Een simulator voor RoboSoccer Middle Size League

Ewout Boendermaker
Universiteit van Amsterdam

1. Inleiding

Multi agent systemen worden in de huidige wereld steeds belangrijker. Op steeds meer gebieden gaan ze in de toekomst ingezet worden. Een actuele toepassing is de rescue robots die zijn gebruikt bij het zoeken naar overlevenden na de instorting van het World Trade Center. Robots zijn daar ingezet om tussen het puin naar overlevenden te zoeken op plaatsen waar het te gevaarlijk was voor mensen. Zij hadden sensoren aan boord om geluid te detecteren en beelden te maken. Deze robots werden nog van afstand met de hand bestuurd. De uitdaging ligt er met name om deze robots intelligenter te maken en ze uiteindelijk semi autonoom te laten opereren[1]. Ook worden dergelijke robots gebruikt voor het zoeken naar en het opruimen van mijnen. Ricardo Cassinis[2] onderzoekt technieken om met groepen simpele robots landmijnen op te sporen. Hierbij is sprake van samenwerking tussen de verschillende robots om een gemeenschappelijk doel te bereiken. Rekening moet worden gehouden met het verliezen van een robot, nadat deze op een mijn is gelopen. Zijn taak moet dan worden overgenomen door de andere robots. Daimler Chrysler doet al jaren onderzoek naar verhoging van de intelligentie van een auto. In eerste instantie om de veiligheid van de bestuurder en de omgeving te verhogen, maar ook om uiteindelijk toe te gaan naar een autonome auto. Sensoren nemen de omgeving op, waardoor obstakels en voetgangers kunnen worden ontweken. Het uiteindelijke doel van deze semi autonome auto's van de toekomst is dat ze met elkaar en met de infrastructuur communiceren.
In het algemeen kan gesteld worden, dat in het onderzoek naar multi agent systemen de samenwerking van het team centraal staat. Het doel van het team moet worden vertaald naar een individueel doel voor ieder individu. Hierbij wordt gebruik gemaakt van communicatie tussen de verschillende agenten en van sensoren die signalen uit de omgeving verwerken.

Robot voetbal is een gebied van onderzoek waar vrijwel alle aspecten van multi agent systemen aan de orde komen. Dit Robosoccer, zoals de Amerikanen zeggen, terwijl voetbal in de rest van wereld football heet, wordt in verschillende leagues gespeeld.
In de simulator league is er een netwerk van computersystemen waar een voetbal simulator op draait. Software van beide partijen kan hierop inloggen om zo teams van softbots tegen elkaar te laten spelen. Het wedstrijdverloop kan op een monitor gevolgd worden. Ofschoon dit type robot voetbal zeer geschikt is om de strategische kant van multi agent systemen te ontwikkelen, is het realiteitsgehalte van de waarneming altijd beperkt.
In de small size league spelen robots van klein formaat op een soort tafel. Er is één camera die het hele veld overziet en iedere robot van een wereldbeeld voorziet. Ofschoon hier zowel teamtactiek als onzekere waarneming een rol spelen, is het nog niet zo dat iedere robot eigen sensoren heeft en een eigen wereldbeeld. Omdat niet iedere agent een eigen wereldbeeld heeft, is het multi agent aspect van deze toepassing beperkt. Er is sprake van een centrale regeling en niet van een gedistribueerde regeling.
In de middle size league spelen robots tegen elkaar die allemaal zijn uitgerust met een camera om het veld en de objecten erin waar te nemen. Ook het feit dat de robots moeten samenwerken om een gemeenschappelijk doel te bereiken, dat zij kunnen communiceren met elkaar om waarnemingen door te geven en rolpatronen af te spreken en dat er rekening moet worden gehouden met een onvoorspelbare tegenstander, maakt dit een bijzonder interessante toepassing van multi agent systemen.

De Universiteit van Amsterdam, de Technische Universiteit en de Universiteit van Utrecht doen gezamenlijk onderzoek naar de middle size league voetbal en hebben een team van robots, “Clockwork Orange” genaamd.

Een team van robots in de middle size league wordt aangestuurd door een netwerk van verschillende software modules, die in een meerlaags architectuur georganiseerd is. Er zijn modules die het team aansturen en modules die een individuele robot aansturen. Teneinde deze modules te kunnen ontwikkelen, testen en verbeteren is het van belang dat alle modules ook kunnen draaien zonder dat er een team van echte robots beschikbaar is. Om bij ieder experiment alle robots op te stellen is zeer tijdrovend. Bovendien zijn ze eindig in gebruik, aangezien ze werken op accu’s. Door te simuleren is het voor te stellen dat de modules uren achtereen draaien zonder dat menselijke interactie benodigd is, teneinde algoritmes voor teamgedrag en individueel gedrag te kunnen bestuderen en evalueren. Dat betekent dat er een stuk software moet komen om robots te vervangen en dat er een stuk software moet komen dat een echte wereld simuleert, waar deze virtuele robots in kunnen bewegen. De algoritmes en de implementatie om het middle size voetbal te simuleren met het doel het verbeteren en het testen van de software modules, is het onderwerp van dit afstudeerverslag.

De volgende eigenschappen zijn van belang om de robots in een gesimuleerde werkelijkheid te laten draaien. Ten eerste moeten de virtuele robots in de bestaande software worden opgenomen, de software mag niet merken dat er geen fysieke robot wordt aangestuurd, maar een stukje software dat de robot vervangt. Ten tweede moet de simulator de omstandigheden modelleren die op het robot voetbal van toepassing zijn. Dit houdt onder andere in dat de wereld waar de robots in bewegen, moet worden vervangen door een wereldmodel waar de gesimuleerde robots in bewegen. Het resultaat is dat er een wedstrijd kan worden gespeeld door de software modules die gesimuleerde robots aansturen in een gesimuleerde wereld. Onderzoek zit in het realiteitsgehalte van de simulator en hoe goed deze voor het testen van de modules bruikbaar is.

De structuur van dit verslag is als volgt. In hoofdstuk 2 wordt robot voetbal in meer detail besproken en worden de belangrijkste en meest relevante facetten van de software uiteengezet. Hieruit wordt duidelijk welke gedeeltes van de software herschreven moeten worden om te kunnen functioneren zonder dat er een fysieke robot aan te pas komt. Hoofdstuk 3 vertelt over het ontwerp en de werking van de simulator. Hoe een robot wordt gesimuleerd wordt hierin uiteen gezet en het creëren van een virtuele wereld komt aan bod. Hoofdstuk 4 behandelt de implementatie van de simulator. De technische oplossingen die gekozen zijn, komen hierin aan bod en de problemen waar tegenaan werd gelopen bij de implementatie. Hoofdstuk 5 gaat over de resultaten, welke situaties er gesimuleerd kunnen worden en wat de uitkomsten hiervan zijn.

2. Robosoccer

2.1 inleiding

De Universiteit van Amsterdam heeft in samenwerking met de Technische Universiteit Delft en de Universiteit van Utrecht een nationaal robot voetbalteam genaamd “Clockwork Orange” opgezet. Dit team is ontworpen om mee te doen in de robosoccer middle size league[9]. De teams uit deze league bestaan uit drie veldspelers en een keeper. De spelers zijn robots, uitgerust met twee motoren om te bewegen, een camera om objecten te herkennen en de eigen positie op het veld te bepalen en een schiet mechanisme om tegen de bal te trappen. Dit is een typisch multi agent systeem, waarbij de agenten opereren volgens het principe: eerst waarnemen, dan redeneren, dan acteren. Tijdens al deze activiteiten wordt gecommuniceerd met de andere agenten. Het waarnemen gebeurt met de camera, waarbij dan objecten binnen het gezichtsveld van de camera moeten worden herkend. Om de verschillende objecten op het veld gemakkelijker te kunnen herkennen, hebben zij allemaal een verschillende kleur. Op basis hiervan wordt geredeneerd over de te ondernemen actie. Voorbeelden van acties zijn dribbelen naar een positie, rijden naar een positie of schieten tegen de bal. Nadat de actie is geselecteerd, kan deze worden uitgevoerd.

2.2 Hardware

De robot die het Delftse en Amsterdamse team gebruiken is de Nomad Scout robot. Dit is een mobiele robot uitgerust met een vision systeem, 16 ultrasone sensoren, een tast bumper ring, odometrie sensoren en een luchtdruk gestuurd schop mechanisme. Aan boord is een Pentium 2 processor. Deze communiceert met een low-level processor board voor de aandrijving van de motoren door middel van een seriële poort. Een TMS320C14 DSP, een digitale signaal processor (DSP), is verantwoordelijk voor het aansturen van de motor. De robot kan ongeveer een uur draaien op de accu’s. Voor communicatie tussen de robots wordt wireless Ethernet gebruikt, BreezeCom SA-10 PRO, die een actieradius van ongeveer 20 tot 50 meter heeft. De specificaties zijn in figuur 2.1 gegeven. Om het schop mechanisme te laten werken is er een luchttank met genoeg lucht voor 50 keer trappen op volle kracht.

Specifications

Diameter:	41 cm.
Height:	35 cm.
Weight:	25 kg. (incl. batteries)
Payload:	5 kg.
Battery Power:	432 watt-hour (removable)
2 wheel differential drive @ geometric center	
Omnidirectional motion	
Ground Clearance:	1.5 cm
Speed:	1.0 m/sec
Acceleration:	2 m/s^2
Encoder Resolution:	
Translation:	756 counts/cm
Rotation:	230 counts/degree

Figuur 2.1 specificatie van de Nomad Scout

2.3 Software

2.3.1 inleiding

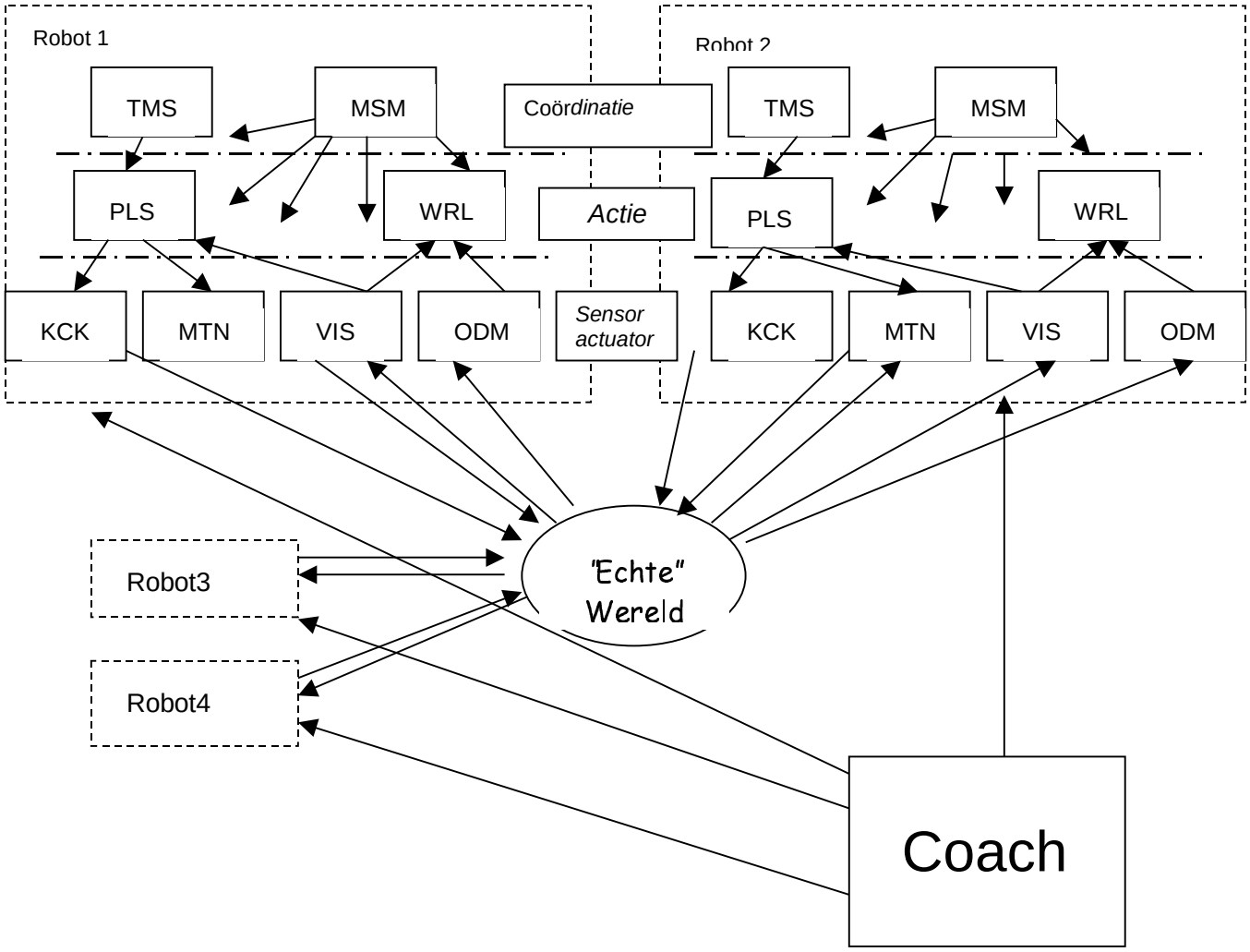
Een team van voetbalrobots wordt aangestuurd door een netwerk van software modules. Deze modules opereren op diverse niveaus binnen de architectuur. Op het hoogste niveau zijn de coördinerende modules aanwezig. Er is één module die het hele team aanstuurt. Dit is de ***Coach Module***. Deze module zorgt ervoor dat bij alle robots bekend is welke kant er op wordt gespeeld en wijst ook aan alle robots een rol toe, zoals bijvoorbeeld keeper. Op het hoogste niveau opereert ook de ***Team Skills Module***. Deze module bepaalt het teamgedrag op grond van de situatie op het veld. Zo kan de situatie vereisen dat er wordt verdedigd, dat de bal moet worden onderschept of dat er aangevallen moet worden. Verder opereert op het hoogste niveau de ***Mission Manager***. Deze module waakt erover dat elke robot nog actief is en zorgt ervoor dat de software naar behoren draait.

