionaire planners. Bij beide soorten wordt gekeken naar de oorspronkelijke toepassing, de werking en de prestatie van de planner.

2.2 Klassieke planners

Onder klassieke planners wordt verstaan de planners die voortborduren op algoritmes als A*. Deze zijn er in zowel on-line als (voornamelijk) off-line varianten. Er zullen twee behandeld worden een on-line te weten Graphplan en een off-line UCPOP. Deze planners hebben gemeen dat ze gebruikt worden voor theoretische planning problemen.

Beide kunnen gebruikt worden voor problemen in het STRIPS domein. STRIPS was een bekende planner uit de jaren zeventig. De representatie die deze planner gebruikte is een standaard geworden die het onder andere mogelijk maakt de prestaties van verschillende planners te vergelijken.



Figuur 2.1: Tijdslijn planners

2.3 UCPOP

2.3.1 Toepassing

UCPOP [18] is een planner in het theoretische domein bedoeld om theoretische problemen zoals bijvoorbeeld "towers of hanoi" op te lossen. Het hogere doel hierbij is het uitwerken van steeds efficiëntere, snellere planning technieken. Voor de theoretische werelden waarin deze planner werkt worden aannames gebruikt om deze werelden simpel te houden. Dit wordt gedaan om de nadruk te leggen op het algoritme, de wereld wordt er niet realistischer op. Zo is een van de aannames dat de wereld statisch is, er komen geen objecten bij nog mogen er objecten verdwijnen.

2.3.2 Werking

De planner gaat uit van "least commitment planning", dit betekent dat in de beschrijving alleen datgene wordt vastgelegd wat werkelijk van invloed is op het uiteindelijke doel. Het is bijvoorbeeld niet belangrijk of je nu eerst je gordel om doet of de deur dichtdoet voor je weg rijd met je auto zolang je allebei maar gedaan hebt.

UCPOP gaat uit van de volgende aannames: statisch (zie hierboven) en eindig universum van objecten, voor ieder object is een type bekend in de beschrijving van de begin-staat. Het maken van een plan gebeurt in "plan space" oftewel hij zoekt de verschillende acties die uiteindelijk de wereld manipuleren. Binnen deze plan space kan geredeneerd worden m.b.v. conjunctie, disjunctie, universele quantificatie en conditionele effecten(iedere actie heeft pre en post condities). Het algoritme gaat als volgt te werk, er is een lijst met subdoelen die in het begin bestaat uit de subdoelen die het einddoel vormen. Uit deze lijst wordt een willekeurig subdoel gekozen waarvoor een actie wordt gezocht die in zijn post condities dit doel realiseert. Van deze actie worden vervolgens de pre condities aan de lijst toegevoegd, en het subdoel wordt verwijderd. De actie mag verder niet de bestaande constraints schenden oftewel een actie mag niet een van de gerealiseerde subdoelen in gevaar brengen. Dit is omdat van doel naar start wordt gezocht, als een doel is om in de auto te zitten en de actie die gekozen wordt als gevolg heeft dat je uit de auto stapt dan kan het doel nooit meer gerealiseerd worden. Immers de gekozen actie is de laatste voor het doel (of subdoel), en kan daarna dus niet meer ongedaan worden gemaakt. Het gevolg van het toepassen van deze constraints is het verkleinen van de zoekruimte hetgeen een efficiënt algoritme oplevert. Zie [18] voor de precieze werking van het algoritme.

2.3.3 Prestatie

Tests in [2] laten zien dat UCPOP een zeer duidelijke snelheidswinst boekt ten opzichte van andere planners door zijn gebruik van constraints maar dat Graphplan (zie hieronder) nog veel sneller is dankzij een verder uitgewerkte methode voor het omgaan met constraints en nog een paar andere trucs. UCPOP is een duidelijk voorbeeld van een off-line planner.

2.4 Graphplan

2.4.1 Toepassing

Graphplan [2] wordt net als UCPOP gebruikt voor theoretische plan problemen. Hetgeen waar Graphplan verschilt is dat er een variant is die (gedeeltelijk) on-line werkt. Dit is interessant omdat hij daardoor, in combinatie met het feit dat hij probabilistische acties aan kan, dichter bij de realiteit staat dan de gemiddelde theoretische planner. Het maakt hem echter nog niet direct toepasbaar op problemen in de echte wereld, hij maakt bijvoorbeeld geen gebruik van continue tijd om maar een "klein" probleem te noemen.

2.4.2 Werking

Graphplan is gebaseerd op een gerichte gelaagde graaf. Iedere laag in deze graaf bestaat uit knopen die de acties voorstellen die op een tijdstip uitgevoerd kunnen worden. De eerste laag is de begintoestand en de laatste laag het doel. De knopen van verschillende lagen zijn met elkaar verbonden, echter verbindingen die onmogelijk tot een oplossing kunnen leiden worden zoveel mogelijk weggehaald. Voor de precieze werking van deze optimalisaties (het zijn er eigenlijk meerdere) zie [2]. Hierdoor ontstaat er een verkleinde efficiënt te doorzoeken zoekruimte. De interessantste variant op Graphplan is Tgraphplan [3], deze werkt in probabilistische zoekruimtes. In deze ruimtes kan de actie van het openen van een deur niet alleen tot gevolg hebben dat deze open gaat, maar ook bestaat er een kleine kans dat je de klink in je hand houdt (zonder de deur eraan) nadat de actie is uitgevoerd. Tgraphplan berekent niet alleen een pad met als assumptie dat het meest waarschijnlijke gebeurd, maar ook tijdens het uitvoeren van het plan paden voor het geval dat er iets onwaarschijnlijks gebeurt. Tgraphplan is een on-line planner want er worden tijdens het uitvoeren van het plan mogelijk toekomstige plan onderdelen berekent.

2.4.3 Prestatie

Tgraphplan werkt bijzonder goed bij probabilistische problemen waar er een redelijk beperkt aantal manieren bestaat om tot een oplossing te komen. De test die dit aantoont in [3] is het "probabilistic blocks" probleem. Hierbij is

het doel om stapels met blokken van een begin positie te herschikken tot de doelpositie. Dit kan met de operatoren pickup, putdown, stack, unstack en faststack, waarvan de laatste de interessante is, deze kan namelijk pickup en stack in een keer doen maar met een 70% kans van slagen. In deze test is Tgraphplan 10 tot 20 maal sneller dan een off-line variant van Graphplan, hetgeen toch wel het nut van een on-line planner aantoont.

2.5 Evolutionaire planners

Met evolutionaire planners wordt bedoeld planners die gebruik maken van evolutionaire programmeer technieken zoals genetisch programmeren en genetische algoritmen. Evolutionair programmeren werkt volgens de principes van Darwin, te weten "survival of the fittest". Er is een poel van mogelijke oplossingen, hiervan wordt de fitness gemeten, de fitste oplossingen mogen zich voortplanten door gedeeltes van hun oplossing met elkaar uit te wisselen(cross over). De andere mogelijkheid is mutatie, waarbij een gedeelte van een oplossing "spontaan" verandert. Deze "oplossingen" kunnen vele vormen aan nemen zoals een genetische codering (genetische algoritmen) programmeer code(genetisch programmeren) in het geval van de oplossingen van Peter Nordin [13] zelfs machine code.

2.6 Adaptive Evolutionary Planner/Navigator

2.6.1 Toepassing

De Adaptive Evolutionary Planner/Navigator (EP/N) [9] is een algoritme om al dan niet on-line het pad te bepalen voor een mobiele robot. In [9] wordt dit gedaan voor een gesimuleerde robot in een 2d wereld bestaande uit punten waar wel of niet over gereden kan worden. Deze wereld hoeft van tevoren niet (geheel) bekend te zijn voor het algoritme. Het is een zeer abstracte wereld waarin de taak bestaat uit het van A naar B rijden.

2.6.2 Werking

Zoals de naam al zegt werkt dit algoritme volgens een evolutionair principe. Plannen bestaan uit een linked list waarin ieder element bestaat uit x en y cordinaten en een variabele. Deze variabele bevat de informatie of dit element en/of zijn opvolger in een obstakel ligt. Er wordt een pool aangemaakt van random geinstantieerde plannen waarop evolutie theorie wordt toegepast, survival of the fittest dus. Zie [9] voor de precieze werking in dit geval, maar het gaat analoog aan andere genetische algoritmen, dus met cross-over en mutatie maar dan met een aantal domein specifieke vormen van mutatie toegevoegd. Zo is er een smooth operator die elementen die (onnodig) scherpe hoeken veroorzaken kan weghalen.

Als het algoritme in on-line mode werkt wordt het beste huidige pad gebruikt en wordt terwijl dit plan wordt afgewerkt steeds verder gezocht naar betere paden beginnend op het punt waar de robot zich op dat moment bevindt. In het geval van een nieuw tot dan toe onbekend obstakel is er dus over het algemeen meteen een pad beschikbaar om het te ontwijken en wordt er hard verder gezocht naar een nieuw optimaal pad. Het is mogelijk om als begin pool van de on-line planner de uitkomst te nemen van een off-line plan voor dat gebied. Dit vertoont dus grote overeenkomsten met de on-line methode van het in 2.4 genoemde Graphplan.

2.6.3 Prestatie

Bij de on-line variant moet tussen iedere stap die de robot doet een aantal generaties evolutie plaats vinden. Hoe meer generaties des te preciezer maar ook des te langzamer de robot. Gelukkig hoef je niet zoveel generaties te hebben voor aardige resultaten 10 tot 20 levert al mooie resultaten op. Het is dan ook niet onwaarschijnlijk dat dit algoritme in een simpele robot zou kunnen werken.

2.7 Real Time Evolution of Behavior and a World Model for a Miniature Robot using Genetic Programming

2.7.1 Toepassing

In dit artikel [13] wordt genetisch programmeren gebruikt om een echte robot rond te laten rijden. In dit geval een Khepera robot. Het enige doel van deze robot is zoveel mogelijk snel en hard rechtuit te rijden en dus niet tegen obstakels aan te knallen. Hier komt vrij weinig hoog niveau plan werk bij kijken maar er moet wel degelijk iets gepland worden. Verder blijft het systeem leren terwijl het aan het uitvoeren is en wordt daarmee dus een on-line planner.

2.7.2 Werking

In [13] worden twee versies een zonder geheugen en een met geheugen beschreven. Alhoewel degene zonder geheugen minder interessant is zal deze ook beschreven worden aangezien het de basis is voor degene met geheugen.

2.7.3 Zonder geheugen

De vorm van genetisch programmeren verschilt van de vorm die de meeste andere mensen gebruiken in het feit dat er geprogrammeerd wordt in machine taal. Het is gebruikelijker om een taal als C te gebruiken hiervoor.

Machine taal heeft echter een aantal voordelen, te weten hoge snelheid en laag geheugen gebruik. Dit maakt deze methode zeer geschikt voor simpele systemen zoals de Khepera robot.

Individuen bestaan dus uit een opeenvolging van machine instructies, verder worden de gebruikelijke methoden van cross-over en mutatie gebruikt. Bij deze operatoren wordt echter wel gecontroleerd of het resultaat syntactisch correct is. Dit is natuurlijk vrij essentieel als je op machine niveau werkt omdat anders je robot tijdens het leren kan vast lopen. De machine instructies maken gebruik van de input waarden van de acht sensoren, variabelen en de twee output waarden voor de twee motoren. Hiermee doen ze dus een functie na die acht waarden als input heeft en twee als output.

De fitness van de individuen wordt bepaald door ze gedurende bepaalde (korte) tijd controle over de robot te geven en te kijken hoe goed ze het doen. De fitness bestaat uit twee gedeeltes "pleasure" en "pain" de eerste wordt verkregen door hard rechtuit te rijden de tweede door tegen objecten aan te botsen. Hoe meer "pleasure" en des te minder "pain" hoe fitter het individu.

2.7.4 Met geheugen

Deze versie bestaat uit twee gedeeltes, het planning gedeelte dat de directe controle over de robot heeft, en het lerende gedeelte dat een wereld model evolueert om in het planning gedeelte te gebruiken. het planning gedeelte werkt als volgt:

1. lees de sensor waarden uit
2. gebruik de sensor waarden en het wereld model om de beste aansturings waarden voor de motoren te bepalen.
3. stuur de beste waarden naar de motoren
4. wacht 300ms
5. lees de sensor waarden opnieuw uit
6. bepaal aan de hand van de nieuwe waarden de fitness
7. bewaar de oude sensor waarden de gebruikte aansturings waarden en de fitness in het geheugen en begin opnieuw.

Op deze manier komt het geheugen dus vol met leervoorbeelden om het wereld model mee te trainen. Het leer gedeelte heeft de net genoemde leervoorbeelden nodig om een model te maken dat aan de hand van de sensor waarden en de aansturings waarden de fitness voorspeld. Aan gezien er maar 256 verschillende aansturings waarden bestaan voor de khepera robot doet het model dit voor alle mogelijke waarden. Aangezien er constant nieuwe

voorbeelden bijkomen is het leer gedeelte hier constant mee bezig. Het geheugen met voorbeelden is maar 50 groot dus moet worden voorkomen dat basis vaardigheden die de robot in het begin leert meteen verloren gaan, hiervoor heeft de robot een "childhood". Deze bestaat uit voorbeelden die niet zo snel vergeten worden en betrekking hebben op basisvaardigheden.

2.7.5 Prestatie

Het systeem zonder geheugen gedraagt zich ongeveer als een primitief insect dat een vreemde omgeving verkent. Hij heeft "collision avoidance" aardig onder de knie, maar een "hoger"gedrag komt er niet uit tevoorschijn. Het systeem met geheugen daarentegen ontwikkeld meerdere "hogere" strategieën waarvan er een zelfs bijna perfect is, te weten zo hard mogelijk rechtuit rijden, vlak voor de muur een half rondje draaien en weer zo hard mogelijk vooruit rijden. Zie [13] voor de precieze uitwerking en resultaten van deze tests. Dit systeem toont dus aan dat een online planner haalbaar is op een echte real-time robot.

2.8 Conclusie

Wat laten de bovenstaande planners nu zien? De klassieke planners tonen aan dat het handig representeren van de zoekruimte door het toepassen van constraints enorme snelheids winsten kan opleveren. Ook laten Tgraphplan en Adaptive Evolutionary Planner/Navigator zien dat voor dynamische en/of gedeeltelijk onbekende omgevingen een continu bijgewerkt alternatief plan grote voordelen heeft. Dit laatste is vooral goed voor de schaalbaarheid als het probleem ingewikkelder wordt. De geheugen versie van [13], Adaptive Evolutionary Planner/Navigator en Tgraphplan laten verder zien dat het nuttig kan zijn om van tevoren al plannen uitgewerkt te hebben. Dit kan dus in de vorm van een "childhood" [13], een pool met individuen getraind op wat er van de omgeving van tevoren bekend is [9], of een off-line bedacht (ideaal)plan [2]. Tenslotte laten bovenstaande voorbeelden duidelijk zien dat voor het domein van autonome robots in een dynamische (echte) omgeving een on-line planner beter presteert dan een off-line planner.

Hoofdstuk 3

MARIE

3.1 Hardware

3.1.1 Onderstel

De basis van MARIE wordt gevormd door een Vessa Trekka invalide karretje.

Figuur 3.1: de Vessa Trekka, basis van MARIE

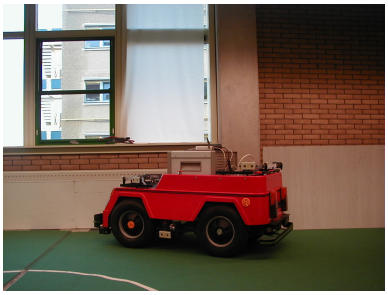
Hieraan is het nodige verbouwd, hieronder zal een aantal van de belangrijkste veranderingen genoemd worden, de basis is echter nog duidelijk herkenbaar.

3.1.2 Accu's

Vanwege alle extra elektronica aan boord, en om de gebruikstijd te vergroten zijn de oorspronkelijke twee accu's vervangen door 3 veel grotere. Er zijn er twee voor de motoren en er is er een speciaal voor de extra elektronica.

3.1.3 Ultrasoon sensoren

MARIE is voorzien van twee soorten ultrasoon sensoren, 12 Robosoft en 8 Polaroid sensoren. De Robosoft sensoren zijn de witte sensoren die bovenop de robot gemonteerd zijn, deze worden voornamelijk gebruikt voor taken als collision detection. De Polaroid sensoren zijn de zwarte op bumperhoogte, deze worden voornamelijk gebruikt voor het in kaart brengen van de omgeving. Recentelijk zijn de Polaroid sensoren verplaatst van de bovenkant van de robot naar de huidige positie. De reden hiervoor is dat in het robolab tegenwoordig een robosoccer voetbalveld ligt dat veel lagere wanden heeft dan de daarvoor gebruikte (tijdelijke) triplex wanden. Op de oude positie keken de polaroid sensoren over de nieuwe wanden heen hetgeen testen erg onpraktisch maakte.



Figuur 3.2: links van MARIE de nieuwe wand, rechts de oude

3.1.4 Computer

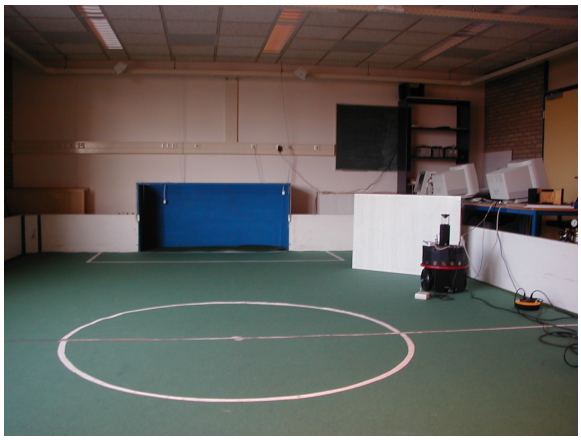
De computer van MARIE bestaat uit een VME-crate (de grote witte kast in het midden) met daarin een Motorola 68020 processor met 4MB geheugen en enkele controllers voor het aansturen van de robot. Sinds kort is MARIE voorzien van een Breezenet wireless netwerkverbinding, zodat ze in constant contact kan staan met workstations op het netwerk.

3.1.5 Encoders calibratie banden ventielen

Alle vier de wielen zijn voorzien van hoekencoders voor positie bepaling door middel van dead reckoning. Om dit goed te laten werken moeten de wielen een vaste radius hebben, hiervoor moeten de banden regelmatig opgepompt worden, en moet het systeem gekalibreerd worden.

3.1.6 Bumpers en noodstop

Ter beveiliging zitten er op MARIE bumpersensoren die een botsing kunnen detecteren alsmede vier rode stop knoppen die de robot in een keer stilzetten. De bumpersensoren geven een signaal naar de software dat er een botsing plaatsvindt en geeft de software de mogelijkheid daar iets aan te doen. De stop knoppen verbreken de stroomvoorziening van de motoren zodat de robot meteen stil staat.



Figuur 3.3: Robotlab

Figuur 3.4: MARIE A:breezenet wireless netwerk B: computer C: Robosoft ultrasoon sensoren D: stop knop E: bumber sensor F: Polaroid ultrasoon sensoren

3.2 Software

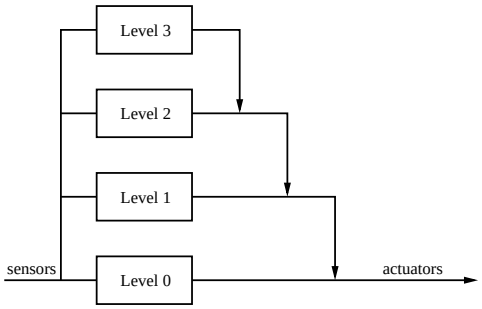
De software die MARIE aanstuurt is gebaseerd op een hybride architectuur. Deze heeft elementen van architecturen gebaseerd op functionele decompositie en van architecturen gebaseerd op gedrag decompositie. Als eerste zal worden uitgelegd hoe de functionele en gedrag decomposities werken, dit om als laatste te kunnen laten zien waarom de hybride architectuur voor MARIE geschikter is.

3.2.1 Functionele decompositie

Bij deze vorm van decompositie wordt de functionaliteit van een robot onderverdeeld in sub-modules. Deze sub-modules hebben allemaal een specifieke functie binnen de robot bijvoorbeeld pad planning of aansturing van de robot. Via een communicatie protocol zorgen deze modules samen voor het gedrag van de robot. Deze vorm van organisatie zorgt ervoor dat de complexiteit binnen de perken wordt gehouden en binnen modules makkelijk aanpassingen zijn te maken. Problemen zij er echter voldoende die ervoor zorgen dat deze aanpak minder geschikt is voor een autonoom systeem. Modules zijn afhankelijk van elkaar, dit heeft tot gevolg dat als er in een module een fout optreed dit ernstige gevolgen heeft voor het functioneren van het hele systeem. Er is geen sprake van "gracefull degradation" als een onderdeel de fout in gaat is er geen enkel alternatief dat dit onderdeel zelfs maar op beperkte wijze kan vervangen. De architectuur wordt top down ontworpen en moet dus in een keer goed zijn, simpele wijzigingen naderhand kunnen ingrijpende gevolgen hebben. Bovendien kan er pas getest worden als er een werkende versie van alle modules is. Doordat alle modules over het algemeen bij een actie van de robot betrokken moeten worden levert dit een slechte reactie tijd op, hetgeen voor een real-time systeem zoals een autonome robot niet erg geschikt is.

3.2.2 Gedrag decompositie

In het geval van gedrag decompositie wordt de software van de robot onderverdeeld in meerdere vormen van gedrag. Deze architectuur wordt van onderaf opgebouwd waarbij dus begonnen wordt bij de essentiele vormen van gedrag die bijvoorbeeld zorgen voor "collision detection" en andere reflexen. Hierboven kunnen dan hogere vormen van gedrag geplaatst worden. Een goed voorbeeld hiervan is de subsumbtion architectuur.



Figuur 3.5: Subsumbtion architectuur

Dit lost een aantal problemen van de functionele decompositie op. De lage niveau gedrags vormen zorgen voor goede reflexen essentieel voor een autonome robot in een (gedeeltelijk) onbekende en dynamische omgeving. Als een hogere vorm van gedrag faalt kan dit gedeeltelijk worden opgevangen door de lagere niveaus.

Er kleven echter ook aan deze aanpak nadelen. Het is niet makkelijk om op hoog niveau directe opdrachten aan de robot te geven, dit vereist namelijk aanpassing van het gedrag en dit levert niet altijd even voorspelbare resultaten op. Het is dus zeer moeilijk om met deze methode het gewenste complexe gedrag te verkrijgen.

3.2.3 Hybride architectuur

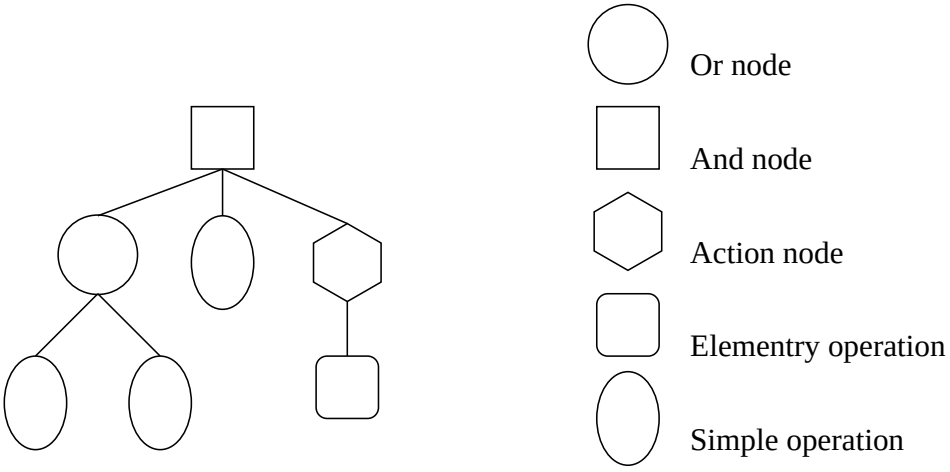
Een hybride systeem is onderverdeeld in subsystemen die expertise hebben op het gebied van een bepaald probleem. Om te zorgen dat deze systemen niet al te afhankelijk van elkaar worden is de communicatie tussen de subsystemen tot een minimum beperkt. Goede reflexen kunnen behaald worden door de subsystemen die zich met lage niveau taken bezig houden gedrag gebaseerd te laten zijn. Op hoge niveaus worden symbolische redeneringen gebruikt om complex gedrag te programmeren. Dit is makkelijk te manipuleren en levert voorspelbare resultaten op. In theorie levert deze aanpak dus een robuuste reflexieve robot op waar makkelijk opdrachten aan te geven zijn. Bij MARIE is de hybride architectuur zeer duidelijk aanwezig. Modules op laag niveau zoals de collision avoidance zijn duidelijk op gedrag gebaseerd en hebben een snelle reactie tijd, hoge niveau modules zoals de pad planner werken met een symbolische representatie die makkelijk directe opdrachten kan verwerken. Het hoogste niveau wordt gevormd door de on-line planner, het doel van deze opdracht. Ook de communicatie wordt binnen MARIE tot een minimum beperkt. Er is een centraal geheugen waarin belangrijke informatie zoals paden en omgevings informatie wordt opgeslagen met als gevolg dat iedere module zijn data maar naar een plek hoeft te sturen. Hierdoor zijn de modules ook minder afhankelijk van elkaar doordat het niet uitmaakt waar de informatie vandaan komt of waar hij heen moet. Als er een module

veranderd wordt vereist dat dus in het algemeen geen aanpassing van andere modules.

3.2.4 Grammatica van Plannen voor MARIE

Een plan voor MARIE wordt op een bepaalde manier gerepresenteerd, hoe deze representatie werkt zal in het komende hoofdstuk worden uitlegd. Aan deze representatie zelf is niets veranderd en is dus dezelfde als beschreven in G. A. Den Boer [4]. Desondanks is het belangrijk om te weten hoe deze in elkaar steekt als je de on-line planner voor MARIE bekijkt.

Een plan wordt gerepresenteerd door een boomstructuur bestaande uit een aantal "nodes" (knopen). Van deze nodes bestaan verschillende types te weten: AND nodes, OR nodes, Action nodes, Elementry Operation nodes en Simple Operation nodes. De laatste 2 types vormen altijd de bladeren van de boom, dit zijn de acties die daadwerkelijk uitgevoerd worden. De eerste 3 types bepalen in welke volgorde en wanneer de bladeren uitgevoerd worden.



Figuur 3.6: Voorbeeld van een boomstructuur

Gedurende de executie van een plan wordt de huidige status van de robot bijgehouden. Dit gebeurd door middel van status flags, sommige van deze flags hebben betrekking op het gehele plan en zijn dus global, andere hebben allen betrekking op het gedeelte dat nu uitgevoerd wordt en zijn dus local. Iedere AND, OR en Action node heeft initial, during en final conditions, waar de huidige status aan moet voldoen voor een succesvolle executie. Het plan is succesvol als aan de final condities van de root node is voldaan. De root node staat boven aan de hierarchie van de plan boom, alle andere nodes zijn dus afstammelingen van deze "pater familias" van het plan. Iedere node kan de huidige status flags manipuleren, dit kan met andere status flags als voorwaarde. Deze manipulatie kan alleen plaats vinden van kinderen

naar ouders. Dit heeft tot gevolg dat aan iedere status verandering een Simple Operation of een Elementry Operation ten grondslag ligt. De status veranderd dus alleen naar aanleiding van datgene de robot daadwerkelijk gedaan heeft. Hoe werken de verschillende nodes nu precies? Hier volgen de beschrijvingen.

3.2.5 AND node

De kinderen van de AND node moeten op volgorde worden uitgevoerd. De final condities van de uitgevoerde (kinder) node moeten overeenkomen met de initial condities van de daaropvolgende node. De enige uitzondering hierop is wanneer een van de kinderen die niet het laatste kind is met zijn final condities al voldoet aan de final condities van de AND node. In dat geval wordt de AND node succesvol beindigd en worden kinderen die nog niet aan bod zijn gekomen niet uitgevoerd. Als ergens de final en initial condities van twee kinderen niet overeenkomen faalt de node, zo ook wanneer de final condities van het laatste kind niet overeenkomen met die van de AND node zelf. Verder faalt de node als tijdens de executie de during conditions van de node geschonden worden. De root node is per definitie altijd een AND node.

3.2.6 OR node

Or nodes hoeven niet in een strikte volgorde te worden uitgevoerd. De kinderen worden een aantal malen (door de maker van het plan te bepalen hoe vaak) bij langs gelopen, en de eerste node waarvan de initial condities overeenkomen met de huidige status wordt uitgevoerd. Dit gebeurd totdat aan de final condities van de OR node voldaan is, in dat geval slaagt de node namelijk. Wordt niet aan de final condities voldaan na het (meerdere malen) bij langs gaan van de kinderen dan faalt de node.

3.2.7 Action node & Elementry Operations

De Action node kan als kinderen alleen Elementry Operations hebben. Deze Elementry operations (als het er meerdere zijn) worden gelijktijdig uitgevoerd. De Action node zit dus altijd precies 1 niveau hoger dan de bladeren van de boom. Elementry nodes kunnen ook alleen voorkomen onder een Action node. Deze twee type nodes zijn dus onlosmakelijk met elkaar verbonden.

3.2.8 Simpel operation

Een simpel operation is een actie van de robot die niet gelijktijdig met een andere uitgevoerd hoeft te worden. Voorbeelden van simple operations zijn stoppen, niets doen, een positie opslaan of het geheugen leeg maken.

Dit type node is altijd een blad van de boom, het komt dus niet hoger in de structuur voor. De ouders van dit type node zijn AND of OR nodes. Net als de action nodes zijn simple operations verantwoordelijk voor een daadwerkelijke verandering in de staat van de robot.

Hoofdstuk 4

Ontwerp beslissingen

4.1 Ontwerp van een on-line planner voor MARIE

Hieronder worden de ontwerp beslissingen voor de on-line planner uitgelegd. Eerst worden de overwegingen en beslissingen over de zoekruimte en het algoritme geven, vervolgens volgt het uiteindelijke ontwerp.

4.2 Afwegingen zoekruimte

De zoekruimte dicteert grotendeels de aanpak van een planner, uit de vorm van de zoekruimte volgt het type algoritme dat hiervoor geschikt is. Het omgekeerde is natuurlijk ook waar, uit de keuze voor een bepaald algoritme volgt in grote lijnen de vorm van de zoekruimte. Een zoekruimte moet de werkelijke wereld op een efficiënte manier samenvatten, zodat relevante informatie snel gevonden wordt.

4.2.1 Plan bomen of Laag niveau

Maakt de planner gebruik van plan bomen of gaat hij op een lager niveau werken? Beide hebben hun voordelen, als je gebruik maakt van de planbomen is het grootste gedeelte van de datastructuur al aanwezig. Er is al een routine om te controleren of een boom correct is, de elementen om naar te zoeken, de knopen in de boom zijn al gedefinieerd. De boomstructuur is uitermate geschikt voor genetische methoden die vaak worden gerepresenteerd in een boomstructuur. De flags in de structuur zijn initial final (en during) condities die ook gebruikt worden in veel strips domein planners.

Echter als je minder "overhead" wil hebben kan het verstandig zijn om op een lager niveau te gaan werken, bijvoorbeeld direct op het input output niveau van de virtual cart[1]. Hiermee negeer je dus modules als de wall-

[1]De software laag die een abstractie maakt van de werkelijke sensoren en actuatoren

sensor volledig. Dit is vooral verstandig als je een zoekalgoritme wilt gaan gebruiken dat grote hoeveelheden data doorzoekt, of als je het belangrijk vindt dat de planner "nu" geheel zelfstandig op MARIE werkt.

Werken op laag niveau biedt interessante mogelijkheden maar druist tegen de structuur van MARIE in zoals die nu bestaat. Bovendien is het zonde een mooie structuur zomaar weg te gooien, dus is ervoor gekozen gebruik te maken van planbomen.

4.2.2 Foute bomen

Hoe beperkt wordt de zoekruimte, worden "foute" bomen toegestaan? Het toestaan van "foute" bomen heeft voordelen, vooral in genetische oplossingen. Een "foute" boom kan immers een mutatie verwijderd zijn van een geweldige oplossing. Voor "klassieke" planners heeft het toestaan van "foute" bomen geen enkel nut. Voor klassieke planners wil je de zoekruimte juist zoveel mogelijk beperken om zo snel mogelijk een goede oplossing te vinden. Er is gekozen voor het niet toestaan van foute bomen omdat er in de nabije toekomst nog geen gebruik van genetische methoden gemaakt gaat worden. Mocht dit in een latere uitbreiding wel gebeuren dan moet het gebruik van "foute bomen" natuurlijk heroverwogen worden.

4.2.3 Welke functies

Gaan alle mogelijke functies gebruikt worden? Zijn alle functies strikt noodzakelijk voor het maken van succesvolle plannen? Sommige functies zijn vrijwel zeker niet nodig in een on-line planner, zoals bijvoorbeeld de functie "user-input". Het is nuttiger om een paar functies erbij te maken specifiek voor de on-line planner. Een mooie manier om dit laatste te doen is via macro's. Dat zijn deel bomen die een standaard functie uitvoeren, zoals een initialisatie. Het is immers onzinnig om een zoekalgoritme een goede initialisatie te laten vinden als daar al een volledige oplossing voor beschikbaar is. Het doel van de macro's is dus het toespitsen van de zoekruimte op het probleem waar geen voor de hand liggende oplossing voor is. Het is niet nodig om het wiel opnieuw uit te vinden, het is wel nuttig om uit te vinden waar je met dat wiel kan komen. Hoeveel meer macro's dan alleen de initialisatie er nodig zijn moet nog blijken, maar er zijn zeker meer voor de hand liggende voorbeelden. Zo is het erg handig om als je de pad planner aanroept het gevonden pad ook op te slaan. Macro's zijn dus bedoelt voor functies die pas een zinnige actie vormen als ze gecombineerd worden met andere functies. De macro's moeten echter zeker niet te groot worden, anders lopen ze het risico niet meer algemeen toepasbaar te zijn. Hoe meer functies bij een macro betrokken zijn hoe meer specifieke informatie er nodig is. Ook kunnen er macro's gemaakt worden die kant en klare exception

naar virtuele versies hiervan. Deze zijn makkelijker aan te sturen en uit te lezen.

plannen bevatten zoals: rij een meter terug en probeer opnieuw. Macro's kunnen het makkelijkst geïmplementeerd worden als kleine scenario's met eventueel bijbehorende data.

4.2.4 Te gebruiken functies

Hieronder volgen de functies die de planner gaat gebruiken. De functies die de planner niet gebruikt kan de gebruiker nog steeds toepassen in zijn (onvolledige) plannen, echter ze behoren niet tot de zoekruimte. De afwegingen welke functies wel te gebruiken in de zoekruimte en vooral welke niet, volgen hieronder.