Het middelste niveau is het actieniveau, waarop maar twee modules opereren. Op dit niveau opereren de modules die de input voor het hoogste niveau verzorgen en die de sensorinformatie verwerken en de acties selecteren. Om te beginnen is dit de ***Player Skills Module,*** die de strategie van de TeamSkills Module uitvoert. Deze module laat een robot dribbelen, bewegen of schieten. Ook de ***World Knowledge Module*** opereert op dit niveau. Deze module leest de sensoren uit en verwerkt deze informatie in het beeld dat iedere robot van de wereld bijhoudt. Ook communiceert deze module met de **World Knowledge Modules** van teamgenoten om zo een zo compleet en nauwkeurig mogelijk wereldbeeld te krijgen.

Op het laagste niveau opereren de modules die direct de sensoren en de actuatoren aansturen. De sensoren worden door de ***Vision Module*** en de ***Odometrie Module*** aangestuurd. De Vision Module leest de camera uit en herkent objecten in de beelden. Hiermee kan worden bepaald of een robot in balbezit is, wat van essentieel belang is

voor de te kiezen tactiek. Ook kan op grond van beelden de positie op het veld worden berekend. De Odometrie Module leest uit hoe vaak de wielen van de robot gedraaid hebben om zodoende de eigen positie in te kunnen schatten. De actuatoren worden door de ***Motion Module*** en de ***Kick Module*** aangestuurd. Deze modules modificeren dus de wereld. De Motion Module stuurt de motoren van de robot aan zodat er kan worden gereden, de Kick Module stuurt de kicker aan waar de bal mee getrapt kan worden. Deze actuatoren en sensoren zijn dus de enige modules die direct informatie van de wereld verkrijgen of de wereld beïnvloeden. Het is belangrijk dit vast te stellen aangezien dit een rol speelt bij het ontwerpen van de simulator. Deze modules zullen moeten worden aangepast om te werken met een gesimuleerde wereld in plaats van met een echte wereld.

Figuur 2.2 een schematische weergave van de architectuur van de 3 laags structuur



Legenda:

MSM	=	Mission Manager Module
TMS	=	Team Skills Module
PLS	=	Player Skills Module
WRL	=	World Knowledge Module
KCK	=	Kick Module
MTN	=	Motion Module
VIS	=	Vision Module
ODM	=	Odometrie Module

2.3.2 Het hoogste niveau

Bovenaan in de hiërarchie van een robot voetbal team staat de ***Coach Module***[3]. De Coach Module detecteert welke robots aanwezig zijn, zodat hij deze commando's kan geven. De coach geeft door aan de robots dat een wedstrijd begint, hij geeft aan wanneer een wedstrijd is afgelopen, hij geeft door als de software van de robot moet worden opgestart en wanneer een robot moet worden gestopt. Per team is er dus één coach. De coach communiceert met de Mission Manager.

De ***Mission Manager*** zorgt voor de uitvoering van de commando's van de coach. Dit is een module die op iedere robot draait om in de gaten te houden of alle software modules naar behoren opgestart worden. Nadat alle modules zijn opgestart, houdt de Mission Manager bij of alle modules blijven draaien. Alle modules die zijn gestart, geven een ready signaal terug, zodat duidelijk is wanneer alle modules van een robot zijn opgestart. Ook krijgt deze module heartbeats binnen van alle draaiende modules. Bij het ontbreken van een heartbeat onderneemt de Mission Manager passende actie. De Mission Manager geeft het sein door aan de overige modules als de wedstrijd begint of is afgelopen.

De ***Team Skills Module*** is de belangrijkste module op dit niveau. Deze genereert de team tactiek en is strategisch leidend voor een robot. De Team Skills Module is zo goed mogelijk op de hoogte van de positie van alle objecten op het veld en van de tactiek van de medespelers. Op grond van de situatie op het veld wordt de teamtactiek vastgesteld. Als het team in balbezit is, zal de tactiek een aanvallende tactiek zijn. Indien het team niet in balbezit is, zal de tactiek verdedigend of onderscheppend zijn. Als gevolg van de teamtactiek krijgt iedere robot een rol of taak toebedeeld. De rol die een robot krijgt toebedeeld, wordt doorgegeven aan de andere robots, die op hun beurt hierop kunnen anticiperen. Het bepalen van de teamtactiek en het verdelen van de individuele rollen van de robots is een continu proces.
Het hoogste niveau is dus verantwoordelijk voor de technische en tactische aansturing van een robot.

2.3.3 Het middelste niveau

Onder het coördinerende niveau staat het actie niveau. Op dit niveau opereren de Player Skills Module en de World Knowledge Module.
De ***Player Skills***[5]stuurt de Motion Module aan. Een commando om naar een bepaalde positie te bewegen vertaalt de Player Skills Module in een lineaire snelheid en een hoeksnelheid. Dit zijn de gegevens die de Motion Module nodig heeft om de motoren van de robot zodanig aan te sturen dat de gevraagde positie wordt bereikt. Ook laat de Player Skills Module de robot dribbelen. Feitelijk is dit rijden met de bal dus vergelijkbaar met gewoon bewegen, ware het niet dat zodanig moet worden bewogen dat hij de bal bij zich houdt. Onder de juiste omstandigheden kan de Player Skills Module ook de beslissing nemen om de bal te schieten. De Player Skills Module vraagt een aantal

malen per seconde aan de Vision Module naar de positie van objecten binnen het gezichtsveld van de camera. Op basis van deze gegevens wordt bepaald of sprake is van balbezit. Indien robots in zicht zijn, zullen deze worden ontweken. Ook worden gegevens over de positie van de goal gevraagd, zodat naar de goal kan worden gedribbeld en hierop kan worden geschoten.

De ***World Knowledge Module***[6] houdt een beeld bij van de wereld. Een robot krijgt van de Vision Module gegevens door over objecten in het gezichtsveld. De Vision Module hanteert hierbij egocentrische coördinaten. Om de positie van deze objecten op een bruikbare manier te communiceren naar andere robots, moeten deze coördinaten worden omgerekend naar absolute coördinaten. Het is hierbij van essentieel belang dat de robot op de hoogte is van zijn eigen positie. Als dit niet het geval is worden de absolute coördinaten die worden doorgegeven erg onnauwkeurig. De eigen positie wordt doorlopend bepaald met odometriegegevens die geleverd worden door de Motion Module. Als gevolg van wielslip en botsingen wordt deze informatie in de loop van de tijd onnauwkeurig. Om dit te corrigeren worden zelflokalisatie technieken gebruikt door de Vision Module. Zodra op grond van vast punten in het veld de positie kan worden bepaald, wordt deze positie aan de World Knowledge Module doorgegeven.

De World Knowledge Module maakt gebruik van het object Soccerworld. Dit is een wereldmodel waar alle elementen van een voetbalveld inzitten. Ten eerste zijn er vaste objecten, de fixed objects, waar de absolute positie van bekend is. Ten tweede zijn er de bewegende objecten, de movable objects. Hiervan wordt de positie opgeslagen, de snelheid en de oriëntatie (heading). Omdat er sprake is van een onzeker wereldbeeld, wordt van ieder movable object ook de onzekerheid berekend en opgeslagen. Het wereldmodel bevat ook alle methodes om informatie over objecten op te vragen en methodes om de parameters van de objecten te updaten. Dit wereldmodel is ook bruikbaar om te integreren in de simulator. Een gesimuleerde wereld moet deze gegevens immers ook bevatten.

2.3.4 Het laagste niveau

Op het laagste niveau, het niveau dat het dichtst op de robot staat, staan de modules die de sensoren en de actuatoren aansturen. De sensoren zijn de camera, die wordt aangestuurd door de Vision Module, en de odometriesensor, die wordt aangestuurd door de Odometrie Module. De actuatoren zijn de motoren, die de wielen van de robot laten draaien, aangestuurd door de Motion Module, en de kicker, om de tegen de bal te trappen, aangestuurd door de Kicker Module.

De ***Vision Module***[7] verwerkt de data die binnenkomen via de camera. De beelden worden geanalyseerd en objecten worden hierin herkend. Deze herkenning vindt plaats op basis van kleur. Gegevens over de positie van objecten binnen het gezichtsveld van de camera worden doorgegeven aan hogere modules. De Player Skills Module vraagt enkele malen per seconde naar de positie van een aantal objecten. De World Knowledge Module krijgt alle objecten binnen om een zo nauwkeurig mogelijk wereldbeeld te kunnen construeren. Ook speelt het interpreteren van camerabeelden een belangrijke rol bij zelflokalisatie. Als de robot voldoende vaste objecten waarneemt, kan hij op grond hiervan zijn positie berekenen. Nadat hij dit gedaan heeft, geeft hij zijn

berekende positie door aan de World Knowledge Module, die het wereldbeeld hiermee kan bijwerken.

De ***Odometrie Module*** verwerkt de gegevens die de odometriesensor genereert. Aan het begin van een wedstrijd wordt de robot op een bepaalde plaats neergezet. De odometrie sensor wordt van deze positie op de hoogte gebracht. Hierna wordt op grond van het aantal wielomwentelingen de positie van de robot bijgehouden.
Ervaring heeft geleerd dat odometriegegevens na verloop van tijd onnauwkeurig worden. Dit komt door wielslip en botsingen. Aangezien de Odometrie Module bijhoudt hoe groot de relatieve fout is, is de betrouwbaarheid van de odometrie positie bekend. Als de relatieve fout te groot wordt, moet de positie van de robot opnieuw bepaald worden. Zodra de robot zich in een dusdanige positie bevindt dat de Vision Module de positie kan berekenen op basis van vaste punten in het veld, worden de odometriegegevens hierop aangepast en is de onnauwkeurigheid ondervangen.

De **Motion Module** stuurt direct de motoren van de robot aan. De commando's tot bewegen die worden geformuleerd door de Player Skills Module, worden vertaald in motor commando's.

De ***Kicker Module*** stuurt het schopmechanisme aan. Als de Player Skills Module opdracht geeft tot het bewegen van het schopmechanisme, dan voert de Kicker Module deze uit.