Functies die hoofdzakelijk alleen gebruikt gaan worden

5001, "SO TST dist - Test target distance"
5002, "SO OP stopQ - Quick stop vehicle"
5003, "SO CA mode - Set ca_mode 0/1/2/3"
5013, "SO OP stopS - Stop Gradual"

Bovenstaande functies produceren in hun eentje al een "nuttige" actie ze zijn niet afhankelijk van extra data die de data-manager in of uit moet komen. Ze worden dus gebruikt als macro's bestaande uit 1 functie. Ze kunnen gebruikt worden in grotere macro's in principe zouden ze echter wanneer ze uit deze macro's weggelaten worden er door de on-line planner weer ingestopt worden. Het is voor het behoud van overzicht dus niet uitgesloten dat ze in grotere macro's worden opgenomen, voor de werking van de on-line planner is dit echter in principe niet noodzakelijk.

Functies die hoofdzakelijk in macro's gebruikt worden

5000, "SO DM store - Store a record to DM"
5004, "SO VC reset - Reset virtual cart"
5006, "SO VC posit - Set Position"
5008, "SO TC path - Store TC path"
5009, "SO SC path - Store SC path"
5010, "SO PP 2pts - Store PP data (2 pts)"
5011, "SO PP goto - Store PP data (from cur)"
5012, "SO DM purge - Purge data from a DM"
5014, "SO DM clean - Purge most data from DM"
5015, "SO FZ path - Store FZ path section"
5016, "SO DO noth - Do nothing"

Bovenstaande functies hebben bijna allemaal te maken met het opslaan van data in de data-manager. Aangezien je iets nuttigs moet hebben om op te slaan en omdat een aantal functies specifiek samengaat met andere functies

heeft het alleen nut om deze te gebruiken in combinatie met andere functies. Dit is vooral van toepassing als er een random zoekmethode wordt gebruikt, immers het bijvoorbeeld zomaar opnieuw bepalen van de huidige positie levert in de meeste gevallen alleen maar onzin op.

Functies die niet gebruikt worden

5005, "SO UI disc - Disconnect cable"
5007, "SO UI input - User input"
6000, "SO OP fwd.p - Forward path"
6001, "SO OP rev.p - Reverse path"
6002, "SO OP lemni - Lemniscate path"
6003, "SO OP video - Video path"

Disconnect cable is niet meer nodig vanwege het wireless netwerk, bovendien is hij onwenselijk als je een proces dat niet op MARIE draait mee laat doen in het plannen.

User input is niet meer nodig aangezien we hier een autonoom systeem willen maken. Het is natuurlijk nuttig voor debuggen maar in autonoom systeem neemt het een gedeelte van de zelfstandigheid weg vandaar dat het voor de on-line planner niet gebruikt zal worden. Video path wordt niet gebruikt omdat er geen camera meer op MARIE is gemonteerd. De functies 6000, 6001 en 6002 worden niet gebruikt omdat ze te specifiek zijn en eventueel (met wat extra werk) ook door andere functies gedaan kunnen worden. Deze functies voeren een van te voren gespecificeerd pad uit. Deze zullen maar in zeer weinig situaties toepasbaar zijn ze worden dus niet uit principe weggelaten maar meer omdat ze zorgen voor extra "rommel". Ze zouden in een aangepaste versie waarin ze een voor de on-line planner nuttig pad herbergen wel terug kunnen komen. Op deze manier zou er een simple operation kunnen zijn voor rijd een meter achteruit toepasbaar als er in de huidige situatie geen weg "vooruit" meer is.

4.3 Afwegingen Algoritme

Een zoek algoritme moet binnen de zoekruimte zo efficiënt mogelijk een plan zien te vinden. De beste aanpak hiervoor kan nogal verschillen afhankelijk van het soort plan dat gevonden moet worden. Een kort sub plan stelt heel andere eisen dan een volledig plan met enige "redundancy" erin.

4.3.1 Volledige of gedeeltelijke plannen

Wat voor plannen kun je maken? Je kan het algoritme een volledig plan laten maken, of je kan het gedeeltelijke plannen laten maken om een (in)compleet plan van de gebruiker aan te vullen. Een volledig plan maken is natuurlijk

de mooie oplossing, het vereist echter veel rekentijd en een geavanceerd algoritme omdat de zoekruimte zo groot is. Als je (in)complete plannen aanvult dan kan je beginnen met een veel simpeler algoritme en dit later uitbouwen tot een geavanceerder algoritme. Dit heeft als voordeel dat je niet meteen in het diepe springt en dus beter een begrip kan opbouwen voor wat er wel en niet werkt. Daarom is dit dan ook de aanpak die genomen wordt.

4.3.2 Semantiek

Voor de zoek methode is het belangrijk dat alle initial en final condities hetzelfde blijven betekenen, daarom zullen de conventies die nu gelden hard afgedwongen moeten worden, verder moeten de knopen waarnaar gezocht wordt niet zeggen dat ze dingen doen die ze niet kunnen. Het mag natuurlijk niet gebeuren dat het algoritme een functie vindt die de betekenis van succes en falen omdraait en dan door niets uit te voeren het doel "bereikt". Om dit soort problemen te voorkomen wordt de zoekruimte dus beperkt. Dit gebeurd door het van tevoren maken van macro's van 1 of meer knopen waarin bovenstaande eisen worden verwerkt. Dat wil zeggen initial en final condities (flags) veranderen niet van betekenis, en geven bovendien weer wat de functie in werkelijkheid doet. Het algoritme mag dus niet zoeken op het niveau waar functies hun initial en final condities kunnen veranderen.

4.3.3 Herplannen

Herplannen kan je continu doen voor het geval de volgende stap mis gaat of je kan hiermee pas beginnen als er werkelijk iets fout is gegaan. Voor volledige plannen is het beter om off-line een basis plan te maken en dat eventueel on-line aan te passen. In dit geval maakt het voor het "herplan" gedeelte niet uit of het oorspronkelijke plan van het algoritme of een gebruiker is gekomen. Een bijkomend probleem zijn vastgelopen planbomen, hoe krijg je uit een vastgelopen boom de informatie die je nodig hebt voor het herplannen. Je moet immers de huidige positie en de doel positie uit de vast gelopen boom weten te ontfutselen wil je op een zinnige manier kunnen herplannen. Er zijn twee oplossingen hiervoor, of je gebruikt een soort monitor die, behalve met het zoeken naar alternatieven voor de volgende stap, zich ook bezig houdt met het bijhouden van de huidige positie, of je zorgt dat bij het optreden van een error altijd de huidige en de goal positie worden teruggegeven. Het plan is om eerst het herplannen pas te laten gebeuren als er iets fout gaat, en later een uitbreiding te maken die continu zoekt naar alternatieven voor het geval er in de volgende functie iets fout gaat. In beide gevallen wordt ervoor gezorgd dat tijdens errors de huidige en goal positie worden teruggegeven.

4.3.4 Genetisch of klassiek

Voor het maken van deze plannen kunnen genetische methoden gebruikt worden, deze zijn als er gebruik wordt gemaakt van plan bomen echter wel zeer rekenintensief en zullen niet in real-time op MARIE zelf kunnen draaien. Ook kan een "klassiek" algoritme gebruikt worden, hetgeen efficiënter is maar minder makkelijk met dynamische omgevingen kan omgaan, en daarom makkelijker kan komen vast te zitten. Als eerste zal daarom een methode met random kiezen van functies gebruikt gaan worden, dit voorkomt vastzitten door bijvoorbeeld het steeds volgen van een zelfde patroon. Met een random methode kan je natuurlijk niet erg diep zoeken, dus als uitbreiding hierop kan een klassiek algoritme geimplementeerd worden. Als laatste uitbreiding kan een genetische methode gebruikt worden, maar zoals de afwegingen over de zoekruimte laten zijn, vereist dit vele aanpassingen.

4.4 Uiteindelijk ontwerp

4.4.1 Basis

De te maken on-line planner zal bedoelt zijn om te werken met gedeeltelijke of complete plannen. Hij zal "gaten" in gedeeltelijke plannen kunnen dichten om er een volledig plan van te maken. Als een compleet plan faalt zal hij het aan kunnen passen zodat het toch tot een oplossing komt. In beide gevallen kan dat door het zoeken van nieuwe functies die met hun initial condities aan de huidige voorwaarden voldoen. Verder is het voor het "redden" van gefaalde plannen ook mogelijk om een standaard exception plan uit te voeren. De zoek methode zal random kiezen zijn, aangezien dit een goede beveiliging biedt tegen loops of het herhaaldelijk dezelfde fout maken.

4.4.2 Uitbreidingen

Als de zoekruimte goed blijkt te werken kan er in plaats van random een geavanceerd zoekalgoritme worden toegepast om eventueel complete plannen on-line te genereren. Een tweede uitbreiding is het continu zoeken naar alternatieven terwijl het plan aan het uitvoeren is. Als laatste kan over worden gegaan op een systeem met genetische zoekmethode, hiervoor moet echter zoveel omgegooid worden dat het niet binnen dit plan past. Deze laatste uitbreiding zal dan ook zeker niet binnenkort uitgevoerd worden.

4.4.3 Verwachtingen

Uit dit plan vallen de volgende hypotheses op te maken over het presteren van dit plan in de werkelijkheid.

- Het basisplan is geschikt om in (gedeeltelijke) plannen gaten op te vullen.

- Om redenen van snelheid en complexiteit is voor het vinden van complete plannen een geavanceerder algoritme dan random kiezen nodig.

In de tests en de conclusie van dit onderzoek zal blijken of deze hypotheses ook uitkomen.

Hoofdstuk 5

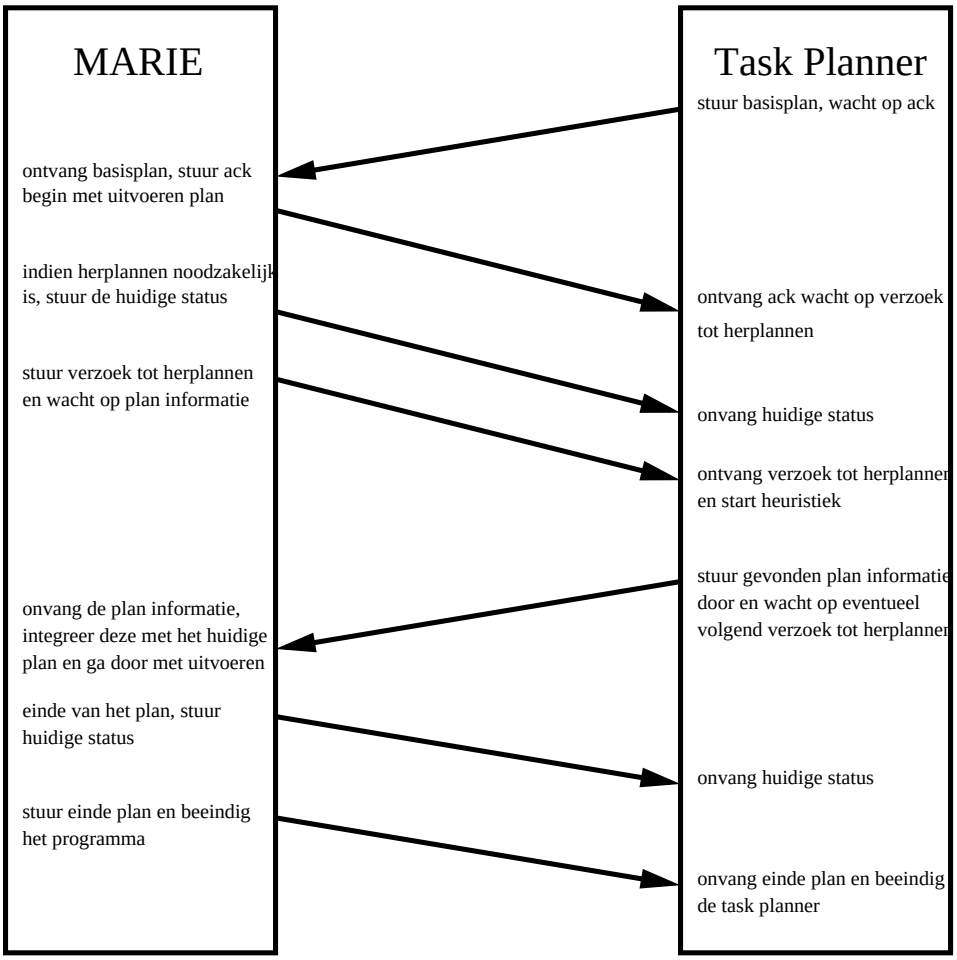
Implementatie

5.1 Kort overzicht

Bij de implementatie is ervoor gekozen het "on-line plannen" buiten MARIE te doen. Dat wil zeggen op een andere computer die een netwerkverbinding met MARIE heeft. Hiervoor moest er dus een communicatie protocol komen tussen MARIE en deze losse computer. Dit is het eerste onderdeel wat behandeld zal worden, verder moet er door MARIE bepaald worden wanneer de on-line planner wordt aangeroepen en moet er de eigenlijke "on-line planner" zijn. Deze twee onderwerpen komen na het communicatie protocol aan bod. Hierna wordt de functionaliteit beschreven, oftewel wat deze programmatuur zal kunnen doen. Als laatste komen mogelijke uitbreidingen en verbeteringen aan bod.

5.2 Communicatie protocol

Voor de communicatie wordt dezelfde verbinding gebruikt die wordt gebruikt voor het versturen van het plan. Dit is een socket verbinding tussen MARIE en een andere computer op het netwerk waar de "task planner" wordt uitgevoerd. Na het succesvol oversturen van de data stuurt MARIE een acknowledgement, en sluit de task planner zich. Althans zo ging het voordat begonnen werd met het implementeren van een on-line planner. Nu wordt de verbinding na het sturen van het plan opengehouden totdat MARIE klaar is met het uitvoeren van het (al dan niet on-line veranderde) plan. Hiervoor was het nodig om extra commando's over te kunnen sturen, te weten een verzoek om het plan aan te passen en een bericht dat het plan voltooid is. Ook is het voor het algoritme nodig te weten wat de huidige status van MARIE is zodat een geschikte aanpassing van het plan te kunnen vinden. De status moet dus ook verstuurd worden van MARIE naar de "task planner".



Figuur 5.1: Communicatie protocol

5.2.1 MARIE

MARIE beslist wanneer het huidige plan aangepast moet worden. In de implementatie kan dit niet op alle mogelijke plekken in een plan. Het aanpassen van het plan kan alleen in AND en OR nodes die al tenminste 1 “child-node” hebben. Als van te voren is bepaald dat op die plaats de taskplanner moet worden aangeroepen dan is dit een lege node. Met een lege node wordt bedoeld een “simple operation” (zie 3.2.6) die niets doet. Waarom is dit nuttig? Als in plaats van een lege node een met een functie gebruikt dan krijgt je macro een andere betekenis. Een node die niets doet voegt alleen maar de functionaliteit van het mogelijk herplannen toe.

Op het moment dat een van de AND of OR nodes (met tenminste een child node) niet geslaagd is, dus nog niet aan zijn final condities voldoet, dan wordt het on-line plan gedeelte van de taskplanner aangeroepen. Dit

mag natuurlijk niet tot in het oneindige gebeuren dus is er een maximaal aantal keren dat de on-line planner binnen 1 node mag worden aangeroepen.

Als alles goed gaat dan krijgt MARIE kort na het verzoek het plan te veranderen de extra plan informatie binnen. De zojuist verkregen informatie wordt dan in het huidige plan opgenomen en de uitvoering van het nu aangepaste plan wordt hervat.

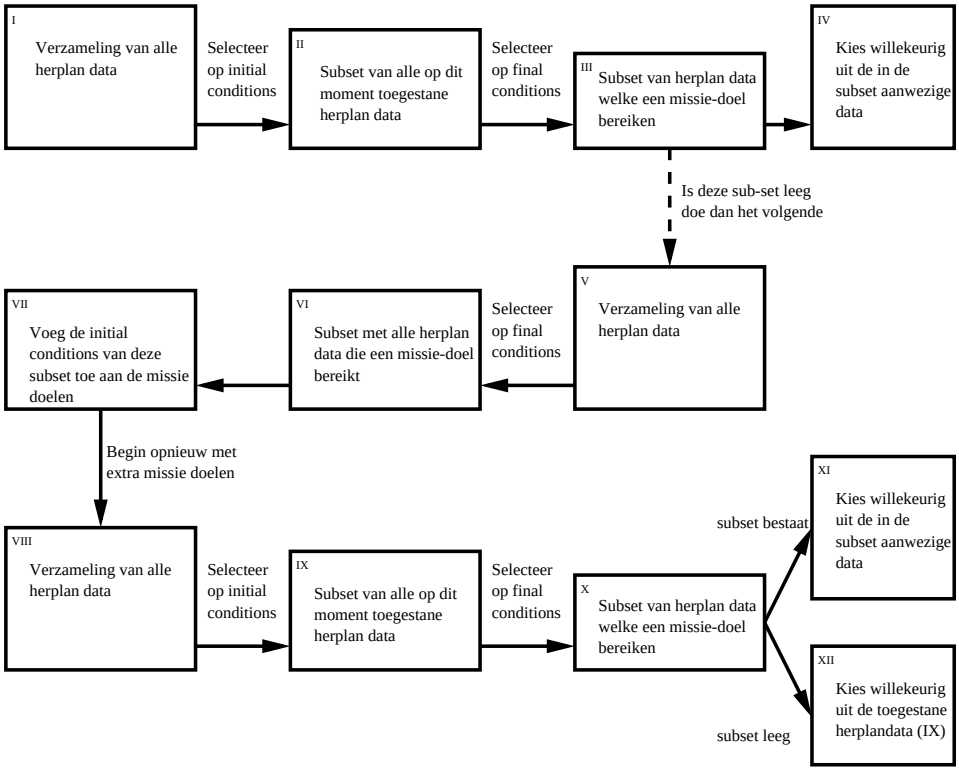
5.2.2 Task planner

De on-line planner is een programma onderdeel dat kan kiezen uit een verzameling van losse plan onderdelen. Sommige van deze onderdelen bestaan uit één actie, andere zijn "macro's" van meerdere acties. Al deze onderdelen zijn gemaakt volgens de plangrammatica (zoals beschreven in 3.2.4). De basis voor de heuristiek is simpel random kiezen, dit is echter uitgebreid met elementen uit UCPOP [18]. Wanneer de on-line planner wordt aangeroepen wordt bekeken welke van de plan onderdelen gebruikt mogen worden. Dit gebeurt aan de hand van de huidige status van MARIE die met het verzoek tot herplannen wordt meegestuurd. Tijdens dit proces wordt meteen gekeken of er plan onderdelen zijn die niet alleen toegestaan zijn, maar die ook nog een van de final condities (ofwel einddoelen) bereiken. Als deze nuttige onderdelen er zijn dan wordt hieruit gekozen, zijn ze er niet dan wordt er gezocht naar onderdelen die de juiste voorwaarden creren voor een onderdeel dat wel een final conditie bereikt. De voorwaarden worden toegevoegd aan de final condities, en er wordt random gekozen uit de onderdelen die deze voorwaarden creren. Dit alles dus naar het idee uit UCPOP [18]. Werkt al deze heuristiek niet dan wordt er random gekozen uit de toegestane onderdelen die niets bijzonders bereiken.

Is het plan onderdeel eenmaal gekozen dan wordt het verzonden naar MARIE en wordt er gewacht op het volgende verzoek tot herplannen of natuurlijk het bericht dat MARIE klaar is met uitvoeren.

5.2.3 Vergelijking OSI

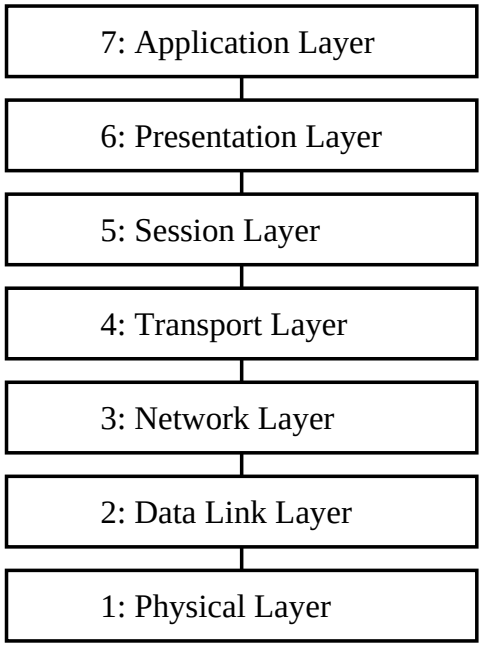
In het hieraan voorafgaande stuk is al uitgelegd hoe het protocol werkt, dit stuk gaat over hoe het protocol zich vergelijkt met het OSI model en over de toepasbaarheid van het protocol. Het protocol is een aanpassing van een al bestaand protocol en bedoeld voor communicatie tussen 2 modules. Het is dus niet geschikt voor communicatie met meerdere task planners of meerdere robots. Mocht dat soort communicatie in de toekomst nodig zijn dan moet het protocol uitgebreid worden. Hoe verhoudt dit protocol zich tot het OSI referentie model? Om deze vraag te kunnen beantwoorden moet eerst even kort uitgelegd worden wat dit OSI model is. De afkorting OSI staat voor Open Systems Interconnection. Dit model geeft aan uit welke onderdelen een communicatie protocol zou moeten bestaan.



Figuur 5.2: Algoritme on-line planner

Het communicatie protocol in MARIE bevat elementen uit de bovenste vier lagen. De transport en session layer komen vrijwel niet terug in het protocol aangezien er op het moment alleen een zeer simpele connectie tussen twee modules over een betrouwbaar netwerk is. In feite wordt alleen op zeer simpele manier een connectie tot stand gebracht zoals te zien valt in figuur 5.1.

Een element uit de presentation laag is aanwezig in zoverre dat de verzoeken tot herplannen en beëindigen van de verbinding zijn gecodeerd in het data type van de status. Vandaar dat MARIE eerst de huidige status doorstuurt en dan apart het verzoek tot herplannen. Er wordt alleen data van het type van de status overgestuurd dus kan dit niet in één bericht. Het data type van de status is een array van twee unsigned longs, waar bij element 0 de 32 global status flags zijn en element 1 de 32 local status flags. Een verzoek tot herplannen is een status waarbij element 0 de waarde nul heeft, dit is namelijk een waarde die MARIE in de praktijk nooit als global status zal hebben. Het bericht voor het einde van het plan is een maximale waarde voor element 1, ook dit komt in de praktijk nooit voor als status van MARIE.
Het protocol behoort echter voornamelijk toe aan de Applicition laag. Aan-



Figuur 5.3: OSI referentie model

gezien het hier gaat om een toepassing die van een netwerkverbinding gebruik maakt.

5.3 Functionaliteit

Wat kan dit algoritme bereiken? Het moet in ieder geval een skelet van een plan hebben om mee te werken aangezien het alleen child-nodes kan toevoegen. Met een absoluut basis plan zou het in principe alle nuttige plannen kunnen maken ervan uitgaande dat de losse plan onderdelen (macro's) slim gekozen worden en goed in elkaar zitten. Dit is in de praktijk minder haalbaar gezien de niet zo krachtige aard van het algoritme, dat wil zeggen het kan niet zelf child nodes aanmaken, of nodes weghalen. Hierdoor moet er erg veel moeite gestoken worden in het uitzoeken en maken van de losse plan onderdelen. Het is realistischer om de on-line planner te gebruiken bij het uitvoeren van onvolledige plannen. Deze plannen missen enige cruciale onderdelen, maar een duidelijke basis structuur is al aanwezig.

5.4 Mogelijke uitbreidingen/verbeteringen

Er zijn natuurlijk nog vele uitbreidingen en verbeteringen mogelijk voor de on-line planner, er zullen hier kort een aantal besproken worden.

5.4.1 Uitsluiten van crashende plannen

Bepaalde plannen zorgen er gegarandeerd voor dat MARIE vastloopt. (b.v. 2x achter elkaar vooruitrijden aanroepen, of vooruitrijden zonder pad etc...) Het spreekt haast vanzelf dat het nuttig is dit tegen te gaan. Dit kan door de problemen te omzeilen door het tactisch kiezen van goals voor de on-line planner, maar het zou gebruikers vriendelijker zijn om hier een andere structurele oplossing voor te implementeren in de on-line planner zelf. Of de programmatuur van MARIE moet elders aangepast worden om deze crashes te voorkomen. Een gerelateerd probleem is bijvoorbeeld de oneindige loop die soms optreed in de "path planner".

5.4.2 Voorspellend plannen

Hiermee wordt bedoeld het plannen voordat er een verzoek hiertoe is binnengekomen. Het nut hiervan is dat wanneer er wel een verzoek tot plannen binnenkomt dit al gebeurd is, hetgeen snelheidswinst kan opleveren. Dit is een van de ideen die in het hoofdstuk planners (2) naar boven is gekomen. De reden dat dit nog niet is geïmplementeerd is dat de zoekruimte op dit moment zo klein is dat de te behalen snelheids winst verwaarloosbaar klein is. Zeker wanneer er een sneller workstation wordt gebruikt voor de taskplanner is er geen enkele reden om een sneller algoritme te hebben.

Hoofdstuk 6

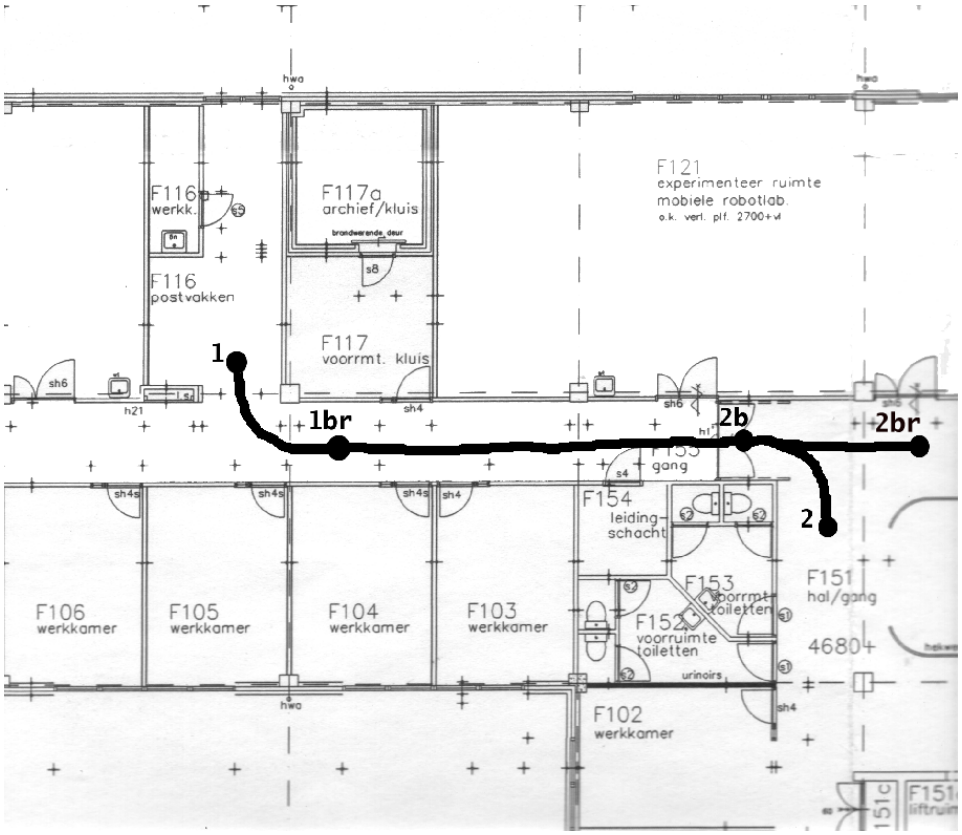
Tests

6.1 Doel

Het doel van de tests is het aantonen van de werking van de on-line planner. Dit houdt in dat de on-line planner met een zeer summier basisplan uiteindelijk een redelijk complex plan kan uitvoeren. Hiervoor moet dus zodra het basisplan ophoudt gekozen worden uit de verschillende alternatieven die mogelijk zijn, en wel op zo'n manier dat het einddoel bereikt wordt. In het ideale geval vindt de on-line planner alle relevante informatie in de juiste volgorde. Dan wordt dus op de efficiëntst mogelijke manier het einddoel bereikt. In het slechtste geval zal de planner in een lokaal minimum blijven hangen en nooit het einddoel bereiken. In feite wordt met deze tests dus gekeken of het simpele algoritme in staat is een (bijna) volledig plan te genereren. Dit is dus moeilijker dan wat volgens de hypothese uit hoofdstuk 4 mogelijk is.

6.2 De Test

In de test moet MARIE over de gang van de eerste verdieping van kruislaan 403 rijden. Het beginpunt is naast de printer (punt 1) en het eindpunt is één van de twee punten bij de vide (2 of 2br). Om bij een eindpunt aan te komen moet het volgende gebeuren, er moet een pad geplant worden van punt 1 naar punt 1br. Tussen punt 1br en punt 2b moet de wall-follower gebruikt worden. Als laatste moet van punt 2b een pad geplant en gereden worden naar één van de twee eind punten. Wanneer MARIE aan de test begint is ze op de juiste manier geinitialiseerd en weet ze haar positie, de on-line planner moet voor de rest zorgen.



Figuur 6.1: De test begint bij 1 en eindigt bij 2 of 2br

6.3 Basisplan

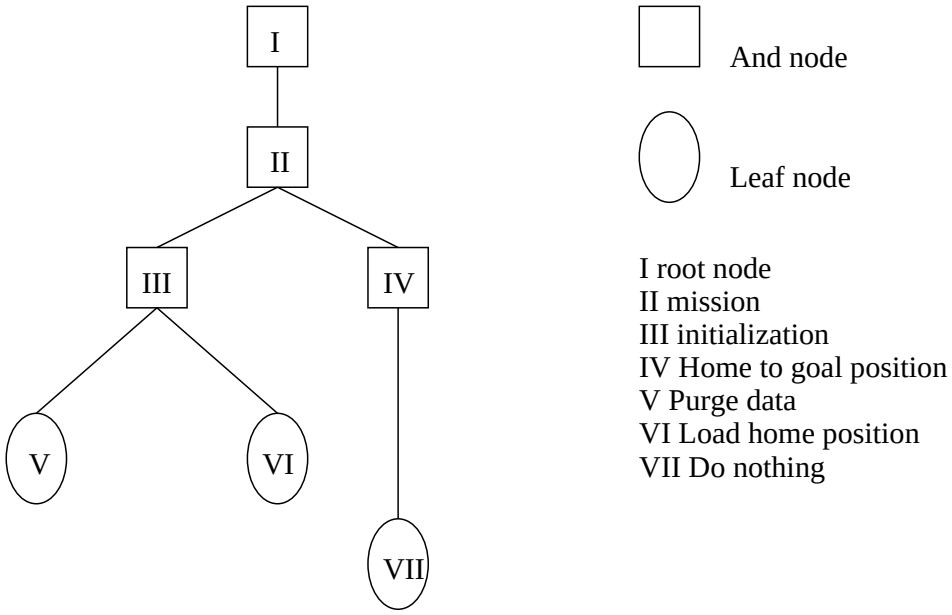
Het basisplan bestaat uit een root node, een initialisatie gedeelte, en een simpele structuur om makkelijk nieuwe nodes aan vast te kunnen maken. De root node bevat algemene informatie te weten: de final state waarin de robot moet verkeren om het plan met succes volbracht te hebben.

De during conditions, dus de flags die tijdens de uitvoering van het plan uit of aan moeten blijven. De initial condities, welke bepalen welke flags aan en uit staan aan het begin van de uitvoering van het plan.

Het initialisatie gedeelte verwijdert eventueel nog aanwezige paden en omgevings informatie die zijn overgebleven van het uitvoeren van een vorig plan. Verder wordt de begin positie geladen.

Als laatste is er een AND node met 1 kind van het type “do nothing”. Dit gebeurd omdat nieuwe plan informatie alleen vastgemaakt kan worden aan een AND of OR node met tenminste 1 kind. Dit is een beperking van de huidige implementatie, om ervoor te zorgen dat dit verder geen invloed kan hebben is de “do nothing” node er eentje die geen enkele actie uitvoert

en ook niets aan de huidige status veranderd.



Figuur 6.2: basis plan

6.4 Macro's

Voor deze test is een aantal macro's gemaakt die de on-line planner als bouwstenen gebruikt. Dit zijn dus losse plan onderdelen die dingen doen als het plannen van een pad of het rijden hiervan. Uiteindelijk zijn een aantal hiervan vrij groot geworden, dit komt voornamelijk doordat er slechts 32 flags beschikbaar zijn voor huidige "global status". De global status is het gedeelte van de huidige status dat tijdens de executie niet van betekenis kan veranderen. Om voldoende onderscheid te kunnen maken tussen alle plan onderdelen, voornamelijk met betrekking tot welke gezien de huidige status toegestaan zijn, kan je met 32 flags maar een beperkt aantal macro's maken. Je bent dus genoodzaakt redelijk algemene macro's te maken die voor complexere taken niet opgesplitst kunnen worden omdat dat de gedetailleerde aard hiervan niet aangegeven kan worden.

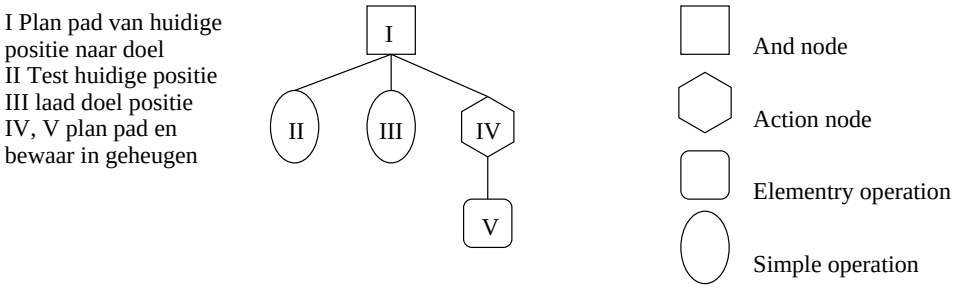
6.5 Simple operation macro's

Dit zijn de simpelste macro's, ze bestaan slechts uit een node te weten een simple operation. Het zijn dus eigenlijk geen macro's. Het gaat hier om "stop smooth" die zoals de naam al aangeeft de robot laat stoppen en de stopped flag aanzet. De andere die in deze test gebruikt wordt is "do

nothing" die zoals reeds eerder vermeld in het geheel niets doet. Het was ook mogelijk geweest om bijvoorbeeld de collision avoidance mode op deze manier te implementeren maar aangezien dit tenminste 4 flags (voor de verschillende modi) vereist is hiervan afgezien.

6.6 Pad planner macro's

Dit zijn macro's die de pad planner van MARIE aanroepen. Ze geven de huidige positie en de doel positie aan de pad planner welke vervolgens een pad naar de data manager (het geheugen) schrijft. Er zijn een aantal versies hiervan die verschillende eisen stellen aan de begin en eind status. Dus afhankelijk van waar MARIE zich bevindt zowel wat betreft positie als uitvoering van het plan zijn andere pad planner macro's van toepassing.