2.3.5 Het Messaging systeem

Alle modules maken gebruik van een messaging systeem[8] om onderling informatie uit te wisselen. Iedere robot heeft een uniek robot ID wat wordt afgeleid van het IP adres van de PC en iedere module heeft een module ID. De combinatie van deze ID's maken een uniek getal waardoor iedere module van iedere robot als geadresseerde van een message kan worden gebruikt. Als er binnen een robot messages worden verstuurd, gebeurt dit binnen één PC en wordt het netwerk dus niet belast. Als een message wordt verstuurd naar een andere robot, wordt deze verstuurd over het wireless ethernet. Een message bestaat uit drie componenten: een geadresseerde, een type boodschap en de inhoud van de boodschap. Zo wordt aan de Motion Module een boodschap van het type setOdometer gegeven met als inhoud van de boodschap een x en een y coördinaat en een oriëntatie hoek om de odometrie sensor te resetten.
Grofweg zijn er drie soorten messages. Ten eerste een melding, de eerder beschreven message setOdoMeter bijvoorbeeld. Ten tweede is er een vraag–antwoord constructie en er is ook een subscription mogelijkheid.
De vraag–antwoord constructie verschilt met een melding in het feit dat een module na het stellen van een vraag zal wachten op een antwoord. De PLS Module vraagt aan de VIS Module bijvoorbeeld om de positie van de bal. De VIS Module zal op het moment dat deze vraag behandeld wordt direct een antwoord geven. Tot die tijd wacht de PLS Module op een antwoord. Als dit antwoord niet komt, ontstaat een deadlock. Dit is het nadeel van dit type boodschap.
Om niet diverse malen per seconde dezelfde vraag aan dezelfde module te stellen en om de mogelijkheid van een deadlock te voorkomen, is de mogelijkheid gecreëerd om te abonneren op gegevens. Zo geeft de WRL Module bij het opstarten door aan de VIS

Module dat hij vijf keer per seconde gegevens wil hebben over de objecten die door de VIS Module worden herkend. De VIS Module behandelt deze “subscription” en stuurt vervolgens vijf keer per seconde gegevens naar de WRL Module zonder dat hier specifiek om gevraagd hoeft te worden. De WRL Module wacht dus niet specifiek op antwoord en verwerkt deze gegevens op het moment dat het uitkomt.

3. De opzet van de simulator

3.1 Inleiding

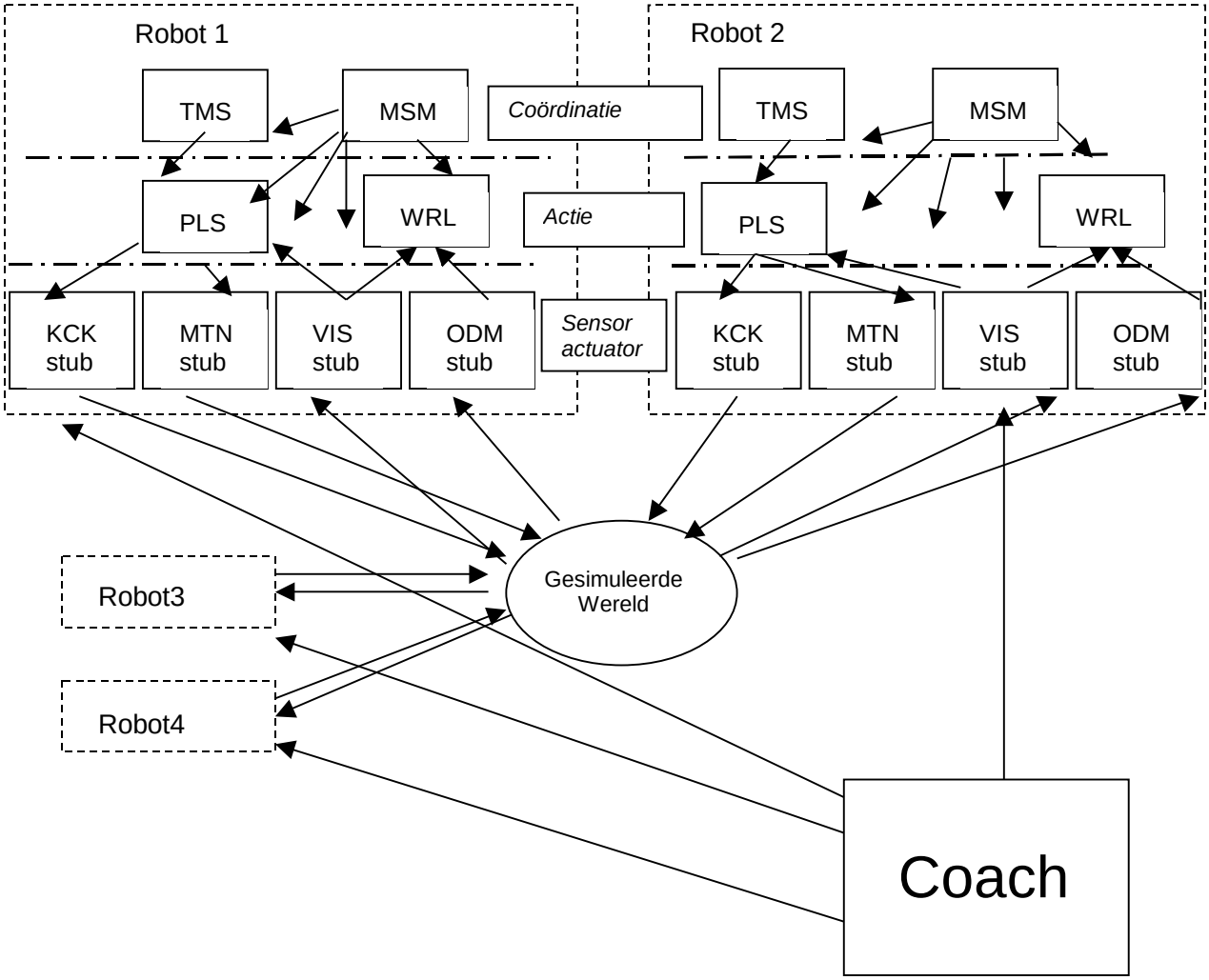
De software architectuur van Clockwork Orange is als uitgangspunt genomen bij het ontwerpen van de simulator, aangezien de bestaande modules met de simulator moeten samenwerken. Alle modules kunnen worden opgestart op dezelfde manier in dezelfde omgeving door de Mission Manager. Dezelfde communicatie module wordt gebruikt en dezelfde coach om het team aan te sturen.

Zoals beschreven is in hoofdstuk 2 en te zien op afbeelding 2.1 zijn er vier modules die rechtstreeks met de wereld communiceren, de Motion Module, de Kick Module, de Vision Module en de Odometrie Module. Bij een gesimuleerde wereld moeten deze modules dus worden vervangen door virtuele modules. Voor de Motion Module geldt dat er geen motoren zijn om commando's aan te geven. In plaats hiervan moeten deze commando's worden afgevangen om de positie van de virtuele robot te updaten. Ook zond de Motion Module gegevens uit over de odometrie. Deze gegevens moeten ook door het nieuwe stukje software worden verstuurd. De Odometrie Module ontvangt deze gegevens en kan dus ongewijzigd gebruikt worden. De Kicker Module heeft geen kicker om aan te sturen. De kick commando's moeten dus worden afgevangen om de positie van de virtuele bal te beïnvloeden. Voor de Vision Module geldt dat er geen wereld is om waar te nemen. De gegevens die de Vision Module genereert op basis van camera beelden moeten nu worden gegenereerd op basis van de gegevens die opgeslagen zijn in het wereldmodel van de simulator. Deze drie herschreven modules worden ***stubs*** genoemd.

Als deze stubs de echte robot modules vervangen, dan is het dus mogelijk om een virtuele robot op te starten. Deze virtuele robot moet zich gaan bewegen in een virtuele wereld. Deze virtuele wereld vormt het hart van de simulator. De simulator bestaat uit een wereldmodel, waarin zich ook de virtuele bal bevindt, een wereld update gedeelte en een message gedeelte. Op het moment dat een virtuele robot wordt opgestart, meldt hij zich aan in het wereldmodel, die door de simulator wordt gecreëerd en bijgehouden. In het wereldmodel wordt van alle objecten die zich hebben ingeschreven de positie, snelheid en oriëntatie opgeslagen. De positie van alle virtuele spelers en de virtuele bal worden daar bijgehouden. Ook de positie van fixed objects is opgeslagen in het wereldmodel. Op deze wijze is het model geschikt ter vervanging van een echte wereld. Aangezien vision gegevens eens per 200 ms naar de World Knowledge Module moeten worden gestuurd, moet minstens eens per 200 ms de positie van alle bewegende objecten worden berekend. Op grond van snelheid, oriëntatie en hun vorige positie kan de huidige positie, snelheid en oriëntatie worden berekend.
Zoals de "echte" wereld in afbeelding 2.1 centraal staat, zo moet de virtuele wereld in de simulator de spin in het web worden in de virtuele omgeving. Dit betekent dat de messages die door de drie laagste modules verstuurd worden, moeten worden afgehandeld. Als de Kick Module een boodschap stuurt, moet de bal in de wereld worden aangepast. De Motion Module stuurt gegevens over de verplaatsing van de

robot. Deze moeten ook in de virtuele wereld worden verwerkt. Tevens moeten voor deze module odometrie gegevens worden gegenereerd als deze er om vraagt. De Vision Module geeft het door als een module zich abonneert op vision data. De simulator genereert deze data en geeft deze terug.

3.2 De Simulator



Figuur 3.1 De opzet van de simulator

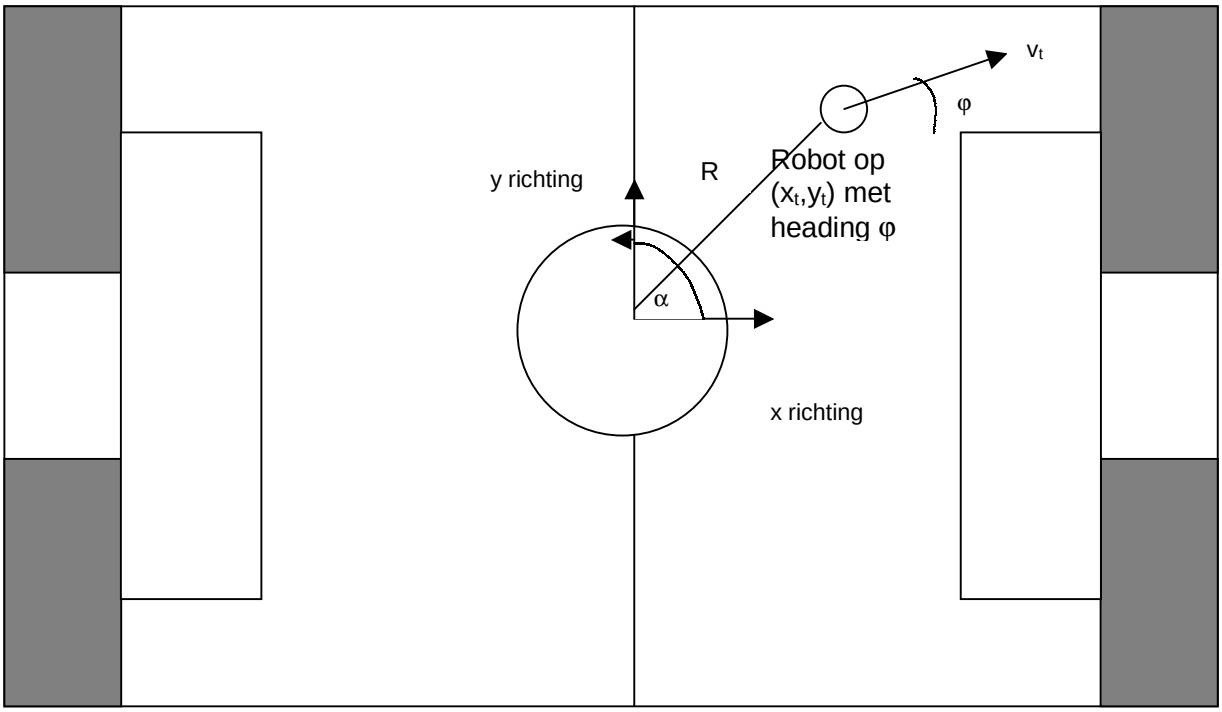
3.2.1 De Simulator, het wereldmodel

De Simulator biedt een wereldmodel aan alle virtuele robots. Behalve dat op software niveau de simulator zodanige oplossingen moet bieden dat alle andere modules zonder