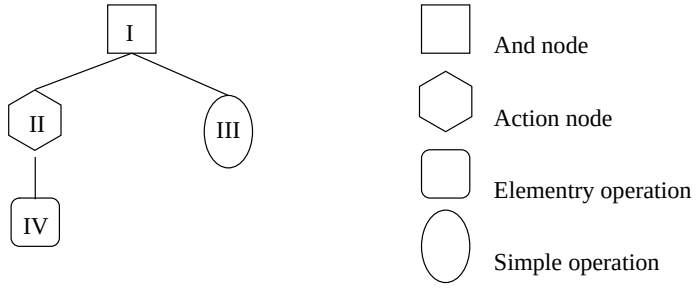
Figuur 6.3: plan onderdeel pad planner

6.7 Rij macro's

Van deze macro's zijn er twee types, de ene voert het pad uit dat door de pad planner berekend is, de ander is de "wall follower" welke de muur een opgegeven aantal meters volgt. Het eerste type voert dus het pad uit en controleert of het doel van dit pad bereikt is. Hiervan zijn er verschillende versies die verschillende dingen veranderen aan de huidige status, dit net als bij de pad planners afhankelijk van de status van MARIE.

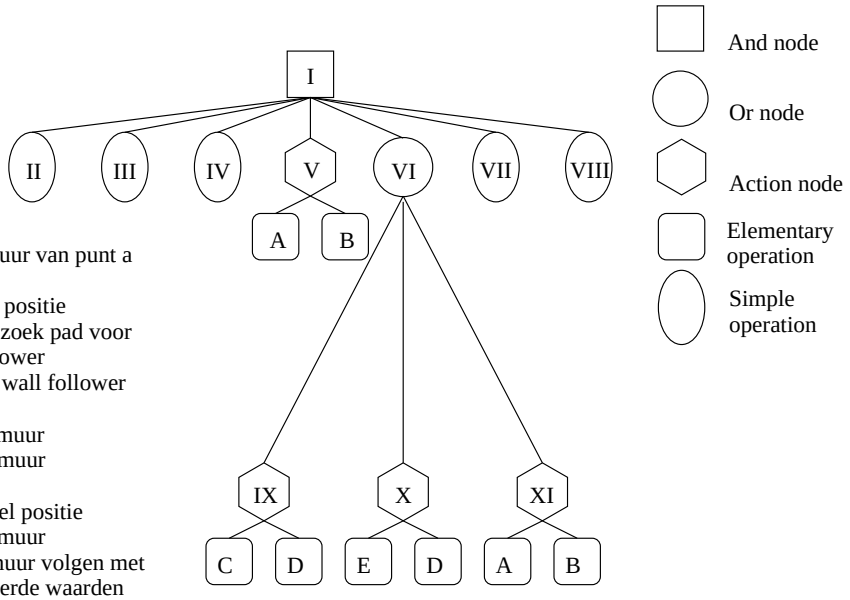
Van de "wall follower" is er maar een. Deze heeft het laden van het "wall follower path" dat de afstand bepaald die langs de muur moet worden afgelegd, in zijn macro geïntegreerd, dit vanwege het gebrek aan flags om deze op begrijpbare manier in een eigen macro te definieren. In deze macro wordt dus eerst het pad geladen, vervolgens wordt de muur gevolgd en als laatste wordt niet de positie gecontroleerd maar in plaats daarvan opnieuw gedefinieerd. Dit gebeurd omdat in de praktijk bleek dat na het volgen van de muur over een grote afstand (meer dan 10 meter) de fout in de positie bepaling te groot was geworden. De afgelegde voorwaartse afstand klopte nog wel, de zijdelingse afstand had een fout van meer dan een meter.

I Rij pad
II, IV Rij naar doel
III Test doel positie



Figuur 6.4: plan onderdeel rij het pad

Aangezien de robot zich op gelijke afstand langs een rechte muur bewoog kon deze zijdelingse fout echter gecorrigeerd worden door de werkelijke positie als nieuwe huidige positie in te laden.

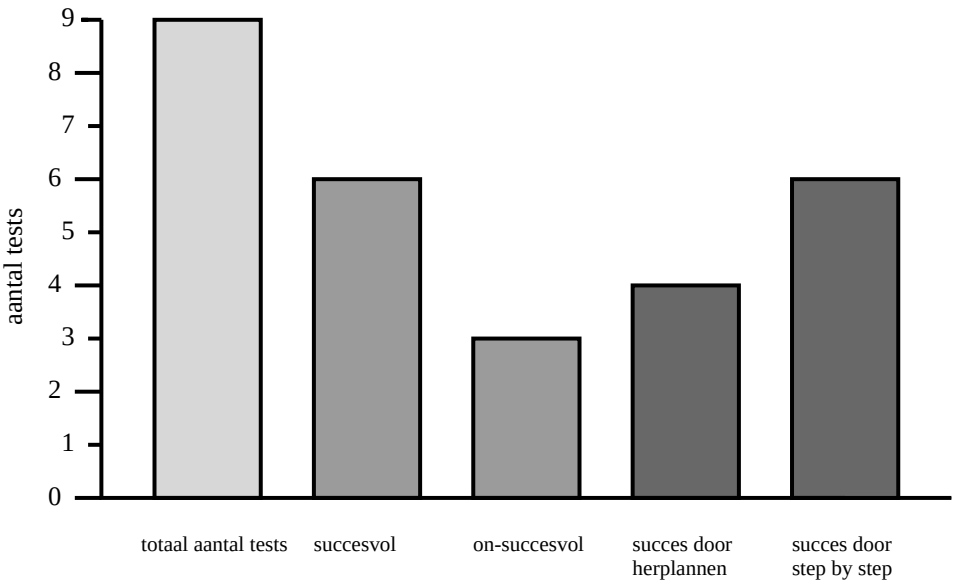


I volg de muur van punt a naar punt b
II laad doel positie
III laad het zoek pad voor de wall follower
IV laad het wall follower pad
V vindt de muur
VI volg de muur
VII stop
VIII test doel positie
IX volg de muur
X blijf de muur volgen met geextrapoleerde waarden
XI vindt de muur opnieuw
A rij het zoek pad
B vindt de muur
C volg het wall follower pad
D volg de muur
E volg geextrapoleerd pad

Figuur 6.5: plan onderdeel volg de muur

Hoofdstuk 7

Resultaten



Figuur 7.1: test resultaten

Zoals in de grafiek al is te zien zijn er in totaal 9 tests gedaan met de uiteindelijke versie van de on-line planner, met het uiteindelijke testplan. Om dit punt te bereiken zijn er meer dan 150 gelogde tests gedaan, en nog vele niet gelogde. Met gelogd wordt bedoelt dat alle output van de verschillende modules te weten de taskplanner, de action dispatcher en de wall-sensor(die extern geïmplementeerd is) in een text-file is opgeslagen. Als eerste zullen er mogelijke verklaringen worden gegeven voor de onsuccesvolle tests, daarna wordt uitgelegd hoe de succesvolle tests tot een goed einde kwamen.

7.1 Mogelijke oorzaken on-succesvolle tests

Alle onsuccesvolle tests faalden al vrij in het begin van de test bij het rijden van de eerste bocht. Deze bocht luistert vrij nauw aangezien hij behoorlijk scherp is, hij ligt zeer dicht bij de maximale stuur uitwijking van MARIE. Daardoor is de plaatsing van MARIE op de juiste begin positie zeer kritisch, en wordt er ook behoorlijk wat gevergd van de padplanner. Hierdoor kunnen een aantal onnauwkeurige sensor waarden de robot laten uitwijken voor een muur die dichterbij lijkt te zijn dan hij werkelijk is. Een uitwijking maakt het in een keer nemen van de bocht vrijwel onmogelijk. Dit effect zal eerder optreden als de begin positie niet precies goed was. Nou is het niet in een keer kunnen nemen van een bocht op zich geen terminaal probleem, maar voor dat MARIE hier achter is komt ze in de buurt van een metalen deurpost die zeer vreemde sensorwaarden kan opleveren door zijn hoge reflectiviteit. Het gevolg hiervan is dat MARIE of tegen de deurpost blijft staan of de muur in het geheel niet ziet waardoor ze er tegenaan rijdt.

7.2 Redenen succes

Zoals te zien in de grafiek waren er 6 tests succesvol hier voor zijn ook nog twee specifieke redenen te zien te weten herplannen en "step by step". In het volgende gedeelte wordt uitgelegd hoe deze werken en waarom ze interessant zijn.

Met herplannen wordt bedoeld dat een macro (planonderdeel) gefaald heeft waarna de on-line planner het plan aanpast met als gevolg dat het plan uiteindelijk zijn doel toch kan bereiken. In alle 4 de gevallen waar het hier om gaat was het de pad planner die in eerste instantie niet in staat was een pad te vinden voor het laatste gedeelte van de missie. Echter nadat de on-line planner 1 of meerdere macro's had toegevoegd(waarvan de laatste weer een pad-planner was) lukte het alsnog een pad te vinden.

In het geval van "step by step ging" het om het rijden van het pad zelf. Bij alle succesvolle tests faalde in het laatste gedeelte regelmatig het rijden van het door de pad-planner bedachte pad. De on-line planner bleef in deze gevallen echter steeds de macro's leveren die probeerde het rijden van het pad te voltooien, met als gevolg dat het doel alsnog bereikt werd. Dit kwam omdat er iedere keer al een gedeelte van het pad was gereden op het moment dat het plan onderdeel faalde.

7.3 Hypotheses

In hoofdstuk 4 werden de volgende hypotheses gesteld.

- Het basisplan is geschikt om in (gedeeltelijke) plannen gaten op te vullen.

- Om redenen van snelheid en complexiteit is voor het vinden van complete plannen een geavanceerder algoritme dan random kiezen nodig.

Na het doen van de tests is gebleken dat het enigszins uitgebreide random kiezen algoritme ook succesvol was in het genereren van volledige plannen. Het algoritme is iets geavanceerder dan puur random kiezen maar het is zeker nog niet van het niveau van een UCPOP achtig algoritme. In de praktijk is de efficiëntie van het zoek algoritme niet de meest beperkende factor voor het beter presteren van MARIE. Waarom dit zo is, en wat wel de meest beperkende factor is, komt aan bod in het volgende hoofdstuk.

Hoofdstuk 8

Toekomstige verbeteringen

8.1 Inleiding

In 5.4 werden reeds enkele verbeteringen voorgesteld, te weten het uitsluiten van crashende plannen en voorspellend plannen. Deze waren echter opgesteld voor het uitvoeren van de tests. Uit die tests is gebleken dat de in 5.4 genoemde verbeteringen niet erg relevant zijn. In de sectie snelheid zal worden uitgelegd waarom een snelheids verbetering, waar voorspellend plannen onder valt, geen grote prioriteit heeft. Ook crashende plannen bleken geen probleem te zijn in de tests, ze kwamen namelijk niet voor. In de rest van dit hoofdstuk zullen dus de verbeteringen aan bod komen die nodig bleken naar aanleiding van de tests.

8.2 Snelheid

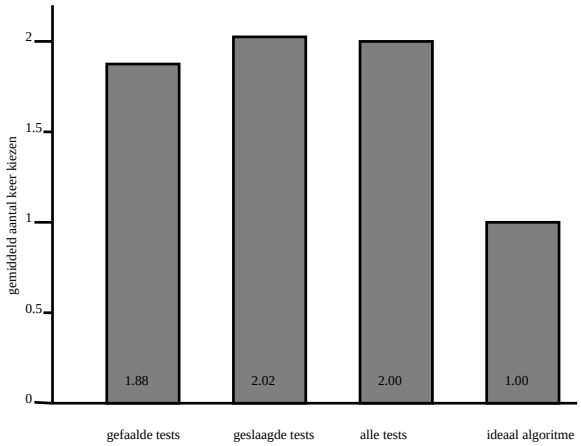
Op het gebied van snelheid kan een aantal dingen verbeterd worden. Sommige van deze verbeteringen zullen meer effect hebben dan andere. Als eerste de mogelijke verbeteringen die waarschijnlijk weinig uithalen.

Een mogelijke verbetering is het ervoor zorgen dat op het moment dat er om nieuwe plan informatie gevraagd wordt door MARIE deze al klaar staat. Dit houdt dus in dat er constant gezocht wordt naar alternatieven als hier nog niet om gevraagd is. In de praktijk is gebleken dat het feitelijke zoeken van de informatie een verwaarloosbaar kleine hoeveelheid tijd vraagt. Deze verbetering zal dus in de praktijk nauwelijks merkbaar zijn. Dit zou nog kunnen veranderen als de heuristiek veel ingewikkelder gaat worden en dus wel veel tijd in beslag neemt.

De on-line planner zou voor snelheidswinst op MARIE zelf kunnen draaien, waarmee vertragingen door het netwerk weggenomen worden. Er hoeven echter geen grote hoeveelheden data verzonden te worden en netwerk snelheid is ook geen significante “bottleneck”.

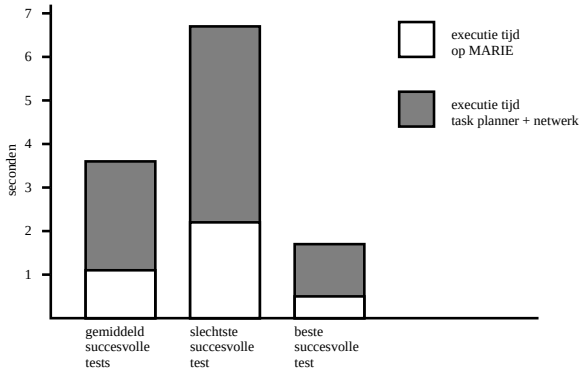
Een verbeterde heuristiek zou wel degelijk een snelheidswinst kunnen

opleveren, op het moment wordt een groot aantal keren gekozen voor macro's die MARIE niet dichter bij haar uiteindelijke doel brengen en dit neemt tijd in beslag. Een volledige implementatie van de heuristiek uit UCPOP [18] zou een aardige snelheidswinst kunnen opleveren.



Figuur 8.1: gemiddeld aantal keren kiezen voordat een nuttig plan onderdeel wordt gevonden

Figuur 8.1 laat zien dat deze verbetering echter nooit meer dan een verdubbeling van de efficiëntie kan opleveren. Om te kijken wat dit in de praktijk betekent volgt nu een benadering van de snelheids winst voor een ideaal algoritme. Dit ideale algoritme kiest altijd een nuttig planonderdeel uit. Voor deze benadering wordt gekeken hoeveel tijd er in de succesvolle tests besteed is aan niet nutigge plan onderdelen. Er wordt niet gekeken naar de winst die een ideaal algoritme zou kunnen hebben in zoektijd aangezien dit buiten MARIE wordt gedaan en daardoor net zo goed verbeterd kan worden door een snellere computer te gebruiken.



Figuur 8.2: tijd besteed aan niet nuttige plan onderdelen

Zoals in figuur 8.2 te zien is wordt er gemiddeld nog geen 4 seconden per test verspeelt door niet nuttige plan onderdelen. Een test duurt tussen de anderhalf en twee minuten, hierop is het verschil wel merkbaar er zijn echter factoren die een grotere vertraging op kunnen leveren. Wanneer er met "step by step" voorbij een moeilijk punt in het plan gekomen moet worden kan dit 10 tot 20 seconden toevoegen. Een ideaal algoritme kan dus voor snelheids winst zorgen, het is niet noodzakelijk hetgene dat de meeste snelheids winst oplevert en is ook niet de verbetering die het meeste nodig is voor een verbetering in het presteren van MARIE.

8.3 Positie bepaling

In de huidige versie van MARIE wordt positie bepaling gedaan doormiddel van "dead reckoning" dat wil zeggen dat de positie beredeneerd wordt aan de hand van de afstand die de vier wielen hebben afgelegd. Deze methode heeft als nadeel dat hoe langer er gereden wordt des te groter de fout wordt. De enige remedie hiervoor op dit moment is een herijking door naar een van te voren gedefinieerde positie te rijden, bijvoorbeeld een hoek waarvan de werkelijke positie bekend is. Het zou echter handiger zijn om dit continu te doen aan de hand van de waardes die uit de wall sensor (module die muren herkent in de sensor waarden) komen en het van te voren bekende wereld model. Dit is vooral in gevallen waar je de robot langere afstanden wil laten afleggen in een gebouw zoals in dit onderzoek. Er zijn in die situatie namelijk maar weinig hoeken waar makkelijk herijkt kan worden.

8.4 Overzicht

Om een beter overzicht te krijgen zou het nuttig zijn als er een centrale applicatie is die op een overzichtelijke manier weergeeft wat de interne status van MARIE is en ook waar de taskplanner tegelijkertijd mee bezig is. Dit is op het moment iets dat uit verschillende text logs gehaald moet worden waarbij de status bijvoorbeeld zonder enige uitleg als een reeks bits wordt weergegeven. Zeker voor iemand die nog niet bekend is met MARIE is dit zeer onoverzichtelijk en is het dus zeer lastig hier nuttige informatie uit te destilleren.

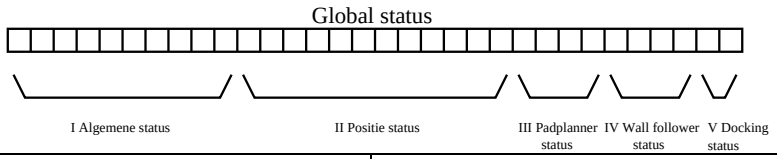
8.5 Complexiteit

Zoals reeds eerder gemeld vormen de 32 flags die de (global) status van MARIE aangeven een behoorlijke beperking op het aantal verschillende macro's dat beschikbaar is voor de on-line planner. Voor een groter aantal macro's van grotere verscheidenheid moet dit aantal flags vergroot worden. De global status is immers een zeer essentieel onderdeel van het wereldbeeld van MARIE. Alle informatie die een high-level planner nodig heeft moet uit deze 32 bits gehaald worden. Voor een systeem dat complexere plannen kan uitvoeren is meer informatie nodig over de status van MARIE in de wereld. Een groter aantal flags(bits) maakt dit beter mogelijk. Een aantal dingen moet hierbij echter wel in de gaten gehouden worden. De 32 flags zoals ze nu bestaan zijn zeer overzichtelijk, als dit zonder verdere aanpassing van de structuur vergroot wordt kan dit overzicht verloren gaan, er zal dus iets gevonden moeten worden om het overzicht te behouden, bijvoorbeeld een vorm van hierarchie. De global status zou dan onderverdeeld kunnen worden in in verschillende statussen die betrekking hebben op verschillende onderdelen van de robot architectuur.

Ook moet de beschrijving van de global status niet complexer worden dan nodig is, op het moment laat de global status nog niet de gewenste hoeveelheid detail in plannen toe, dit moet niet overslaan naar plannen die niet meer te begrijpen zijn door een te grote gefragmenteerdheid. Hiermee wordt bedoelt dat de losse planonderdelen of macro's die de on-line planner op het moment gebruikt nog te groot zijn, maar dat wanneer ze in te kleine stukjes worden onderverdeeld het overzicht zoek kan raken.

Voor het plan zoals het in de tests gebruikt is zijn er tenminste 4 extra flags nodig om de collision avoidance aan te geven, verder moet er tenminste een zijn om vooraf gedefinieerde pad informatie aan te geven. Dat levert al 5 bits op, wil je de positie van de robot tijdens het uitvoeren van het plan overzichtelijker weergeven, dan ben je er zo nog 5 kwijt. Voor het huidige plan zijn dus ongeveer 10 extra bits nodig, en het huidige plan is niet bijzonder gecompliceerd. Vooral als je langs meerdere punten wil rijden

ben je veel bits kwijt aan positie bepaling. Dus voor plannen met langere af te leggen routes erin groeit het aantal bits lineair. Verder moeten ook extra bits gereserveerd worden voor functies die door dit plan niet gebruikt worden zoals bijvoorbeeld inparkeren. Voor een volgende stap in de ontwikkeling van MARIE lijkt een totaal van 64 bits onderverdeeld in duidelijke categorieën een verstandige keuze.



I Algemene status:	
!!systems go	==global 0
!!completed	==global 1
!!not completed	==global 2
!!stopped	==global 3
!!moving	==global 4
!!US ready	==global 20
!!VI ready	==global 21
!!pose known	==global 25
!!pose lost	==global 28
!!severe global	==global 30

II Positie status:	
!!Goal 1	==global 5
!!Goal 1b	==global 6
!!Goal 1br	==global 15
!!Goal 2	==global 16
!!Goal 2b	==global 17
!!Goal 2br	==global 18
!!Goal not reached	==global 19
!!Goal 3	==global 24
!!Goal 3b	==global 26
!!Goal 3br	==global 27
!!Between b	==global 29
!!Drive away	==global 31

III Pad planner status:	
!!TC path	==global 7
!!no TC path	==global 8
!!PP data	==global 13
!!no PP data	==global 14

IV Wall follower status:	
!!WF path	==global 9
!!no WF path	==global 10
!!wall found	==global 22
!!wall lost	==global 23

V Docking status:	
!!docked	==global 11
!!not docked	==global 12

Figuur 8.3: Huidige Globalstatus

VI Collision avoidance status:	VIII Extra docking status:
CA mode 0	Docking spot found
CA mode 1	Docking spot not found
CA mode 2	Docking path
CA mode 3	no Docking path
VII Positie bepaling status:	IX
Drive home	Pre defined path loaded
Drive to br	
Goal 4	
Goal 4b	
Goal 4br	

Figuur 8.4: Uitbreiding Globalstatus

Hoofdstuk 9

Conclusie

In de conclusie zal worden gekeken naar de volgende onderwerpen: hoe vergelijkt het resultaat met andere on-line planners, wat kan er in de toekomst gedaan worden en zijn de doelen die aan begin zijn gesteld berijkt.

In vergelijking met andere on-line theoretische planners, zoals UCPOP en Graphplan, presteert deze implementatie nog niet erg goed. De plannen die eruit komen zijn niet optimaal, dit is inherent aan het random kiezen de voornaamste heuristiek die gebruikt is. In vergelijking met de echte robots uit het onderzoek van Peter Nordin [13] kan MARIE echter veel meer. De robots uit [13] kunnen namelijk alleen lage niveau taken als obstakels vermijden, terwijl MARIE plannen op hoog niveau kan uitvoeren.

De efficiëntie die de on-line planner uit dit onderzoek mist is echter in de praktijk van minder belang. In hoofdstuk 8 is gebleken dat de snelheids winst van een ideaal algoritme gemiddeld niet meer zal zijn dan een krappe 4 seconden in de test zoals beschreven in hoofdstuk 6. Hiermee is het niet de grootste vertragende factor van MARIE op dit moment.

Bij plannen op hoog niveau is snelheid ook niet van het grootste belang, voor tijd kritische problemen die het plan moeten aanpassen zijn al mechanismen zoals de collision avoidance. Het implementeren van een betere heuristiek wordt vooral interessant als er meer keuze mogelijkheden zijn voor de on-line planner. De huidige 32 global status flags die de toestand van MARIE weergeven beperken de keuze mogelijkheden van de planner. Hierdoor is er geen groot verschil tussen heuristieken merkbaar is. Als de status flags echter worden uitgebreid en dus een veel grotere verscheidenheid aan herplan informatie mogelijk maken dan worden krachtiger heuristieken veel nuttiger.

Uitbreidingen voor de toekomst in volgorde van belangrijkheid:

- Uitbreiden van de global status flags zodat problemen beter in verschillende planonderdelen kunnen worden opgesplitst en ook om te zorgen dat planonderdelen uit het verleden hergebruikt kunnen worden

- Betere positie bepaling bijvoorbeeld door koppeling van het a-priori wereld model aan de sensorwaarden
- Visueel overzicht van de interne status van MARIE
- een efficiënter algoritme voor het aankunnen van grotere zoekruimtes

De eerste twee uitbreidingen zijn het belangrijkst, hierdoor zal MARIE complexere plannen aan kunnen dan nu het geval is.

Bij het maken van het ontwerp zoals beschreven in hoofdstuk 4 zijn de volgende hypotheses gesteld:

- Het basisplan is geschikt om in (gedeeltelijke) plannen gaten op te vullen.
- Om redenen van snelheid en complexiteit is voor het vinden van complete plannen een geavanceerder algoritme dan random kiezen nodig.

Uit de tests kwam naar voren dat deze hypotheses aan de conservatieve kant waren, de uiteidelijke implementatie die voornamelijk gebruikt maakt van random kiezen is wel degelijk in staat om complete plannen te genereren. Wat dit betreft is de on-line planner dus beter uitgevallen dan de oorspronkelijke verwachtingen.

De belangrijkste doelen die aan het begin gesteld zijn:

- Grotere betrouwbaarheid
- Grotere robuustheid

In de tests is gebleken dat bij iedere succesvolle test “step by step” nodig was om het einddoel te bereiken. Met andere woorden, als je alle nuttige planonderdelen van de test in de juiste volgorde achterelkaar zet, dus met dezelfde plan informatie van tevoren een plan maakt, dan zal dit (bijna) nooit tot een succesvol resultaat leiden. Voor het bereiken van een succesvol resultaat op deze test met een van tevoren bedacht plan moet in dit plan een aanzienlijke hoeveelheid extra werk gestoken worden. Dit laat duidelijk zien dat de on-line planner een grotere betrouwbaarheid tot gevolg heeft.

De tests laten ook zien dat een online planner gaten in een plan kan opvullen. Immers in de test ontbreekt in het basis plan bijna alle informatie en toch komt MARIE een 66% van de tests bij haar einddoel. Als er in een plan kleinere gaten zitten zal de on-line planner deze logischer wijs ook op kunnen vullen. Dit leid tot een grotere robuustheid voor incomplete plannen.

Wat dit onderzoek heeft laten zien is dat een on-line planner op een echte robot toe te passen is waarbij deze zorgt voor grotere betrouwbaarheid en robuustheid.

Bibliografie

[1] Michael Beetz and Drew McDermott. Improving Robot Plans During Their Execution. Proc. of 2nd Int Conf on AI Planning Systems, Chicago, IL, June 1994, AAAI press, Boston, MA.

[2] Avrim L. Blum, Merrick L. Furst. Fast Planning Through Graph Analysis. Artificial Intelligence, 90:281-300, 1997.

[3] Avrim L. Blum and John C Langford. Probabilistic Planning in the Graphplan Framework.

[4] G.A. Den Boer. A control architecture for the MARIE autonomous mobile robot. 1995

[5] Howie Choset, Joel Burdick. Sensor Based Planning, Part I: The Generalized Voronoi Graph. 1995 IEEE International Conference on Robotics and Automation.

[6] Karen Zita Haigh, Manuela M. Veloso. High-Level Planning and Low-Level Execution: Towards a Complete Robotic Agent. Proceedings of the First International Conference on Autonomous Agents, Feb. 1997, Menlo Park, CA.

[7] Nikolaos A. Massios and Frans Voorbraak. Planning Strategies for Decision-Theoretic Robotic Surveillance. Tenth Netherlands/Belgium Conference on Artificial Intelligence (NAIC'98) http://carol.wins.uva.nl/~massios/surveillance/publications/NAIC98.ps

[8] Lisa A. Meeden, Deepak Kumar. Trends In Evolutionary Robotics. Soft Computing for Intelligent Robotic Systems, edited by L.C. Jain and T. Fukuda, Physica-Verlag, New York, NY, 215-233, 1998.

[9] Jing Xiao, Zbigniew Michalewicz, Lixin Zhang, Krzysztof Trojanowski. Adaptive Evolutionary Planner/Navigator for Mobile Robots. IEEE transactions on evolutionary computation, vol. 1, no 1, april 1997.

[10] Jing Xiao, Zbigniew Michalewicz, and lixin Zhang. Evolutionary Planner/Navigator: Operator Performance and Self-Tuning. IEEE Int. Conf. Evolutionary Computation, Nagoya, Japan, May 1996, pp. 366-371.

[11] Krzysztof Trojanowski, Zbigniew Michalewicz, Jing Xiao, Adding Memory to the Evolutionary Planner/Navigator. 1997 IEEE International Conference on Evolutionary Computation, pp. 483-487, Indianapolis, April 1997.

[12] Jing Xiao, Lixin Zhang, and Zbigniew Michalewicz. On Topological Diversity and Multiple Path Planning. in the 2nd International Conf. on Computational Intelligence and Neuralscience, pp. 10-13, Research Triangle Park, NC, March 1997.

[13] Peter Nordin, Wolfgang Banzhaf. An On-Line Method to Evolve Behavior and to Control a Miniature Robot in Real Time with Genetic Programming. 20-1-1997

[14] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise Strategies for Improving Local Search. proceedings of AAAI-94, Seattle, WA, July 1994

[15] Yi Cheng Huang, Bart Selman, Henry Kautz. Control Knowledge in Planning: Benefits and Tradeoffs. AAAI-99, dec 1999.

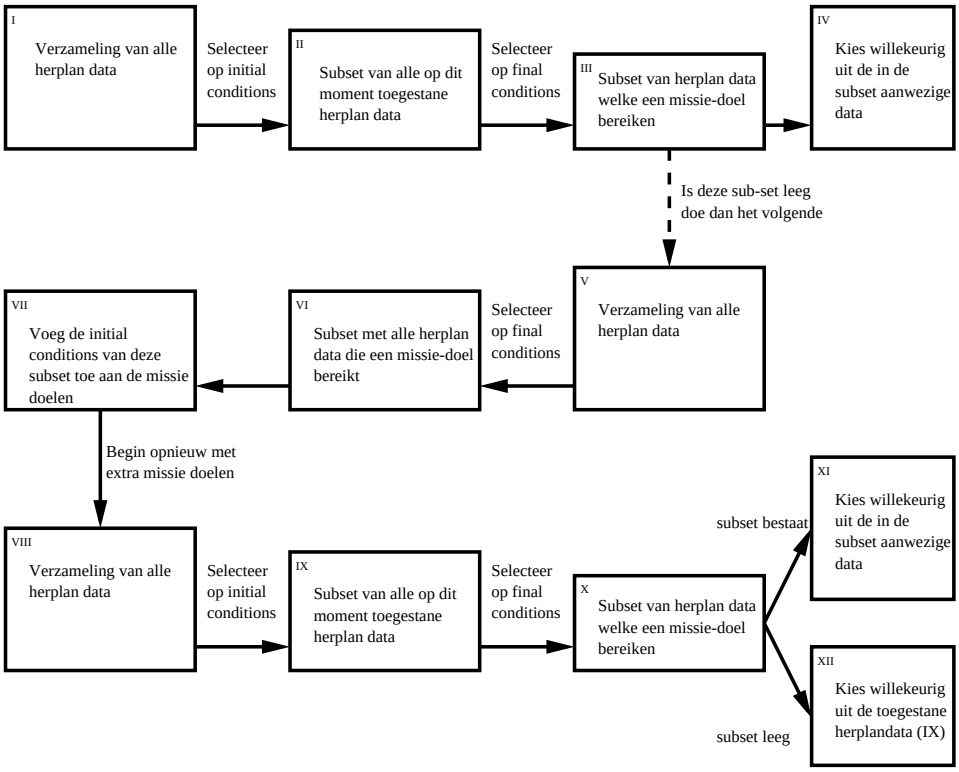
[16] Henry Kautz, Bart Selman. The Role of Domain-Specific Knowledge in the Planning as Satisfiability Framework. Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98), Pittsburgh, PA, 1998.

[17] Reid Simmons and David Apfelbaum. A Task Description Language for Robot Control. Proceedings of the Conference on Intelligent Robots and Systems (IROS), Victoria Canada, 1998.

[18] Daniel S. Weld. An Introduction to Least Commitment Planning. AI Magazine Summer/Fall 1994.

Bijlage A

algoritme



Figuur A.1: Algoritme on-line planner

Hier volgt de code van het algoritme dat gebruikt wordt om herplan informatie uit te kiezen. In de code zal worden aangegeven welke gedeeltes overeenkomen met de stappen uit het bovenstaande figuur.

```
PUBLIC int replan(sd, status, re_plan_info, globalgoals, localgoals)
     int sd;
```

```
    CONDITION status;
    AD_TREE_NODE_PTR *re_plan_info;
    unsigned long *globalgoals[64];
    unsigned long *localgoals[64];

{
  int i,j,k,l,size,random;
  int *choose, *prefer;

  struct timeval tp;
  void *vp;
  /*
  printf("\n now replanning\n");
  print_bits(status[0],"current global status");
  print_bits(status[1],"current local status");
  */
```

I Verzameling van alle herplan data.

```
  size = 0;
  while (re_plan_info[size] != NULL)
    {
      size++;
    }
    choose = (int*) malloc((size+1)*sizeof(int));
  if( choose == NULL)
    {
      printf("error allocating memmory for array choose\n");
      exit(1);
    }

    prefer = (int*) malloc((size+1)*sizeof(int));
  if( prefer == NULL)
    {
      printf("error allocating memmory for array prefer\n");
      exit(1);
    }

  j=0;
  k=0;
  for(i=0;i<size;i++)
    {
      /*
      printf("re_plan_info number: %d \n",i);
```

```
      print_bits(re_plan_info[i]->d.cond_initial[0],"initial global status");
      print_bits(re_plan_info[i]->d.cond_initial[1],"initial local status");
      print_bits(re_plan_info[i]->d.cond_final[0],"final global status");
      print_bits(re_plan_info[i]->d.cond_final[1],"final local status");
      */
      if(implies(re_plan_info[i]->d.cond_initial[0], status[0]) &&
 implies(re_plan_info[i]->d.cond_initial[1], status[1]))
{
```

II Subset van alle op dit moment toegestane herplan data, selectie op initial condities

```
  choose[j] = i;

  if(satisfies(re_plan_info[i]->d.cond_final[0], globalgoals,
       status[0],0))
    {
```

III Subset van herplan data welke een missie doel bereikt, selectie op final condities

```
      prefer[k]= i;
      k++;
      /*
      printf("\n !!! \n prefer bestaat!!! \n !!! \n");
      */
    }
  j++;
}


    }
  /*
  for(i=0;i<32;i++)
    {
      if(globalgoals[i]!=NULL) print_bits(globalgoals[i],"global goal");
      if(localgoals[i]!=NULL) print_bits(localgoals[i],"local goal");
    }
  */
  if(k == 0)
    {
```

V De subset die missie doelen bereikt is leeg

```
      /*
      printf("adding goals \n");
      */
      add_goals(re_plan_info,globalgoals, localgoals, status[0], 0);
```

VIII Begin opnieuw met extra missie doelen (stappen VI & VII bevinden zich in de functie add_goals)

```
      j=0;
      k=0;
      for(i=0;i<size;i++)
{
  if(implies(re_plan_info[i]->d.cond_initial[0], status[0]) &&
     implies(re_plan_info[i]->d.cond_initial[1], status[1]))
    {
```

IX Subset van alle op dit moment toegestane herplan data, selectie op initial condities

```
      choose[j] = i;

      if(satisfies(re_plan_info[i]->d.cond_final[0], globalgoals,
   status[0],0))
{
```