probleem of merkbaar verschil draaien, moeten de omstandigheden die van toepassing zijn in een echte wereld door de simulator worden gegenereerd. Een virtuele robot beweegt zich door de gesimuleerde wereld, doordat de gesimuleerde wereld zijn baan berekent. Omdat ook alle andere virtuele robots en overige objecten uit een voetbalveld zich in de gesimuleerde wereld bevinden, heeft de gesimuleerde wereld alle gegevens om data te genereren die een robot normaal met de Vision Module zou genereren.
In eerste instantie is de simulator dus in staat om alle motor commando's af te vangen en deze perfect te vertalen in verplaatsingen in de virtuele wereld en om data die uit de virtuele wereld komen over andere objecten foutloos te vertalen naar gesimuleerde vision gegevens. Eén van de grootste problemen van het robosoccer is dat deze situatie niet strookt met de realiteit: in de praktijk blijkt dat de bewegingen van de robot in de wereld niet perfect zijn. De wielen van een robot blijken te spinnen. Ook kan een robot botsen tegen een tegenstander of tegen de boarding. Dit betekent dat rekening moet worden gehouden met een afwijking in de odometrie gegevens. De werkelijke positie van een robot verschilt dus met de positie die de odometrie gegevens aangeven. Aan de ene kant houdt de simulator dus de exacte positie bij van iedere virtuele robot, in aanmerking genomen dat er sprake is van wielspin, aan de andere kant moeten van iedere virtuele robot de odometrie gegevens worden bijgehouden om deze te communiceren naar modules die hier om vragen. De exacte positie wordt gebruikt om het zicht van een virtuele robot te simuleren. Hoe langer een robot rijdt, des te groter wordt de afwijking in het bepalen van de positie op grond van odometrie gegevens. Proeven met een rijdende robot hebben aangetoond dat de wielspin ongeveer 3% afwijking tot gevolg heeft in voorwaartse richting. Hier komt nog bij dat indien een robot zodanig slipt dat hij van zijn rechte lijn afwijkt, er ook een fout in zijwaartse richting optreedt. Bij iedere positie update van een virtuele robot wordt hier rekening mee gehouden en wordt een slipfout toegevoegd.
Dit is geïmplementeerd doordat, wanneer de positie van een robot wordt berekend, de afgelegde weg met een slip factor wordt verminderd.
In de echte wereld wordt door de Vision Module, wanneer mogelijk, de positie bepaald. Dit gebeurt op grond van vaste objecten in het veld die worden herkend door de camera. Door de positie ten opzichte van deze vaste objecten te berekenen, kan de absolute positie in het veld worden berekend. De simulator genereert ook zo'n zelf lokalisatie. De odometrie gegevens worden dan gelijk gesteld aan de exacte positie.
Ook blijkt dat in de praktijk dat de data die gegenereerd worden door de camera, over de positie van de andere robots in het veld en de positie van de bal, een zekere onnauwkeurigheid kunnen bevatten. De simulator bevat ook de mogelijkheid om ruis toe te voegen aan de vision gegevens uit de gesimuleerde wereld.

3.2.2.1 Update van het wereldmodel, berekenen van de positie van robots

Eens in de 200 ms wordt het wereldbeeld bijgewerkt. Van alle bewegende objecten moet opnieuw de positie en oriëntatie worden berekend. Om te beginnen wordt de positie van alle robots berekend. De Player Skills Module stuurt aan de Motion Stub met welke lineaire snelheid en hoeksnelheid de robot moet bewegen. De simulator krijgt deze snelheden van de Motion Stub. Met deze gegevens kan de simulator de positie van de virtuele robot berekenen. Het coördinatenstelsel zoals gebruikt wordt om posities en oriëntaties op het veld aan te geven, is gegeven in figuur 3.1. Het centrum van het coördinatenstelsel is de middenstip van het voetbalveld.



figuur 3.2
Snelheid wordt gegeven in mm/s
De hoek wordt aangegeven in 1/10 graden

Het berekenen van een positie na een interval Δt gaat als volgt:

De coördinaten van de robot zijn bekend op tijdstip t. De lineaire snelheid v_t en de hoeksnelheid Ω_t zijn bekend op tijdstip t. R is de straal van de beweging. f is de gemiddelde slipfactor. In werkelijkheid is de slipfactor stochastisch, maar vanwege het hoge aantal updates per seconde correleert deze naar het gemiddelde. b is de bias, ten gevolge van een verschillende slipfactor links en rechts. Bij de berekening wordt uitgegaan van een lineair versnelde of vertraagde beweging, wat tot de volgende formules leidt.

1) *Berekenen effectieve snelheden*
   $v_{eff} = (v_t + v_{t+\Delta t})/2$
   $\Omega_{eff} = (\Omega_t + \Omega_{t+\Delta t})/2$

2) *Berekenen nieuwe oriëntatie φ_t*
   $\Delta\varphi = \Omega_{eff} * \Delta t$
   $\varphi_{t+\Delta t} = |\varphi_t + \Delta\varphi|$ *mod 3600*

3) *Berekenen afgelegde weg in egocentrische coördinaten*
   $R = v_{eff} / \Omega_{eff}$

$\Delta xr = |R*\sin(\Delta\varphi)| * sign(v_{eff}) * f$
$\Delta yr = |R-R*\cos(\Delta\varphi)|*sign(v_{eff})*sign(\Omega_{eff}) * f + b * \Delta xr$
*Als Ω_{eff} =0 dan geldt $\Delta yr=0$ en $\Delta xr = v_{eff} * \Delta t$
**Als v_{eff} =0 dan geldt $\Delta xr =0$ en $\Delta yr =0$
***f en b dienen voor de verschillende echte robots te worden geschat.

4) *Omrekenen robot relatief coördinaten systeem naar wereldrelatief coördinaten systeem*
$x_{t+\Delta t} = x_t + \Delta xr *\cos(\varphi_t) - \Delta yr *\sin(\varphi_t)$
$y_{t+\Delta t} = y_t + \Delta xr *\sin(\varphi_t) + \Delta yr *\cos(\varphi_t)$

Het kan voorkomen dat een robot de opdracht krijgt om door een andere robot of door de boarding heen te rijden als gevolg van een storing aan een van de modules. In dit geval zal de robot niet bewegen en wachten tot hij door de coach op de middenstip wordt gezet.

3.2.2.2 Berekenen van de positie van de bal

Ook de positie van de virtuele bal moet door de simulator bijgehouden worden. De bal heeft alleen een lineaire snelheid - de bal rolt immers alleen rechtdoor - en een oriëntatie, de richting waarin de bal rolt. De bal is continu aan vertraging van 100 mm/s^2 onderhevig, totdat de bal stil ligt. Als de bal geen obstakels tegenkomt, wordt de positie op tijdstip t+Δt als volgt berekend:

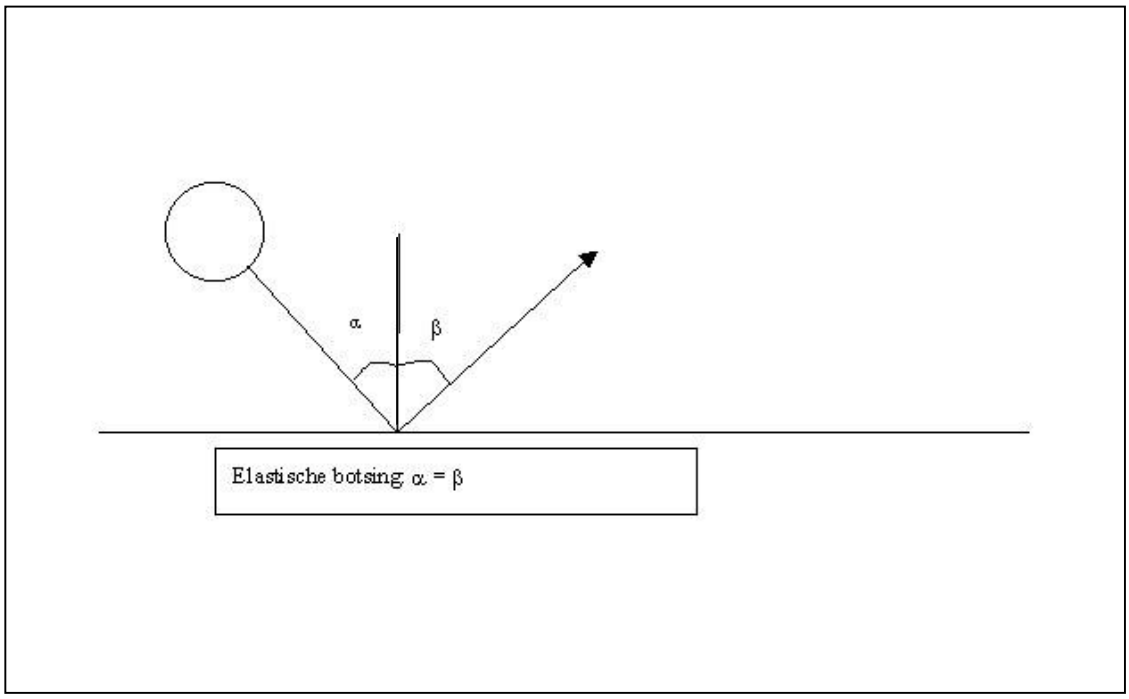
1) *Bereken de effectieve snelheid*
$v_t = at + v_0$
$v_{t+\Delta t} = v_t + a\Delta t$
$v_{eff} = v_t + ½\ a\Delta t$
* a=0 als v=0

2) *Bereken afgelegde weg in egocentrische coördinaten*
$\Delta x_b = v_t\Delta t + ½\ a\Delta t^2 = v_{eff}\ \Delta t$

3) *Bereken de absolute positie*
$x_{t+\Delta t} = x_t + \Delta x_b *\cos(\varphi_t)$
$y_{t+\Delta t} = y_t + \Delta x_b *\sin(\varphi_t)$

De bal is continu onderhevig aan een vertraging a van -100 mm/s^2

De bal kan obstakels tegenkomen die de snelheid en richting van de bal beïnvloeden. Het simpelste geval is als de bal tegen de boarding aanrolt. De bal botst dan met de boarding met de eigenschappen van een elastische botsing.



Figuur 3.3 Een elastische botsing tussen de bal en de muur

Er zijn 4 gevallen mogelijk waarin de bal botst met de boarding. Deze vertalen zich in posities van de bal buiten het veld. De x coördinaat van de bal kan te groot worden, de y coördinaat kan te groot worden, of de x of y coördinaat kan te klein worden. Als de bal een baan heeft gevolgd die hem buiten de boarding zou hebben gebracht, dan wordt de positie van de bal zodanig aangepast, dat de bal spiegelt in de boarding. Uiteraard moet de straal van de bal in ogenschouw genomen worden: de positie van de bal vermeerderd met de straal moet altijd binnen de boarding zijn. De heading van de bal wordt ook opnieuw berekend:

Geval 1: De x coördinaat wordt te groot, de bal botst met de boarding

$x_t + R_{bal} > x_{max}$

1) *Hoeveel heeft de bal de boarding overschreden*

$\Delta x = x_t + R_{bal} - x_{max}$

2) *Pas x_t aan*

$x_t = x_{max} - \Delta x$

3) *Bereken nieuwe heading van de bal*

$\varphi_t = |1800 - \varphi_{t-\Delta x}| \ mod\ 3600$

Geval 2: De y coördinaat wordt te groot, de bal botst met de boarding

$y_t + R_{bal} > y_{max}$

1) *Hoeveel heeft de bal de boarding overschreden*

$\Delta y = y_t + R_{bal} - y_{max}$

2) *Pas y_t aan*
$y_t = y_t - \Delta y$

3) *Bereken nieuwe heading van de bal*
$\varphi_t = -\varphi_{t-\Delta x}$

Geval 3: De x coördinaat wordt te klein, de bal botst met de boarding
$x_t - R_{bal} < x_{max}$

1) *Hoeveel heeft de bal de boarding overschreden*
$\Delta x = x_t - R_{bal} - x_{max}$

2) *Pas x_t aan*
$x_t = x_{max} - \Delta x$

3) *Bereken nieuwe heading van de bal*
$\varphi_t = |1800 - \varphi_{t-\Delta x}| \ mod\ 3600$

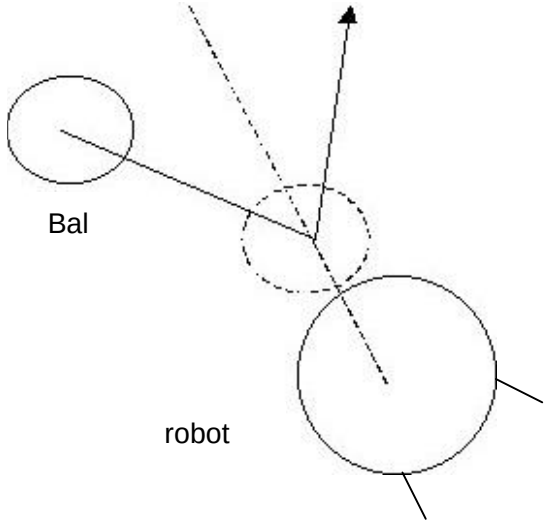
Geval 4: de y coördinaat wordt te klein, de bal botst met de boarding
$y_t - R_{bal} < y_{max}$