X Subset van herplan data welke een missie doel bereikt, selectie op final condities.

```
  prefer[k]= i;
  k++;
  /*
  printf("\n !!! \n prefer bestaat!!! \n !!! \n");
  */
}
      j++;
    }
}

      if(k == 0)
{
```

XII Kies willekeurig uit de toegestane herplan data (IX).

```
  /*
  printf("prefer is leeg \n");
  */
  gettimeofday(&tp, vp);
  srand((int)(tp.tv_usec));
  random = rand() % j;
  i = choose[random];
  /*
```

```
  printf("random gekozen is : %d random seed: %d \n",i,(int)(tp.tv_usec));
  */
}
    }
  else
    {
```

IV & XI Kies willekeurig uit de in de subset (die missie doelen bereikt) aanwezige data.

```
      gettimeofday(&tp, vp);
      srand((int)(tp.tv_usec));
      random = rand() % k;
      i = prefer[random];
      /*
      printf("random gekozen (uit prefer) is : %d \n",i);
      */
      }
  ad_tree2sock(re_plan_info[i],sd);
  free(choose);
  free(prefer);
  return(0);
}

PUBLIC int satisfies(node, goals, current, offset)
unsigned long node;
unsigned long current;
unsigned long goals[64];
int offset;
{
  int i;
  int ret = 0;

  for(i=(0+offset);i<(32+offset);i++)
    {
      if((implies(goals[i], node)) && (goals[i] != NULL)
 && !(implies(goals[i], current))) ret = 1;
    }
  return(ret);
}

  /*
     Als er geen nodes in prefer zitten en er dus geen nodes zijn die direct
     aan een final conditie van de root node (ook wel een goal genoemd) kunnen
     voldoen, dan worden er nodes gezocht die wel aan de goals voldoen met
```

```
     hun eigen final condities maar die met de huidige status vanwege hun
     initial condities niet kunnen worden uitgevoerd. De initial condities
     van deze nodes worden dan toegevoegd aan de goals.
     Hierbij moet erop gelet worden dat er niet goals worden toegevoegd
     voor nodes die dezelfde final condities tot gevolg hebben,
     dit gebeurd door middel van check, die is samengesteld uit de laatste
     32 elementen van het goals array, welke de goals bevat waarvoor al
     een node gevonden is en waarvoor de (sub) goals zijn toegevoegd.
     de eerste 32 elementen van goals bevatten de werkelijke goals.
  */

PUBLIC int add_goals(re_plan_info,globalgoals, localgoals, current, gl)
AD_TREE_NODE_PTR *re_plan_info;
unsigned long current;
unsigned long globalgoals[64];
unsigned long localgoals[64];
int gl;
{
  int i,j,k,l,size;
  CONDITION check;
  j=0;
  k=0;
  check[0]=0;
  check[1]=0;
  size = 0;
  while (re_plan_info[size] != NULL)
    {
      size++;
    }

  for(i=0;i<size;i++)
    {

      for(l=32;l<65;l++)
{
  check[0] = check[0] | globalgoals[l];
  check[1] = check[1] | localgoals[l];
}
      /*
#ifdef MARIE_DEBUG
 printf("add_goals: re_plan_info number: %d \n",i);
 print_bits(re_plan_info[i]->d.cond_initial[0],"add_goals: initial global status")
 print_bits(re_plan_info[i]->d.cond_initial[1],"add_goals: initial local status");
 print_bits(re_plan_info[i]->d.cond_final[0],"add_goals: final global status");
```

```
 print_bits(re_plan_info[i]->d.cond_final[1],"add_goals: final local status");
#endif
      */
```

VI Subset van alle herplan data die een missie doel bereikt, geselecteerd op final condities

```
       if(satisfies(re_plan_info[i]->d.cond_final[0],globalgoals, current,0) &&
  !implies(re_plan_info[i]->d.cond_final[0], check[0]))
 {
```

VII Voeg de initial condities van de subset uit VI toe aan de missie doelen.

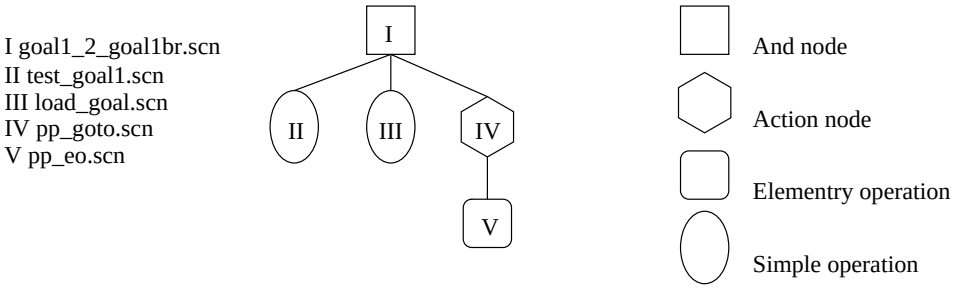
```
       for(j=0;j<32;j++)
 {
   if(globalgoals[j] == NULL)
     {
       globalgoals[j] = (re_plan_info[i]->d.cond_initial[0])
        & ((long) 1 << j);

     }
   if(localgoals[j] == NULL)
     {
       localgoals[j] = (re_plan_info[i]->d.cond_initial[1])
        & ((long) 1 << j);

     }
   if(implies(globalgoals[j],re_plan_info[i]->d.cond_final[0]))
     {
       globalgoals[j+32] = (re_plan_info[i]->d.cond_final[gl])
        & ((long) 1 << j);
     }
 }
       k=1;
 }
    }
return(k);
}
```

Bijlage B

Plan informatie Pad Planner



Figuur B.1: plan informatie pad planner

Hier volgen de daadwerkelijke scripts die de plan informatie voor de pad plan macro bevatten, dit als voorbeeld van de werkelijke vorm van macro's.

B.1 goal1_2_goal1br.scn

```
and
Plan goal path
32121
end### edit_global_flags
end### edit_local_flags
??-success
end### copy_flags_down
??-success
end### copy_flags_back
end### turn on parent flags
end### turn off parent flags
??+no TC path
??+stopped
??+Goal 1
```

```
??-Between b
??+Drive away
end### global initial
end###  local initial
??-no TC path
end### global during
end###  local during
??+TC path
end### global final
end###  local final
1 ### max. number of retries (dummy for and-node)
scenario test_goal1.scn
scenario load_goal.scn
scenario pp_goto.scn
end ### back to parent-node
read goal1_2_goal1br.data
```

B.2 test_goal1.scn

```
simple### /
Test if position near specified### create_simple_op/
??SO TST dist
end### edit_global_flags/for simple op
end### edit_local_flags/for simple op
??-finished
end### map_flags_down/
??-finished
end### map_flags_up/
??Goal 1
??+SO success
??+SO opt 1
end### get_some_flags/ On<-on 'Goal 1'<-
??+SO not found
??+SO failed
??+SO opt 2
??+SO opt 3
??+SO opt 4
end### get_some_flags/ On<-off 'Goal 1'<-
??pose known
??+SO success
??+SO opt 1
end### get_some_flags/ On<-on 'pose known'<-
```

```
??+SO not found
??+SO failed
??+SO opt 2
??+SO opt 3
??+SO opt 4
end### get_some_flags/ On<-off 'pose known'<-
end### turn on/map_flags_up
??Goal not reached
??+SO success
??+SO opt 1
end### get_some_flags/ Off<-on 'Goal not reached'<-
??+SO not found
??+SO failed
??+SO opt 2
??+SO opt 3
??+SO opt 4
end### get_some_flags/ Off<-off 'Goal noet reached'<-
??Goal 1b
??+SO success
??+SO opt 1
end### get_some_flags/ Off<-on 'Goal 1b'<-
??+SO not found
??+SO failed
??+SO opt 2
??+SO opt 3
??+SO opt 4
end### get_some_flags/ Off<-off 'Goal 1b'<-
??Goal 1br
??+SO success
??+SO opt 1
end### get_some_flags/ Off<-on 'Goal 1br'<-
??+SO not found
??+SO failed
??+SO opt 2
??+SO opt 3
??+SO opt 4
end### get_some_flags/ Off<-off 'Goal 1br'<-
end### turn off/map_flags_up
??+stopped
end
end
### moet hier niet getest worden of op goal positie zijn beland?
none### Target position in x/
none### Target position in y/
```

```
none### Target position in phi/

0.5### Allowed error in x,y/ (general value, no planning-paramater dependence done

0.5### Allowed error in phi/ (general value, no planning-paramater dependence done
??+SO success
??+SO opt 1
end### get_some_flags/probable result
```

B.3 load_goal.scn

```
simple### /
Load goal position to DM### create_simple_op/
??SO PP pts
end### edit_global_flags/for simple op
end### edit_local_flags/for simple op
end### map_flags_down/
end### map_flags_up/
??PP data
??+SO success
end### get_some_flags/ On<-on 'goal position stored<-
??+SO not found
??+SO failed
end### get_some_flags/ On<-off 'goal position stored'<-
??success
??+SO success
end### get_some_flags/ On<-on 'Success<-
??+SO not found
??+SO failed
end### get_some_flags/ On<-off 'Success'<-
end### turn on/map_flags_up
??success
end### get_some_flags/ Off<-on 'Success'<-
??+SO success
end### get_some_flags/ Off<-off 'Success'<-
end### turn off/map_flags_up
end
end
??PARAM
1###N path segments
??AD
none### End X/
none### End Y/
```

```
none### End Phi/
0.35### Max speed/
??+SO success
end### get_some_flags/probable result
```

B.4 pp_goto.scn

```
action### /
Plan path from current to stored goal position
??ACT PP goto### /
end### edit_global_flags/for action
end### edit_local_flags/for action
??-finished
end### map_flags_down/
??-finished
end### map_flags_up/
end### turn on/map_flags_up
??PP data
end### get_some_flags/ Off<-on 'goal position stored'<-
end### get_some_flags/ Off<-off 'goal position stored'<-
end### turn off/map_flags_up
??+PP data### get_some_flags/global initial On:'goal position stored'
end### get_some_flags/global initial
??-evade### get_some_flags/local initial Off:'CA evade obstacle'
end### get_some_flags/local initial
end### get_some_flags/global during
end### get_some_flags/local during
??+TC path### get_some_flags/global final On:'clothoid path defined'
end### get_some_flags/global final
??+ACT normal
end### get_some_flags/local final
25### /
scenario pp_eo.scn### create_eo_node
end### /
```

B.4.1 pp_eo.scn

```
Clothoid path planner### create_eo_node/
??PP
??DEF PP failed
??DEF PP no via
end### edit_local_flags/for eo node
```

```
end### map_flags_up/
??TC path
+0### get_some_flags/ On<-on 'clothoid path defined<- On:'eo_state running'
end### get_some_flags/ On<-on 'clothoid path defined<-
+4### get_some_flags/ On<-off 'clothoid path defined'<- On:'eo_state stopped on er
end### get_some_flags/ On<-off 'clothoid path defined'<-
??severe global
end### get_some_flags/ On<-on 'severe error<-
end### get_some_flags/ On<-off 'severe error'<-
local 1
end### get_some_flags/ On<-on 'No error<-
+0### get_some_flags/ On<-off 'No error'<- On:'eo_state running'
end### get_some_flags/ On<-off 'No error'<-
end### turn on/map_flags_up
??no TC path
+0### get_some_flags/ Off<-on 'clothoid path not defined'<- On:'eo_state running'
end### get_some_flags/ Off<-on 'clothoid path not defined'<-
+2### get_some_flags/ Off<-off 'clothoid path not defined'<- On:'eo_state received
+1## just init### get_some_flags/ Off<-off 'clothoid path not defined'<- On:'eo_st
end### get_some_flags/ Off<-off 'clothoid path not defined'<-
??severe global
+0### get_some_flags/ Off<-on 'severe error'<- On:'eo_state running'
end### get_some_flags/ Off<-on 'severe error'<-
+2### get_some_flags/ Off<-off 'severe error'<- On:'eo_state received new params'
+1## just init### get_some_flags/ Off<-off 'severe error'<- On:'eo_state no parame
end### get_some_flags/ Off<-off 'severe error'<-
end### turn off/map_flags_up
yes### /
always### /
??PARAM
??PARAM
map_readDM
m
??FEAT ## map should be here
map class## The name### /
l### /
none
map source
l
??ENVIRONMENT_SOURCE   ## check this too
pose class
l
??DM_CLASS_POSE_SEG
pose source
```

```
l
none
plan type
l
??PP global rel
end### /
+0### get_some_flags/probable result On:'eo_state running'
end### get_some_flags/probable result
```

B.5 goal1_2_goal1br.data

```
### data for goal1_2_goal1br.scn
read test_goal1.data
read load_goal1br.data
read pp_goto.data

### data for test_goal1.scn
read goal1.pos

### data for load_goal.scn
read goal1br.pos### goal position

### data for pp_goto.scn
read pp_eo.data

### data for pp_eo.scn
??DM_SUB_CLASS_ENVIRONMENT
??AD
```

Bijlage C

test script

Dit test script laat de output zien van MARIE tijdens een succesvolle test. de volgende stukken planinformatie worden door de on-line planner aangeleverd:

plan pad van 1 naar 1br
stop smooth
stop smooth
ca mode
stop smooth
stop smooth
stop smooth
ca mode
stop smooth
ca mode
stop smooth
stop smooth
rij van 1 naar 1br
volg de muur van 1br naar 2b
plan pad van 2b naar doelpositie
ca mode
plan pad van 2b naar doelpositie
rij van 2b naar doelpositie
plan pad van huidige positie naar doel positie
rij van huidige positie naar doelpositie
stop smooth
plan pad van huidige positie naar doel positie
rij van huidige positie naar doelpositie
ca mode
plan pad van huidige positie naar doel positie
rij van huidige positie naar doelpositie

```
Script started on Thu Nov 02 14:05:33 2000
(ftrpstra@night 1) rlogin vw1


-> cd "/home/ftrpstra"
value = 0 = 0x0
-> ld <ad_shell.vx
value = 0 = 0x0
-> ad_shell
Spawned vc_control task with id = 0x3b8414
nNot all steering motor commando's successful
Spawned DmFeature task with id = 0x3a7bec
Spawned DmParameter task with id = 0x3a5900
Spawned DmWorld task with id = 0x3a3614
Spawned Trajectory controller task with id = 0x39ce08
0x39ce08 (TcTask): waiting for eo_client on port 1511
Spawned Fuzzy controller task with id = 0x39654c
Spawned Sensor controller task with id = 0x390898
Spawned Path planner task with id = 0x38b8d4
Spawned Docking spot detector task with id = 0x382728
SdTask: initialized
Spawned Line segment extractor task with id = 0x37e254
ad_init: Attempting to connect to 'TC - Trajectory controller'
ad_init: Attempting to connect to 'SC - Sensor-based controller'
ad_init: Attempting to connect to 'WS - Wall-sensor'
ad_init: Attempting to connect to 'PP - Path planner'
ad_init: Attempting to connect to 'FZ - Fuzzy controller'
ad_init: Attempting to connect to 'US - Ultra sonic system'
ad_init: Attempting to connect to 'SD - Line segment extractor'
ad_init: Attempting to connect to 'DD - Docking spot detector'
Purged 0 old path segments from the data-manager
Task 'DmParameter' has id 3a5900
Task 'Trajectory controller' has id 39ce08
Task 'Fuzzy controller' has id 39654c
Task 'Main task' has id 3e7010
Task 'vc_control' has id 3b8414
Task 'DmWorld' has id 3a3614
Task 'Path planner' has id 38b8d4
Task 'Line segment extractor' has id 37e254
Task 'Bosus' has id 3a1318
Task 'Sensor controller' has id 390898
Task 'Docking spot detector' has id 382728
Task 'DmFeature' has id 3a7bec
Task 'vc_info' has id 3b02b4
```

```
Task 'UsRead' has id 39fef8
ad_sock = 53, fd_last = 54
0 values for steering positions stored
nr. desired  actual
marie: Action dispatcher: accepted client on socket nr. 69
Accepted client of socket nr 69
0 values for steering positions stored
nr. desired  actual
marie: ad_dispatch: for node 0
For this node no parent node has been defined
looking at child node with id: 30000
node being tested on initial conditions: 30000
ad_dispatch: for node 30000
ad_dispatch: this node's parent is: 0
looking at child node with id: 21000
node being tested on initial conditions: 21000
ad_dispatch: for node 21000
ad_dispatch: this node's parent is: 30000
looking at child node with id: 5014
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 1111111
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 65792
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
node being tested on initial conditions: 22000
ad_dispatch: for node 22000
ad_dispatch: this node's parent is: 30000
looking at child node with id: 5016
node being tested on initial conditions: 5016
ad_dispatch: for node 5016
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
start met herplannen
```

[1]

```
message to be sent:
...3.5..8.01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node being tested on initial conditions: 32121
ad_dispatch: for node 32121
ad_dispatch: this node's parent is: 22000
looking at child node with id: 5001
node being tested on initial conditions: 5001
ad_dispatch: for node 5001
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
so_test_distance: current pos:  5.639  1.199 -1.570
                     goal pos:  5.639  1.199 -1.570
Angle and distance OK
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 9200
ad_dispatch: for node 9200
ad_dispatch: this node's parent is: 32121
looking at child node with id: 1515
0x38b8d4 (PpTask): found 1 records

path planning skipped for n_loop = 3
skipped corner = 6.341000, -.450000
task_tree id: 5016
task_tree id: 32121
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
01.............................   Local return state
message to be sent:
.............................   Global return state
01.............................   Local return state
node id: 5016
```