1) *Hoeveel heeft de bal de boarding overschreden*
$\Delta y = y_t + R_{bal} - y_{max}$

2) *Pas y_t aan*
$y_t = y_{max} - \Delta y$

3) *Bereken nieuwe heading van de bal*
$\varphi_t = -\varphi_{t-\Delta x}$

Een ander obstakel dat de richting van de bal beïnvloedt, is een robot. Als de bal tegen de achterkant of zijkant van een robot aankomt, zal de baan van de bal veranderen. Indien dit het geval is, wordt uitgegaan van een elastische botsing zoals gevisualiseerd op figuur 3.2, waarbij de robot wordt voorgesteld als een ronde muur.



figuur 3.2 Een botsing tussen de bal buiten de bak en een robot

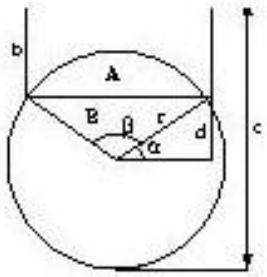
Bereken de plaats waar de bal de robot raakt. Hier wordt doorlopend door de simulator op getest. De afstand tussen de bal en de robot is dan gelijk aan de som van de stralen (R) van de bal en de robot.

$$\sqrt{(x_{bal} - x_{robot})^2 + (y_{bal} - y_{robot})^2} = R_{robot} + R_{bal}$$

1) *Als de bal de robot raakt, wordt de relatieve hoek tussen de richting van de bal en de robot op die plaats berekend*
   $\alpha = atan((x_{robot} - x_{bal})/(y_{robot} - y_{bal}))$

2) *De nieuwe richting van de bal is te berekenen*
   $\varphi_t = |\varphi_{t-\Delta x} - 1800 - 2 * \alpha|_{mod\ 3600}$

Een bijzondere botsing met een robot vindt plaats als de bal de robot aan de voorkant botst. De bal komt dan in de bak aan de voorkant van de robot terecht (figuur 3.4). De robot absorbeert dan de snelheid van de bal. De bal blijft dan in de bak liggen en neemt de snelheid en de richting van de robot aan. Als de bal de linker of rechter zijkant van de bak raakt, dan zal dit de richting van de bal beïnvloeden: de bal zal de richting van de robot aannemen verminderd of vermeerderd met 0.3 radialen (D_{bal}). De robot kan op deze wijze met de bal draaien.



Figuur 3.4 Schematische weergave van de robot en de bak van het schopmechanisme.

R	*20.85 cm*
A	*35 cm*
B	*21 cm*
C	*52 cm*

De laatste factor die de beweging van de bal beïnvloedt, is het schop mechanisme. Als de Simulator een kick message binnenkrijgt van een virtuele robot, wordt eerst gecontroleerd of de bal zich in de bak van de robot bevindt. Als dit het geval is, zal de bal een snelheid K_{bal} aannemen van 1500mm/s.

In alle gevallen van bal beweging geldt dat de snelheid van de bal lineair afneemt met een vertraging a van -100mm/s^2.

3.2.2.3 Het simuleren van de vision

De simulator simuleert ook de gegevens die normaal door een camera worden geleverd. Aangezien alle informatie over de positie van objecten beschikbaar is door het wereldmodel, kan de simulator gegevens over objecten binnen het gezichtsveld, gegevens die in de real world door de camera zouden worden geregistreerd, doorsturen naar de Vision Stub. Om de software in de toekomst aan nog meer realistische tests te onderwerpen, zou gedacht kunnen worden aan het toevoegen van een onzekerheid aan de beelden die gesimuleerd worden (ε_x en ε_y). Tot nu toe is daar vanaf gezien. De vision gegevens bevatten al onnauwkeurigheid, doordat de eigen positie onzeker is. De ruis die aan de vision zou kunnen worden toegevoegd zou afstandsafhankelijk moeten zijn. Hoe verder een object van de camera verwijderd is, des te groter zou de afwijking in de data moeten zijn.

Stel xr en yr zijn de robot relatieve coördinaten van een waargenomen object. Ruis wordt dan toegevoegd als volgt:

$$x_{ruis} = xr + \sqrt{xr^2 + yr^2} \cdot \varepsilon_x$$

$$y_{ruis} = yr + \sqrt{xr^2 + yr^2} \cdot \varepsilon_y$$

3.2.3 Het afhandelen van de messages

De simulator heeft ook een belangrijke functie binnen de communicatie tussen de verschillende modules. Alle informatie die normaal gesproken met de wereld werd uitgewisseld, moet nu door de simulator gegenereerd worden en afgehandeld. De stubs krijgen verzoeken van andere modules om informatie: zo vraagt bijvoorbeeld de Odometrie Module om odometrie gegevens of de World Knowledge Module om vision gegevens. De messages die deze verzoeken bevatten, worden doorgestuurd naar de simulator. De simulator haalt de benodigde gegevens uit het wereldmodel en stuurt de benodigde informatie terug. Ook geven stubs opdracht de gesimuleerde wereld te wijzigen: de Motion Stub kan het wereld model de opdracht geven om de positie van een robot aan te passen. De message met deze opdracht wordt door de Player Skills Module aan de Motion Stub gestuurd. Deze stuurt hem door aan de simulator.
Als een virtuele robot wordt opgestart gaan de stubs op zoek naar het adres van de computer waar de simulator op draait. Op het moment dat een Motion Stub wordt gedetecteerd op een andere computer, wordt een nieuwe virtuele robot ingevoegd in de virtuele wereld. Vanaf dit moment wordt de virtuele robot en zijn computer adres in de database opgenomen en kunnen er messages naar verstuurd worden. Ook wordt het opstarten van Kick Stubs en Motion Stubs gedetecteerd. De simulator stuurt dan een hand-shake terug, zodat de stubs op de hoogte zijn van het adres van de computer waar de simulator op draait.

3.3 De Motion Stub

De Motion Module heeft twee functies. Ten eerste moeten de messages van de Player Skills Module ontvangen worden om de motoren aan te sturen en ten tweede moeten de odometrie gegevens aan de ODM Module geleverd worden. Door de Player Skills Module wordt doorgegeven welke lineaire snelheid en welke hoeksnelheid de robot moet krijgen. De Motion Stub moet dus deze gegevens doorgeven aan de simulator die de positie van de virtuele robot hiermee berekent. Diverse andere modules kunnen aan de Motion Module vragen wat de positie van een robot is op grond van odometrie gegevens. De Motion Stub vraagt dit aan de simulator. De simulator houdt deze gegevens bij in het wereldmodel en kan zo aan iedere module deze gegevens verstrekken. Om goed te kunnen samenwerken met de andere modules moeten alle stubs ook een stukje communicatie met de Mission Manager onderhouden. Bij het opstarten van de modules moet de staat van opstarten aan de Mission Manager worden doorgegeven, zodat deze dit kan controleren. Gedurende het draaien van de stub moet ook steeds een teken van leven aan de Mission Manager worden gegeven. Dit gebeurt door een heartbeat signaal uit te zenden. Als de Mission Manager geen heartbeat signaal meer ontvangt, gaat de Mission Manager er van uit dat de module niet meer draait en wordt deze opnieuw opgestart. Bij het opstarten van de Motion Stub wordt een broadcast over het netwerk gestuurd om de simulator te zoeken. Zodra de simulator dit bericht binnen krijgt, reageert hij door een hand-shake te sturen naar de Motion Stub. Uit dit hand-shake signaal kan de Motion Stub opmaken op welke computer de simulator zich bevindt. Dit adres wordt opgeslagen. Van dit adres kan dan gebruik worden gemaakt bij het versturen van messages aan de simulator.

3.4 De Vision Stub

De Vision Module interpreteert de camerabeelden en geeft hierover gegevens aan de World Knowledge Module. De Vision Stub moet dus simuleren objecten uit camerabeelden herkend te hebben. De informatie over de objecten krijgt de Vision Stub van de simulator. Uit het wereldbeeld bepaalt de simulator welke objecten binnen de invalshoek van de camera van een robot zouden vallen. De simulator rekent de absolute coördinaten om naar egocentrische coördinaten. De Vision Stub ontvangt deze gegevens en stuurt ze door naar de World Knowledge Module.
De Player Skills Module vraagt ook wel eens direct aan de Vision Module de positie van de bal om onnauwkeurigheid als gevolg van tijdsvertraging te verkleinen. De Vision Stub moet dus ook deze functie simuleren. De positie van de bal wordt aan de simulator gevraagd en doorgegeven. Vragen van de Player Skills Module en de Team Skills Module over de positie van de goals en de positie van robots worden op dezelfde manier behandeld.
Een belangrijke functionaliteit die de Vision Module ook nog heeft, is zelflokalisatie. Zodra de Vision Module op grond van fixed objects in het veld de positie van de robot uit kan rekenen, geeft hij de correcte positie door aan de WRL Module. Om dit te ondervangen, stuurt de Vision Stub eens in de 10 seconden de correcte positie door naar de World Knowledge Module.

Net zoals de Motion Stub moet ook de Vision Stub communiceren met de Mission Manager. Dit gebeurt op dezelfde manier: het moet een signaal afgeven bij het opstarten en een heartbeat signaal afgeven zodat de Mission Manager op de hoogte is van het feit dat de module draait.
Na het opstarten van de Vision Stub gaat deze op zoek naar de simulator door een broadcast over het netwerk te sturen. Na het ontvangen van de hand-shake van de simulator wordt het adres van de computer van de simulator opgeslagen, zodat dit later gebruikt kan worden om messages naar te sturen.

3.5 De Kick Stub

De Kick Stub moet de functionaliteit van de Kick Module overnemen. De Kick Module stuurt de kicker van de robot aan. Deze kicker heeft invloed op de baan en de snelheid van de bal. De Kick Stub moet dus de messages die de PLS stuurt over het kicken van de bal afvangen en aan de simulator doorgeven dat de baan en snelheid van de bal moet worden aangepast. Als de Kick Stub een Kick commando krijgt dan zal de bal een snelheid (K_{bal})krijgen van 1000 mm/s.
Om te functioneren in het geheel gaat ook de Kick Stub op zoek naar de simulator bij het opstarten en gebruikt hij het antwoord van de simulator om deze later messages te kunnen sturen.
Ook worden de benodigde signalen aan de Mission Manager gegeven om het geheel te laten draaien.

3.6 Samenvatting

De simulator bevat een aantal essentiële parameters die in dit hoofdstuk zijn beschreven en die de werkelijkheid zo goed mogelijk dienen te benaderen. Voor deze parameters zijn in eerste instantie de waarden van tabel 3.1 gekozen. Bepaalde variabelen zijn robot afhankelijk, zoals f en b. Andere zijn robotontwerp afhankelijk, zoals K_{bal} en D_{bal}. Als een andere kicker of een andere bak gebruikt wordt
veranderen deze variabelen. De vision onzekerheid is afhankelijk van de kwaliteit van de camera en de kwaliteit van de Vision Module. De ondergrond heeft invloed op de balvertraging a. In hoofdstuk 5 zal worden teruggekomen op het bepalen van deze parameters. Om de simulator anders in te stellen, kunnen deze parameters worden gewijzigd.

Parameter	**Symbool**	**Waarde**
Slipfactor	f	0.99
Slip bias	B	0
Bal vertraging	a	100 mm/s^2
Vision onzekerheid X	εx	0
Vision onzekerheid y	εy	0
Frequentie Zelflokalisatie	Z_f	1/10
Kicksnelheid bal	K_{bal}	1500 mm/s
Draai snelheid bal	D_{bal}	0.3 radialen

Tabel 3.1 waarden van verschillende simulator parameters

4. De implementatie van de Simulator

4.1 Inleiding

Voordat de implementatie zijn aanvang kon vinden, moesten een aantal aspecten van de software uitgebreid bestudeerd worden. Aangezien alle communicatie tussen de verschillende modules via een ingewikkeld en weinig gebruikersvriendelijk message systeem verloopt, moest ook de simulator dit message systeem ondersteunen. Hiernaast moesten alle modules die met de te schrijven stubs communiceerden doorgelicht worden: de stubs moesten immers naadloos in het geheel passen. Meer specifiek houdt dit in dat alle modules van het middelste niveau van de architectuur zoals beschreven in figuur 2.1 in de gesimuleerde omgeving moesten communiceren met de stubs. Alle messages die met behulp van het message systeem naar een module op het laagste niveau gestuurd worden, moesten dus door een stub worden afgevangen en afgehandeld.