[1] herplanner wordt voor het eerst aangeroepen

```
node id: 32121
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node being tested on initial conditions: 5003
ad_dispatch: for node 5003
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
```

```
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
```

```
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
task_tree id: 5013
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5013
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
task_tree id: 5013
task_tree id: 5013
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
```

```
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5013
node id: 5013
node id: 5003
node being tested on initial conditions: 5003
ad_dispatch: for node 5003
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
```

```
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5003
node being tested on initial conditions: 5003
ad_dispatch: for node 5003
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
task_tree id: 5016
task_tree id: 32121
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
task_tree id: 5013
task_tree id: 5013
task_tree id: 5003
task_tree id: 5013
task_tree id: 5003
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
```

```
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 5016
node id: 32121
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5013
node id: 5013
node id: 5003
node id: 5013
node id: 5003
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 22000
looking at child node with id: 253932
AD_FAIL_NO_NEXT
...3.5..8.......6.......4.......   Required global final
...3.5.7..01..4.....0....5.....1   Actual global status
..................8.............   Required local final
.1..4.........4.................   Actual local status
task_tree id: 21000
task_tree id: 22000
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1..4.........4................1   Local return state
message to be sent:
.............................   Global return state
.1..4.........4................1   Local return state
node id: 21000
node id: 22000
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 30000
looking at child node with id: 253932
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
```

```
start met herplannen
message to be sent:
...3.5.7..01..4.....0....5.....1   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node being tested on initial conditions: 11003
ad_dispatch: for node 11003
ad_dispatch: this node's parent is: 30000
looking at child node with id: 9100
node being tested on initial conditions: 9100
ad_dispatch: for node 9100
ad_dispatch: this node's parent is: 11003
looking at child node with id: 1511
0x39ce08 (TcTask): found 9 records
0x39ce08 (TcTask): id = 1, ca_mode = 1
0x39ce08 (TcTask): id = 2, ca_mode = 1
0x39ce08 (TcTask): id = 3, ca_mode = 1
0x39ce08 (TcTask): id = 4, ca_mode = 1
0x39ce08 (TcTask): id = 5, ca_mode = 1
0x39ce08 (TcTask): id = 6, ca_mode = 1
0x39ce08 (TcTask): id = 7, ca_mode = 1
0x39ce08 (TcTask): id = 8, ca_mode = 1
0x39ce08 (TcTask): id = 9, ca_mode = 1
0x39ce08 (TcTask): id = 0, ca_mode = 1
node being tested on initial conditions: 5001
ad_dispatch: for node 5001
ad_dispatch: this node's parent is: 11003
looking at child node with id: 253932
so_test_distance: current pos:  8.006 -1.072   .116
                     goal pos:  8.000  -.900  0.030
Angle and distance OK
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
start met herplannen
message to be sent:
...3.5..8.0.2.45....0....5...9..   Global return state
```

```
.1...........4.................   Local return state
message to be sent:
...............................   Global return state
.1...........4.................   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node being tested on initial conditions: 12000
ad_dispatch: for node 12000
ad_dispatch: this node's parent is: 30000
looking at child node with id: 5010
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 12000
looking at child node with id: 253932
node being tested on initial conditions: 5008
ad_dispatch: for node 5008
ad_dispatch: this node's parent is: 12000
looking at child node with id: 253932
cur_x 8.006536 cur_y -1.072803 cur_phi .116163
node being tested on initial conditions: 5009
ad_dispatch: for node 5009
ad_dispatch: this node's parent is: 12000
looking at child node with id: 253932
node being tested on initial conditions: 9400
ad_dispatch: for node 9400
ad_dispatch: this node's parent is: 12000
looking at child node with id: 1512
0x39ce08 (TcTask): found 4 records
0x39ce08 (TcTask): id = 10, ca_mode = 1
0x39ce08 (TcTask): id = 11, ca_mode = 1
0x39ce08 (TcTask): id = 12, ca_mode = 1
node being tested on initial conditions: 9999
ad_dispatch: for node 9999
ad_dispatch: this node's parent is: 12000
looking at child node with id: 14000
ad_dispatch: for node 14000
ad_dispatch: this node's parent is: 9999
looking at child node with id: 1513
0x390898 (ScTask): found 5 records
ad_dispatch: for node 14002
ad_dispatch: this node's parent is: 9999
```

```
looking at child node with id: 1513
ad_dispatch: for node 14001
ad_dispatch: this node's parent is: 9999
looking at child node with id: 1513
ad_dispatch: for node 14002
ad_dispatch: this node's parent is: 9999
looking at child node with id: 1513
ad_dispatch: for node 14001
ad_dispatch: this node's parent is: 9999
looking at child node with id: 1513
ad_dispatch: for node 14002
ad_dispatch: this node's parent is: 9999
looking at child node with id: 1513
ad_dispatch: for node 14001
ad_dispatch: this node's parent is: 9999
looking at child node with id: 1513
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 12000
looking at child node with id: 253932
Failed during condition
...............5.............9..   Required global during
...3.5....0.2345...90.2.........   Actual global status
.............................   Required local during
01............4...8............   Actual local status
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
start met herplannen
message to be sent:
...3.5....0.2345...90.2.........   Global return state
01............4...8......5......   Local return state
message to be sent:
.............................   Global return state
01............4...8......5......   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node being tested on initial conditions: 32121
```

```
ad_dispatch: for node 32121
ad_dispatch: this node's parent is: 30000
looking at child node with id: 21000
node being tested on initial conditions: 21000
ad_dispatch: for node 21000
ad_dispatch: this node's parent is: 32121
looking at child node with id: 5014
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 1111111
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 65792
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 9200
ad_dispatch: for node 9200
ad_dispatch: this node's parent is: 32121
looking at child node with id: 1515
0x38b8d4 (PpTask): found 2 records

Path with one curve: endpoint can not be reached

Path with one curve: required radius too small
PpTask: path planning failed
```

[2]

```
task_tree id: 21000
```

[2] onverwacht falen van de pad planner, wordt verop door de on-line planner opgelost, dit is een voorbeeld van succes door herplannen

```
task_tree id: 5006
task_tree id: 5010
task_tree id: 9200
start met herplannen
message to be sent:
...3.5....012345.7.90.2..5.....   Global return state
01.......................6.....   Local return state
message to be sent:
.............................   Global return state
01.......................6.....   Local return state
node id: 21000
node id: 5006
node id: 5010
node id: 9200
node id: 5003
node being tested on initial conditions: 5003
ad_dispatch: for node 5003
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
task_tree id: 21000
task_tree id: 5006
task_tree id: 5010
task_tree id: 9200
task_tree id: 5003
start met herplannen
message to be sent:
...3.5....012.45.7.90.2..5.....   Global return state
.1...........................   Local return state
message to be sent:
.............................   Global return state
.1...........................   Local return state
node id: 21000
node id: 5006
node id: 5010
node id: 9200
node id: 5003
node id: 32121
node being tested on initial conditions: 32121
ad_dispatch: for node 32121
ad_dispatch: this node's parent is: 32121
looking at child node with id: 21000
node being tested on initial conditions: 21000
ad_dispatch: for node 21000
ad_dispatch: this node's parent is: 32121
```

```
looking at child node with id: 5014
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 1111111
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 65792
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 9200
ad_dispatch: for node 9200
ad_dispatch: this node's parent is: 32121
looking at child node with id: 1515
0x38b8d4 (PpTask): found 1 records

path planning skipped for n_loop = 3
skipped corner = 19.393000, -1.427999
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
task_tree id: 32121
start met herplannen
message to be sent:
...3.5.7..012.45.7.90.2..5.....   Global return state
01..4.........4....9............   Local return state
message to be sent:
.............................   Global return state
01..4.........4....9............   Local return state
```

```
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node being tested on initial conditions: 11001
ad_dispatch: for node 11001
ad_dispatch: this node's parent is: 30000
looking at child node with id: 9100
node being tested on initial conditions: 9100
ad_dispatch: for node 9100
ad_dispatch: this node's parent is: 11001
looking at child node with id: 1511
0x39ce08 (TcTask): found 12 records
0x39ce08 (TcTask): id = 14, ca_mode = 1
0x39ce08 (TcTask): id = 15, ca_mode = 1
0x39ce08 (TcTask): id = 16, ca_mode = 1
Maximum time for this segment exceeded!
Time now = 56.400001, end-time for segment = 56.256816
Segment length = .467206, v_start = .000000, v_end = .305681
start pose: x = 17.920000, y = -.600000, phi = 0.030000, curve = .000000
end pose: x = 18.386995, y = -.585985, phi = 0.030000, curve = .000000
t_max 33.056815
Failed during condition
```

[3]

```
.......7.........7.9.............   Required global during
...3.5..8.0.2.45.7.90.2.........   Actual global status
................................   Required local during
01..............................   Actual local status
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
task_tree id: 32121
task_tree id: 11001
start met herplannen
message to be sent:
```

[3] deze "rij macro" en alle volgende die in deze test voorkomen zijn een voorbeeld van "step by step"

```
...3.5..8.0.2.45.7.90.2.........   Global return state
01.......................5......   Local return state
message to be sent:
.............................      Global return state
01.......................5......   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node id: 32121
node being tested on initial conditions: 32121
ad_dispatch: for node 32121
ad_dispatch: this node's parent is: 30000
looking at child node with id: 21000
node being tested on initial conditions: 21000
ad_dispatch: for node 21000
ad_dispatch: this node's parent is: 32121
looking at child node with id: 5014
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 1111111
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 65792
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 9200
```

```
ad_dispatch: for node 9200
ad_dispatch: this node's parent is: 32121
looking at child node with id: 1515
0x38b8d4 (PpTask): found 1 records

path planning skipped for n_loop = 3
skipped corner = 19.393000, -1.427999
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
task_tree id: 32121
task_tree id: 11001
task_tree id: 32121
start met herplannen
message to be sent:
...3.5.7..012.45.7.90.2..5.....   Global return state
01..4.........4....9............   Local return state
message to be sent:
.............................     Global return state
01..4.........4....9............   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node id: 32121
node id: 11001
node being tested on initial conditions: 11001
ad_dispatch: for node 11001
ad_dispatch: this node's parent is: 30000
looking at child node with id: 9100
node being tested on initial conditions: 9100
ad_dispatch: for node 9100
ad_dispatch: this node's parent is: 11001
looking at child node with id: 1511
0x39ce08 (TcTask): found 12 records
0x39ce08 (TcTask): id = 26, ca_mode = 1
0x39ce08 (TcTask): id = 27, ca_mode = 1
0x39ce08 (TcTask): id = 28, ca_mode = 1
Maximum time for this segment exceeded!
```

```
Time now = 89.599998, end-time for segment = 89.456817
Segment length = .467206, v_start = .000000, v_end = .305681
start pose: x = 17.920000, y = -.600000, phi = 0.030000, curve = .000000
end pose: x = 18.386995, y = -.585985, phi = 0.030000, curve = .000000
t_max 33.056815
Failed during condition
.......7.........7.9.............   Required global during
...3.5..8.0.2.45.7.90.2.........   Actual global status
.............................   Required local during
01...........................   Actual local status
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
task_tree id: 32121
task_tree id: 11001
task_tree id: 32121
task_tree id: 11001
start met herplannen
message to be sent:
...3.5..8.0.2.45.7.90.2.........   Global return state
01.....................5......   Local return state
message to be sent:
.............................   Global return state
01.....................5......   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node id: 32121
node id: 11001
node id: 5013
node being tested on initial conditions: 5013
ad_dispatch: for node 5013
ad_dispatch: this node's parent is: 30000
looking at child node with id: 253932
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
```

```
task_tree id: 12000
task_tree id: 32121
task_tree id: 11001
task_tree id: 32121
task_tree id: 11001
task_tree id: 5013
start met herplannen
message to be sent:
...3.5..8.0.2.45.7.90.2.........   Global return state
.1.............................   Local return state
message to be sent:
.............................   Global return state
.1.............................   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node id: 32121
node id: 11001
node id: 5013
node id: 32121
node being tested on initial conditions: 32121
ad_dispatch: for node 32121
ad_dispatch: this node's parent is: 30000
looking at child node with id: 21000
node being tested on initial conditions: 21000
ad_dispatch: for node 21000
ad_dispatch: this node's parent is: 32121
looking at child node with id: 5014
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 1111111
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 65792
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
```

```
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 9200
ad_dispatch: for node 9200
ad_dispatch: this node's parent is: 32121
looking at child node with id: 1515
0x38b8d4 (PpTask): found 1 records

path planning skipped for n_loop = 3
skipped corner = 19.393000, -1.427999
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
task_tree id: 32121
task_tree id: 11001
task_tree id: 32121
task_tree id: 11001
task_tree id: 5013
task_tree id: 32121
start met herplannen
message to be sent:
...3.5.7..012.45.7.90.2..5.....   Global return state
01..4.........4....9............   Local return state
message to be sent:
.............................     Global return state
01..4.........4....9............   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node id: 32121
```

```
node id: 11001
node id: 5013
node id: 32121
node id: 11002
node being tested on initial conditions: 11002
ad_dispatch: for node 11002
ad_dispatch: this node's parent is: 30000
looking at child node with id: 9100
node being tested on initial conditions: 9100
ad_dispatch: for node 9100
ad_dispatch: this node's parent is: 11002
looking at child node with id: 1511
0x39ce08 (TcTask): found 12 records
0x39ce08 (TcTask): id = 38, ca_mode = 1
0x39ce08 (TcTask): id = 39, ca_mode = 1
0x39ce08 (TcTask): id = 40, ca_mode = 1
Maximum time for this segment exceeded!
Time now = 122.800003, end-time for segment = 122.656814
Segment length = .467206, v_start = .000000, v_end = .305681
start pose: x = 17.920000, y = -.600000, phi = 0.030000, curve = .000000
end pose: x = 18.386995, y = -.585985, phi = 0.030000, curve = .000000
t_max 33.056815
Failed during condition
.......7.........7.9.............   Required global during
...3.5..8.0.2.45.7.90.2.........   Actual global status
.............................   Required local during
01............4.................   Actual local status
task_tree id: 21000
task_tree id: 22000
task_tree id: 5013
task_tree id: 11003
task_tree id: 12000
task_tree id: 32121
task_tree id: 11001
task_tree id: 32121
task_tree id: 11001
task_tree id: 5013
task_tree id: 32121
task_tree id: 11002
start met herplannen
message to be sent:
...3.5..8.0.2.45.7.90.2.........   Global return state
01............4.........5......   Local return state
message to be sent:
```

```
.............................   Global return state
01...........4.........5......   Local return state
node id: 21000
node id: 22000
node id: 5013
node id: 11003
node id: 12000
node id: 32121
node id: 11001
node id: 32121
node id: 11001
node id: 5013
node id: 32121
node id: 11002
node id: 5003
node being tested on initial conditions: 5003
ad_dispatch: for node 5003
ad_dispatch: this node's parent is: 30000
looking at child node with id: 253932
AD_FAIL_NO_NEXT
...3.5.........6.......4.......   Required global final
...3.5..8.0.2.45.7.90.2.........   Actual global status
...............6...............   Required local final
01..4.........4...8.............   Actual local status
task_tree id: 30000
start met herplannen
message to be sent:
...3.5..8.0.2.45.7.90.2.........   Global return state
01..4.........4...8...........1   Local return state
message to be sent:
.............................   Global return state
01..4.........4...8...........1   Local return state
node id: 30000
node id: 5003
node being tested on initial conditions: 5003
ad_dispatch: for node 5003
ad_dispatch: this node's parent is: 0
looking at child node with id: 253932
task_tree id: 30000
task_tree id: 5003
start met herplannen
message to be sent:
.1.3.5..8.0.2.45.7.90.2.........   Global return state
.1...........................   Local return state
```

```
message to be sent:
.............................   Global return state
.1...........................   Local return state
node id: 30000
node id: 5003
node id: 32121
node being tested on initial conditions: 32121
ad_dispatch: for node 32121
ad_dispatch: this node's parent is: 0
looking at child node with id: 21000
node being tested on initial conditions: 21000
ad_dispatch: for node 21000
ad_dispatch: this node's parent is: 32121
looking at child node with id: 5014
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 1111111
node being tested on initial conditions: 5014
ad_dispatch: for node 5014
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
so_purge: purged 0 elements in class 65792
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 21000
looking at child node with id: 253932
node being tested on initial conditions: 5006
ad_dispatch: for node 5006
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 5010
ad_dispatch: for node 5010
ad_dispatch: this node's parent is: 32121
looking at child node with id: 253932
node being tested on initial conditions: 9200
ad_dispatch: for node 9200
ad_dispatch: this node's parent is: 32121
looking at child node with id: 1515
0x38b8d4 (PpTask): found 1 records

path planning skipped for n_loop = 3
skipped corner = 19.393000, -1.427999
```

```
task_tree id: 30000
task_tree id: 5003
task_tree id: 32121
start met herplannen
message to be sent:
.1.3.5.7..012.45.7.90.2..5.....   Global return state
01..4.........4....9............   Local return state
message to be sent:
.............................     Global return state
01..4.........4....9............   Local return state
node id: 30000
node id: 5003
node id: 32121
node id: 11002
node being tested on initial conditions: 11002
ad_dispatch: for node 11002
ad_dispatch: this node's parent is: 0
looking at child node with id: 9100
node being tested on initial conditions: 9100
ad_dispatch: for node 9100
ad_dispatch: this node's parent is: 11002
looking at child node with id: 1511
0x39ce08 (TcTask): found 12 records
0x39ce08 (TcTask): id = 50, ca_mode = 1
0x39ce08 (TcTask): id = 51, ca_mode = 1
0x39ce08 (TcTask): id = 52, ca_mode = 1
Read error from network: Connection reset by peerConnection closed.
(ftrpstra@night 2) exit
script done on Thu Nov 02 14:12:31 2000
```