Waar in de real world omgeving de modules van het laagste niveau informatie uit de wereld halen met sensoren en de wereld met actuatoren modificeren, moeten in een gesimuleerde omgeving de stubs communiceren met de gesimuleerde wereld. De virtuele sensor stubs halen informatie uit de gesimuleerde wereld en de virtuele actuator stubs veranderen de gesimuleerde wereld. Communicatie tussen de stubs en de gesimuleerde wereld gebeurt met hetzelfde message systeem als waarmee de communicatie tussen de andere modules geschiedt.

Wat ook nog moest worden gerealiseerd is dat de stubs aangestuurd kunnen worden door de Mission Manager. Dit houdt in dat de stubs opgestart kunnen worden door de Mission Manager en hier de Mission Manager van op de hoogte brengen. Ook moeten de stubs continu aan de Mission Manager een teken van leven, een heartbeat, sturen, zodat kan worden geanticipeerd als één van de stubs is gecrashed.

Verreweg de meeste tijd van het tot stand brengen van de simulator is gaan zitten in de implementatie.
De opstart fase waarin alle bestaande software bestudeerd moest worden en alle communicatiestromen tussen de verschillende modules in kaart moest worden gebracht, was zeer tijdrovend. Ten eerste kwam dit door gebrekkige documentatie over de al bestaande modules, ten tweede door het feit dat er druk aan de modules werd gewerkt. Door het ontbreken van volledige documentatie moesten duizenden regels code worden doorgelopen om de software te kunnen begrijpen. Als de software eenmaal in kaart was gebracht, kon het zo maar, dat deze de dag hierna veranderd was.
De fase waarin de software werd geschreven, was ook langdurig. Om iedere stub dezelfde in en output te laten genereren als de module waar deze naar gemodelleerd was, bleek een zeer omvangrijk karwei. Ook het schrijven van de simulator die met al deze stubs moest communiceren was een grote klus. Het feit dat de simulator de spin in

het web moest worden tussen continu veranderende modules, maakte dat de werking van de simulator onzeker was in de tijd dat er nog ontwikkeld werd aan de robots.

4.2 Algemene opmerkingen en problemen bij de implementatie

Aan het begin van het project werd de simulator voorgesteld als volgt: één computer waar een virtueel wereldmodel op draait en acht virtuele robots die ook op die computer draaien. Het wereldbeeld zou gevisualiseerd worden en er zou gemakkelijk waargenomen kunnen worden hoe de virtuele robots zich zouden gedragen in de virtuele wereld. Bovendien zou er dan geen uitgebreide infrastructuur nodig zijn om de gesimuleerde voetbalwedstrijd te spelen. Het message systeem zoals dat wordt gebruikt door alle bestaande modules bleek hiertoe echter niet toereikend. Het was geïmplementeerd met het idee dat een robot een aparte computer is. Communiceren tussen robots werd dus gezien als communiceren tussen verschillende computers. Iedere robot moest een uniek ID hebben om als adres voor een message te kunnen dienen. Dit unieke ID bleek gekoppeld aan het IP adres. Dit heeft ervoor gezorgd dat er per computer maar één robot of virtuele robot kan draaien. Hierdoor moest de keuze gemaakt worden om iedere virtuele robot op een aparte computer te draaien, zodat zij allemaal een verschillend adres zouden hebben. Dit heeft tot gevolg dat er voor een wedstrijd tussen twee teams van vier robots acht computers nodig zijn. Het gesimuleerde wereldbeeld draait dan op één van de acht computers, waar ook al een robot op draait. Dit is geen probleem: er is maar één gesimuleerd wereldbeeld, deze heeft dus altijd een uniek ID. Behalve dat deze omstandigheid ervoor zorgt dat je een behoorlijke infrastructuur nodig hebt, namelijk een netwerk met minstens acht computers, heeft deze ook tot gevolg dat er enorm veel verkeer over het ethernet plaatsvindt. Waar in het geval van een echte robot het grootste gedeelte van de messages plaatsvinden binnen een robot zelf en dus binnen één computer - denk hierbij aan het opvragen van de eigen positie en het opvragen van objecten uit de Vision Module of het opvragen van de eigen positie aan de Odometrie Module wat enkele keren per seconde gebeurt - vindt al deze communicatie bij een simulatie over het netwerk plaats. Dit veroorzaakt enorm veel netwerkverkeer. Hier komt nog eens bij dat het grootste gedeelte van de communicatie plaatsvindt met het systeem waar de simulator op draait. Alle informatie over posities van objecten wordt op dit systeem bijgehouden en vragen hierover moeten voor alle acht virtuele robots worden behandeld. Dit heeft ervoor gezorgd, dat het in de praktijk niet mogelijk bleek om met acht robots op de simulator in te loggen. Teveel vragen werden dan niet of te laat behandeld, waardoor de message queues vol raakten en diverse modules vastliepen.
Ofschoon de implementatie van de simulator wel geschikt is om een echte wedstrijd te kunnen spelen met twee teams van vier spelers, kan van deze functionaliteit pas in de toekomst gebruik worden gemaakt. Er moet dan een ander message systeem worden gebruikt, dat economischer met de netwerkcapaciteit omgaat.

4.3 De Simulator Module

4.3.1 Inleiding

De basis van de Simulator Module bestaat uit een wereldmodel. In dit wereldmodel worden de gegevens van de bewegende objecten bijgehouden. Als de softwaremodules van een virtuele robot worden opgestart, dan logt een virtuele robot in op dit wereldmodel. Een virtuele robot krijgt van de coach een plaats toegewezen, die hij krijgt in het wereldmodel. Van een robot worden een aantal dingen opgeslagen in dit model. Ten eerste uiteraard de positie: zijn x coördinaat, y coördinaat en oriëntatie. Zo vaak als de rekenkracht van het systeem het toelaat, wordt de positie van de robot opnieuw berekend zodat het wereldbeeld zo up-to-date mogelijk blijft. In deze positie wordt een ruis toegevoegd die het gevolg is van een aantal factoren zoals slip of wielspin. Deze positie verschilt dus van de op odometrie gebaseerde positie.

Daarom houdt het wereldmodel ook van iedere robot de op odometrie gebaseerde positie bij. Deze kennis moet paraat zijn om vragen om odometriegegevens van bijvoorbeeld de vision module te kunnen beantwoorden. Op grond van de echte positie wordt de relatieve positie ten opzichte van andere bewegende objecten bepaald, die wordt gebruikt bij het doorgeven van vision gegevens aan andere modules. Verder wordt van iedere robot bijgehouden wat zijn lineaire snelheid is en wat zijn hoeksnelheid. Deze positiegegevens en de snelheidsgegevens zijn dus alle eigenschappen van een virtuele robot. Verder bestaat de simulator Module net als alle andere modules uit een receiver loop. Dit is een continue draaiend proces wat luistert naar het message systeem. Door de stubs worden messages gestuurd waar de simulator naar moet handelen. Grofweg bestaan de messages uit informatieverzoeken en uit te voeren acties.

4.3.2 Messages die de simulator afhandelt

MSM gerelateerde messages

- simStop

  De message simStop wordt verstuurd door de Mission Manager. Als deze boodschap wordt ontvangen dan stopt de simulator.

- simPauze

  De message simPauze wordt verstuurd door de Mission Manager. Als deze boodschap wordt ontvangen dan pauzeert de simulator.

- simContinue

  De message simContinue wordt verstuurd door de Mission Manager. Als deze boodschap wordt ontvangen dan continueert de simulator

Motion gerelateerde messages

- simMtnStubIsHere

De message simMtnStubIsHere wordt verstuurd door een Motion Stub. Als deze boodschap wordt ontvangen dan weet de simulator dat ergens op het netwerk een Motion Stub is opgestart. De simulator reageert door een message terug te sturen aan de Motion Stub om zijn IP adres bekend te maken. Ook voegt de simulator na het ontvangen van een simMtnStubIsHere een virtuele robot toe aan de gesimuleerde wereld. Na het ontvangen van deze message beschouwt de simulator de virtuele robot als actief. Vanaf dit moment is de robot voor andere robots "zichtbaar" in de virtuele wereld.

- simGetOdometer

De vraag simGetOdometer wordt verstuurd door een Motion Stub. De simulator antwoordt met de odometrie gegevens van de desbetreffende virtuele robot.

- simSetOdometer

De message simSetOdometer wordt verstuurd door een Motion Stub. De odometrie gegevens van de virtuele robot worden gereset op de nieuwe data.

- simSetOdometerandPosition

Deze message is toegevoegd om de coach de mogelijkheid te geven de virtuele robots op hun plek te zetten. Deze message bevat de nieuwe positie waar de coach de virtuele robot neerzet. Als deze message binnenkomt, worden de odometrie gegevens en de positie gereset.

- simMtnMoveContinuous

Deze message wordt verstuurd door een Motion Stub. Deze boodschap bevat gegevens over de snelheid waarmee de virtuele robot door de gesimuleerde wereld beweegt.

- simMotionStubIsDead

Als de Motion Stub de opdracht krijgt van de Mission Manager om af te sluiten dan meldt de Motion Stub dit op deze wijze aan de simulator. De simulator zet de snelheid van de virtuele robot op nul, aangezien ook in de real world de robot stil zou staan na het afsluiten van de Motion Module. De positie van de robot wordt onthouden, zodat nadat de modules weer opnieuw zijn opgestart, de robot weer verder kan gaan vanaf de laatst bekende positie.

Vision gerelateerde messages

- simVisSetShapeSubscription

De message simVisSetShapeSubscription wordt verstuurd door een Vision Stub. De Vision Stub reageert hiermee op een abonnement verzoek van de World Knowledge Module om vision gegevens te ontvangen. Nadat de simulator deze message heeft ontvangen zullen iedere 200 ms vision gegevens aan de Vision Stub worden verstuurd.

- simVisStubIsHere

Deze message wordt verstuurd door een Vision Stub. De simulator stuurt een boodschap terug naar de stub om zijn IP adres bekend te maken.

- simReposition

Deze vraag wordt verstuurd door een Vision Stub op het moment dat de Vision Stub een zelflokalisatie wil gaan simuleren. Om dit te simuleren heeft de Vision Stub de positie van de robot nodig. De simulator geeft deze terug als antwoord.

- simGetBall

Deze message wordt verstuurd door een Vision Stub. De Vision Stub vraagt hiermee naar de positie van de bal. De simulator checkt eerst of de bal wel in het gezichtsveld van de virtuele robot is. Als dit het geval is, wordt de positie van de bal in egocentrische coördinaten teruggegeven.

- simGetRobots

Deze message wordt verstuurd door een Vision Stub. De Vision Stub vraagt hiermee naar de positie van alle robots. Van iedere robot die op de simulator is ingelogd, berekent de simulator of deze zich in het gezichtsveld van de virtuele robot bevindt. Van iedere robot binnen het gezichtsveld wordt de positie teruggegeven in egocentrische coördinaten.

- simGetLeftGoal / simGetRightGoal

Deze message wordt verstuurd door een Vision Stub. Indien de linker / rechter goal zich binnen het gezichtsveld van de virtuele robot bevindt, wordt de positie teruggegeven in egocentrische coördinaten.

Kick gerelateerde messages

- simKckStubIsHere

Deze message wordt verstuurd door een Kick Stub. De simulator stuurt een boodschap terug naar de stub om zijn IP adres bekend te maken.

- simIKickedTheBall

Deze message wordt verstuurd door een Kick Stub. Dit betekent dat de virtuele robot tegen de bal trapt. Er wordt gecheckt of de bal inderdaad in de bak van de robot ligt. Als dit het geval is wordt de bal een snelheid meegegeven.

Visualisatie gerelateerde message

- simReadWorld

Deze message wordt verstuurd door de Visualisatie Module. De Visualisatie Module geeft de staat van het veld grafisch weer. De simulator geeft de positie van alle objecten op het veld terug, zodat de het scherm kan worden ge-updated.

4.3.3 Overige functies van de simulator

4.3.3.1 Wereld gerelateerde functies

Iedere keer als de message loop wordt doorlopen, worden naast het behandelen van binnengekomen boodschappen ook enkele andere functies doorlopen. Ten eerste wordt opdracht gegeven om de positie van de bal opnieuw te berekenen. De snelheid van de bal en het tijdsinterval zijn bekend.
Ook moet van iedere robot opnieuw de positie worden uitgerekend. Hiertoe wordt een robot functie aangeroepen. Van iedere robot is de snelheid bekend en het interval waar deze snelheid op van toepassing is.
Als de positie van de bal en de robots opnieuw is berekend, wordt voor iedere robot gecontroleerd of deze niet tegen een andere robot is aangereden. Als dit het geval is zullen beide robots stil komen te staan, totdat de coach ze handmatig elders op het veld plaatst.
Ook wordt berekend of de bal en één van de robots contact hebben gemaakt. Als dit het geval is zullen de richting en de snelheid van de bal worden aangepast zoals beschreven in 3.2.2.2.
Aangezien de World Knowledge Module eens per 200 ms moet worden gevoed door de Vision Stub met vision gegevens, is er een timer ingesteld die ervoor zorgt dat iedere 200 ms alle objecten binnen het gezichtsveld naar de Vision Stub worden gestuurd. Deze gegevens worden verstuurd als er een Shape Subscription verzoek is gedaan.

4.3.3.2 Robot gerelateerde functies

De speler gerelateerde functies zijn allemaal functies die door de simulator worden aangeroepen om de gegevens van een virtuele robot te wijzigen of om gegevens over een virtuele robot te verkrijgen. De speler gerelateerde functies zijn GetOdometer, SetOdometer, SetOdometerandPosition, MoveContinuous, updatePosition en SetShapeSubscription. Al deze functies worden aangeroepen als een bijbehorende message door de simulator is ontvangen. Updateposition wordt standaard een paar keer per seconde door de simulator aangeroepen. De frequentie hiervan is systeem afhankelijk: op een snel systeem of een systeem met veel resources wordt deze functie vaker aangeroepen dan op een langzaam systeem.

4.3.3.3 Bal gerelateerde functies

Een aantal functies zijn er om de bal te bewegen en om informatie over de bal op te vragen. Ten eerste moet de bal worden geinitialiseerd. Zodra de simulator opstart, plaatst deze de bal in het veld. In de initialisatie functie wordt de begin positie van de bal gegeven. Verder is er de functie updatePosition, die een aantal maal per seconde door de simulator wordt aangeroepen. Ook de frequentie hiervan is systeem afhankelijk. De

functie updatePosition checkt bovendien ook of de bal te lang stil ligt. Als dit het geval is, wordt hij op de middenstip geplaatst. Ook maakt updatePosition gebruik van de functie checkOutOfBounce die in de gaten houdt of de bal de boarding raakt. Als dit het geval is wordt door deze functie de botsing met de boarding uitgerekend.

4.4 De Motion Stub Module

4.4.1 Inleiding

De kern van de Motion Stub bestaat ook uit de receiver loop. Zodra de Motion Stub wordt opgestart, meldt deze zich aan bij de simulator. Verder is de voornaamste taak van de Motion Stub om messages die hij krijgt, vooral van de Player Skills Module, door te geven aan de simulator, zodat deze de wereld aan kan passen aan de Motion commando's. De Motion Stub is ook doorgeefluik voor odometrie vragen en antwoorden. Voor deze opzet is gekozen omdat dit ook in de oorspronkelijke Motion Module, de real world versie, op deze manier is opgezet.

4.4.2 Messages die de Motion Stub afhandelt

MSM gerelateerde messages

- MTN_STOP_MODULE

  Deze message wordt verstuurd door de Mission Manager. Als deze message binnen komt, zal de Motion Stub een simMtnStubDied melding geven aan de simulator. Hierna zal de Motion Stub zich afsluiten.

- MTN_HEARTBEAT_TRIGGER

  Deze boodschap wordt door de Mission Manager gestuurd. De Motion Stub reageert door een heartbeat, een signaal van leven, terug te geven. Als de Motion Stub dit niet zou doen, dan zou de Mission Manager er van uit gaan dat de module niet zou draaien.

PLS gerelateerde message

- MTN_MOVE_CONTINUOUS

  Deze message wordt verstuurd door de Player Skills Module. De inhoud van de boodschap bestaat uit de lineaire snelheid en de hoeksnelheid die de robot aan moet nemen. De Motion Stub stuurt deze message door naar de simulator zodat deze de nieuwe snelheden aan de virtuele robot kan toekennen.

ODM gerelateerde message

- MTN_SET_ODOMETER

Deze message kan afkomstig zijn van de Odometrie Module. Indien dit het geval is, wordt de boodschap met odometrie gegevens doorgestuurd naar de simulator. Ook kan deze message afkomstig zijn van de Mission Manager. In dit geval is het een commando van de coach om de robot op een bepaalde plaats neer te zetten. In plaats van een gewone setOdometer, wordt nu naar de simulator een setOdometerAndPosition gestuurd.

- MTN_GET_ODOMETER

Diverse modules stellen deze vraag om odometrie gegevens aan de Motion Stub. De Motion Stub stelt deze vraag vervolgens aan de simulator, wacht op het antwoord en stuurt dit antwoord terug naar de oorspronkelijke vragensteller.

Simulator gerelateerde message

- MTN_HERE_I_AM

Deze message wordt verstuurd door de Simulator Module om zich aan te melden bij de simulator. Nadat deze boodschap is ontvangen, is het netwerkadres van de simulator bekend. Dit netwerkadres is van belang als de Motion Stub messages naar de simulator stuurt. Dit is dus de eerste boodschap die binnenkomt. Zonder dat deze message binnenkomt, kan de Motion Stub geen messages naar de simulator sturen.

Algemene message

- MTN_SUBSCRIBE_POSITION

Deze message wordt verstuurd door modules die zich willen abonneren op odometrie gegevens.

4.4.3 Overige functies van de Motion Stub

Behalve de functies die gebruikt worden voor het afhandelen van de messages, bevat de Motion Stub ook een functionaliteit die nog niet belicht is. De originele Motion Module bevatte vrij uitgebreide functies om subscriptions van andere modules op odometrie data te behandelen. Deze functies zijn ongewijzigd overgenomen, met die toevoeging dat de odometrie data nu betrokken moeten worden van de simulator, nu de odometrie sensor uiteraard niet beschikbaar is. Er wordt een lijst bijgehouden van alle modules die gegevens willen hebben met een refresh rate. Iedere keer als deze rate wordt overschreden, worden aan de simulator de odometrie gegevens gevraagd en worden deze gegevens naar de abonnee gestuurd.

4.5 De Vision Stub Module

4.5.1 Inleiding

De Vision Stub Module is voornamelijk doorgeefluik van data. In het real time robosoccer leest de Vision Module de camera uit, analyseert hij de beelden en verstuurt

hij data hierover naar andere modules. In de gesimuleerde omgeving echter heeft de Vision Stub Module zelf geen data. Alle gegevens komen uit de simulator. De Vision Stub giet alle gegevens in het juiste formaat zodat het voor de andere modules lijkt of het camera gegevens zijn.
Ook simuleert de Vision Stub de zelflokalisatie.

4.5.2 Messages die de Vision Stub afhandelt

MSM gerelateerde messages

- VIS_CONTINUE_MODULE

De Mission Manager stuurt deze message als de overige modules klaar zijn voor de Vision Module.

- VIS_PAUSE_MODULE

De Mission Manager stuurt deze message om de Vision Stub te laten pauzeren.

- VIS_STOP_MODULE

De Mission Manager stuurt deze message om de Vision Stub te stoppen.

- VIS_HEARTBEAT_TRIGGER

Deze boodschap wordt door de Mission Manager gestuurd. De Vision Stub reageert door een heartbeat, een signaal van leven, terug te geven. Als de Vision Stub dit niet zou doen, dan zou de Mission Manager er van uit gaan dat de module niet zou draaien.

WRL gerelateerde functies

- VIS_SET_SHAPE_SUBSCRIPTION

De World Knowledge Module stuurt deze message om zich te abonneren op vision gegevens. De Vision Stub geeft dit door aan de simulator die vervolgens eens per 200 ms vision gegevens aan de Vision Stub stuurt. De Vision Stub giet deze gegevens in het juiste formaat en stuurt ze naar de World Knowledge Module.

Simulator gerelateerde message

- VIS_HERE_I_AM

Deze message wordt verstuurd door de Simulator Module om zich aan te melden bij de simulator. Nadat deze boodschap is ontvangen is het netwerkadres van de simulator bekend. Dit netwerkadres is van belang als de Vision Stub messages naar de simulator stuurt. Dit is dus de eerste boodschap die binnenkomt. Zonder dat deze message binnenkomt, kan de Vision Stub geen messages naar de simulator sturen.

- VIS_MEASUREMENT

Deze message wordt verstuurd door de simulator. De message wordt vergezeld van een zogenaamd measurement, gegevens over een object uit de gesimuleerde wereld. Dit formaat wordt gebruikt door de World Knowledge Module. Deze gegevens worden naar deze module doorgestuurd.

4.5.2.4 Algemene message

- VIS_GET_OBJECT

  Deze message wordt verstuurd door de Player Skills Module of door de Team Skills Module. Er wordt gevraagd naar de positie van verschillende objecten in het veld. Er kan gevraagd worden naar de bal, naar andere robots of naar de positie van de goals. Afhankelijk hiervan wordt naar de simulator een getball, getrobots, getleftgoal of getrightgoal gedaan. De simulator zal dan gegevens terug sturen die de Vision Stub gebruikt als antwoord aan de oorspronkelijke vragensteller.

4.5.3 Overige functies van de Vision Stub

Omdat de Vision Module ook nog de zelflokalisatie doet door op grond van vaste punten in het veld de eigen positie te berekenen, moet de Vision Stub ook dit simuleren. Er kan ingesteld worden hoe vaak dit moet plaatsvinden. Als zelflokalisatie plaats moet vinden, dan wordt de werkelijke positie van een robot opgevraagd aan de simulator. Met deze gegevens wordt een WRL_Reposition commando aan de World Knowledge Module gegeven en is ook deze functie gesimuleerd.

4.6 De Kick Stub Module

4.6.1 Inleiding

De Kick Stub is de simpelste van alle stubs. Deze behoeft zich alleen maar aan te melden bij de simulator, een teken van leven te geven aan de Mission Manager en door te geven aan de simulator wanneer een robot tegen de bal trapt.

4.6.2 Messages die de Kick Stub afhandelt

MSM gerelateerde messages

- KCK_STOPMODULE

  De Mission Manager stuurt deze message om de Kick Stub te stoppen.

- KCK_HEARTBEAT_TRIGGER

  Deze boodschap wordt door de Mission Manager gestuurd. De Kick Stub reageert door een heartbeat, een signaal van leven, terug te geven. Als de Kick Stub dit niet zou doen, dan zou de Mission Manager er van uit gaan dat de module niet zou draaien.

PLS gerelateerde message

- KCK_KICK

  Deze message wordt verstuurd door de Player Skills Module. De robot krijgt dus de opdracht tegen de bal te trappen. Dit wordt doorgestuurd naar de simulator.

Simulator gerelateerde message

- KCK_HERE_I_AM

  Deze message wordt verstuurd door de Simulator Module om zich aan te melden bij de simulator. Nadat deze boodschap is ontvangen, is het netwerkadres van de simulator bekend. Dit netwerkadres is van belang als de Kick Stub messages naar de simulator stuurt. Dit is dus de eerste boodschap die binnenkomt. Zonder dat deze message binnenkomt kan de Kick Stub geen messages naar de simulator sturen.

5. Resultaten

5.1 Inleiding

Na implementatie en testen van de software is een evaluatie van de functionaliteit essentieel. Twee aspecten moesten hierbij onder de loep genomen worden. Ten eerste moest de software niet merken dat in plaats van met robots met een simulator gewerkt werd en ten tweede moest ook de werkelijkheid redelijk kunnen worden nagebootst. Omdat het uiteindelijke doel is: het verbeteren en makkelijker kunnen testen van de Team Skills Module, is het van belang dat één robot individueel met een bal overweg kan. Als dit is gerealiseerd, kan ook een heel team worden gesimuleerd.

5.2 Robuustheidstest

Om de robuustheid van de simulator te kunnen testen, is een opstelling gebruikt waarin één virtuele robot was ingelogd op de simulator die een wedstrijd speelde. Alle modules moeten hiervoor draaien. In de praktijk zag het er uit als volgt: de robot gaat naar de bal, scoort de bal, de bal gaat naar de middenstip, de robot zoekt de bal en begint opnieuw. De simulator bleek dagen lang te kunnen draaien. Ook met alle modules en alle stubs draaiend bleek de simulator stabiel. Als een module crashte, zorgde de Mission Manager voor een reanimatie en kon de robot in de simulator weer verder. Tijdens de tests met de echte robots zoals verder in dit hoofdstuk beschreven, viel op dat ook dan de modules regelmatig crashten en door de Mission Manager opnieuw worden opgestart. In vrijwel alle gevallen is de oorzaak het vollopen van een message queue. Een module moet dan zo lang op een antwoord wachten dat geen heartbeat meer wordt gegeven, waardoor wordt verondersteld dat deze niet meer draait. In de toekomst kunnen de bugs uit de crashende modules gehaald worden. De simulator kan dan gebruikt worden om de modules langdurig te laten draaien, zodat met enige zekerheid van een langdurige stabiliteit kan worden uitgegaan.

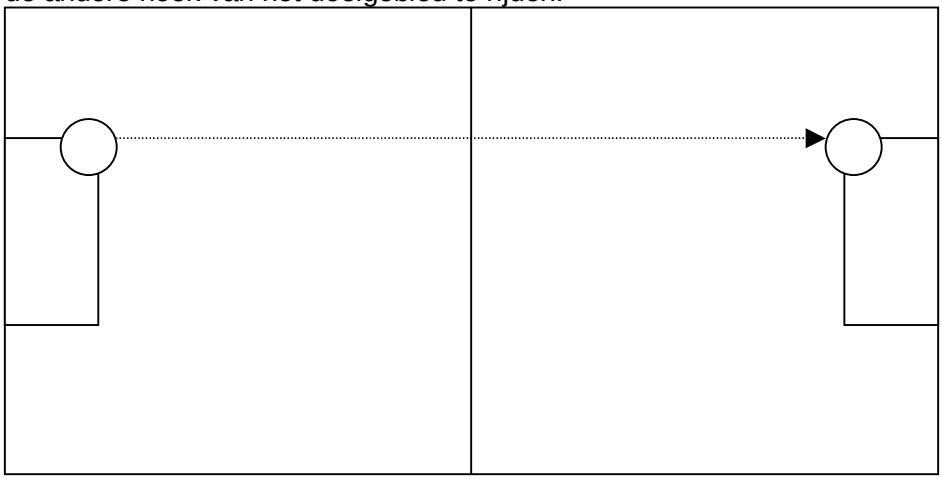
5.3 Realiteitstest

5.3.1 Inleiding

Om te kunnen bekijken hoe waarheidsgetrouw de simulator is, moest bekeken worden hoe goed de bewegingen van de virtuele robot de werkelijkheid benaderden. Ook moest worden bekeken of de bal in combinatie met de robots zich wel net zo bewoog als in de werkelijkheid. Dit is de essentiële basisfunctionaliteit van de virtuele robot. Om deze beide zaken te kunnen testen zijn vier testopstellingen gebruikt. Eén om de robot in een rechte lijn te laten rijden, één om de robot met de bal te laten dribbelen met de bal tussen de robot en de goal, één met de bal op een plaats zodanig dat hij in de richting van de goal moet worden gedraaid en één waarbij de schopsnelheid en het afremmen van de bal moest worden getest.

5.3.2 Test van een rijdende robot

Ten eerste moest de rijdende robot worden getest. Hiertoe is gemeten hoe de robot zich bewoog als hij de opdracht kreeg om van de hoek van het doelgebeid naar de overkant naar de andere hoek van het doelgebied te rijden.



Afbeelding 5.1 De robot rijdt van (–3500,1500) naar (3500,1500)

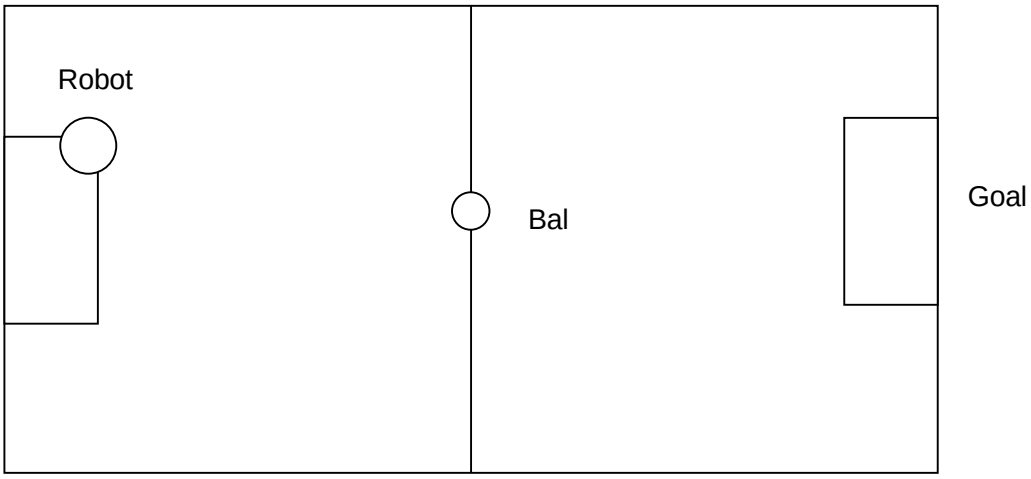
Echte Robot		**Simulator**	
Afgelegde X	Afgelegde Y	Afgelegde X	Afgelegde Y
6940	220	6880	275
6910	220	6900	276
6980	340	6820	272
6960	270	6830	273
6950	180	6790	271
6950	420	6800	272
6940	310	6850	275
6960	150	6900	276
6950	180	6790	271
6920	380	6900	276
Gemiddeld			
6946	267	6846	273
Std Dev			
20.1	92.2	46.2	2.1

Tabel 5.1 de resultaten van de rijtest met de robot

Zoals ook te zien valt aan de tabel heeft een robot of het veld waarop gereden werd een systematische afwijking. De robot had een structurele afwijking naar links. De afgelegde weg in x richting ligt tussen de 6910 mm en de 6980 mm. De robot stopt dus altijd voordat hij de plaats van bestemming heeft bereikt. Dit is als volgt te verklaren: door wielspin komt de robot minder ver dan het aantal wielomwentelingen tot gevolg zou moeten hebben. De afwijking in deze richting is zeer beperkt, minder dan 1% met een standaarddeviatie van 20 mm. De simulator is hierop ingesteld, zoals eerder beschreven

in hoofdstuk 3. De afwijking in de y richting is groter dan de afwijking in de x richting. Ook de standaard deviatie is veel groter. Dit komt doordat een afwijking in de y richting zeer groot kan worden als de robot aan het begin van de baan al een afwijking naar links heeft of licht naar links gedraaid raakt. De simulator houdt rekening met een afwijking in zijwaartse richting, die afhankelijk is van de afgelegde weg. Bij een afgelegde weg van 7 m is de afwijking van een rechte lijn ongeveer 4%. De echte robot is een robot die structureel naar links afwijkt. De simulator is daarom in dit geval ook ingesteld met een bias naar links van 4%. In de praktijk verdient dit aspect van de ruis nog enige aandacht. Beter is het om in de toekomst ruis toe te voegen in de hoeksnelheid, waardoor niet alleen ruis wordt toegevoegd in de y richting, maar ook in de oriëntatie.

5.3.3 De test met de bal tussen de robot en de goal



Figuur 5.2 Test met de robot en een bal

Om het gedrag van de robot met een bal te testen, is gekozen voor bovenstaande opstelling. De robot start op positie (-3500,1500) en de bal start op (0,0). In de eerste test heeft de robot oriëntatie 0 en kijkt dus recht naar de bal en naar de goal waarin moet worden gescoord. Dit lijkt een relatief makkelijke opdracht.
Ook is dezelfde test gedaan met de robot terwijl deze oriëntatie –900 had. De robot kan de bal dus niet zien. De robot moet dus eerst de bal gaan zoeken voordat deze kan gaan scoren.
Bij de test met oriëntatie 0 rijdt de robot in de richting van de bal. Zodra deze de bal bereikt, gaat hij met de bal dribbelen in de richting van de goal. Zodra de robot dicht genoeg bij de goal komt, schiet hij. Zoals te zien in tabel 5.2 is de succes factor zeer hoog. De enige keer dat de robot niet tot scoren kwam, werd de bal naast de goal geschoten en kwam de bal vast te liggen tegen de boarding. Ook werd de tijdsduur gemeten hoe lang het duurde voordat gescoord werd. Bij het berekenen van de gemiddelde tijdsduur zijn alleen de pogingen in ogenschouw genomen waarbij gescoord werd. Afwijkingen in de tijdsduur voordat er gescoord werd, onstonden doordat de robot de bal wel eens kwijt raakte tijdens het dribbelen. De robot moest dan weer opnieuw de bal gaan zoeken. In de tabel valt dit waar te nemen aan de tijdsduur voordat er gescoord

werd. Een rechtstreeks doelpunt werd gescoord in 8 s. Als het doelpunt langer op zich liet wachten, dan kwam dit doordat de robot de bal onderweg verloren had.
De testopstelling met oriëntatie –900 bleek een zelfde resultaat op te leveren. De tijden voordat er werd gescoord, zijn iets langer, aangezien eerst de bal moest worden gezocht. In de praktijk bleek dit binnen 2 seconden gebeurd te zijn. Vandaar dat de resultaten vrijwel hetzelfde zijn, zij het dat de tijd tot een doelpunt iets langer is.
In de simulator werd dezelfde testopstelling gebruikt. Wat opvalt uit de tabel is dat de tijden tot een doelpunt korter zijn. Dit komt doordat de een virtuele robot iets beter blijkt te dribbelen. Net als de echte robot scoort de virtuele robot in de ideale situatie in 8 s. De virtuele robot raakt minder vaak de bal kwijt en daardoor wordt er gemiddeld eerder gescoord. Ook schiet de simulator nauwkeuriger: er werd niet mis geschoten. Ook bij de tweede testopstelling gedroeg de simulator zich goed. Net als in de echte wereld draaide de virtuele robot net zolang totdat de bal in zicht kwam, waarna de bal werd onderschept, gedribbeld en gescoord.

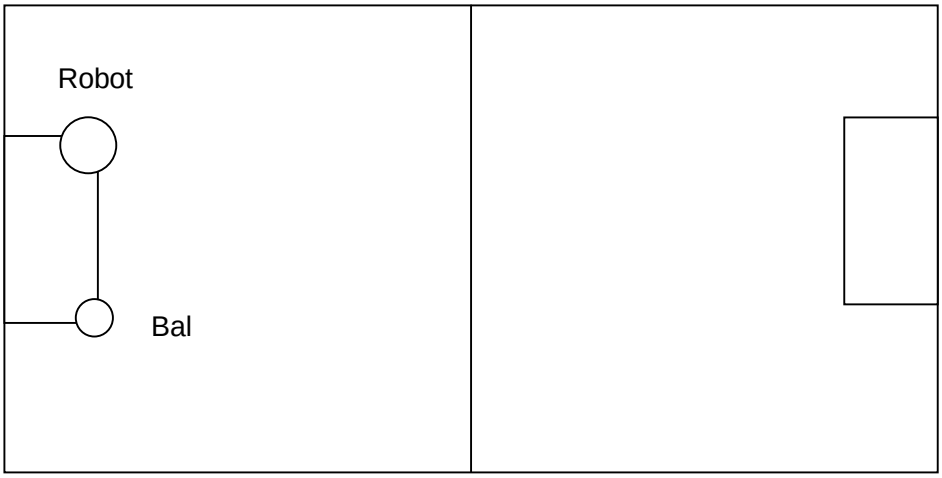
Echte robot		**Simulator**	
Succes	Tijd (s)	Succes	Tijd
Goal	15	Goal	12
Goal	8	Goal	8
		Goal	8
Goal	8	Goal	9
		Goal	8
Goal	10	Goal	9
Goal	9	Goal	8
Goal	10	Goal	10
Goal	8	Goal	9
Goal	8	Goal	8
Percentage succes	**Gemiddeld + σ**	**Percentage succes**	**Gemiddeld + σ**
80%	9.5 + 2.3	100%	8.9 + 1.2

Tabel 5.2 de resultaten van de eerste dribbeltest met oriëntatie 0

Echte robot		**Simulator**	
Succes	Tijd (s)	Succes	Tijd
Goal	9	Goal	9
Goal	9	Goal	11
Goal	11	Goal	9
Goal	10	Goal	9
Goal	13	Goal	10
		Goal	10
Goal	10	Goal	9
Goal	14	Goal	10
Goal	9	Goal	9
Goal	10	Goal	11
Percentage succes	**Gemiddeld + σ**	**Percentage succes**	**Gemiddeld + σ**
90%	10.6 + 1.8	100%	9.7 + 0.8

Tabel 5.3 de resultaten van de eerste dribbeltest met oriëntatie –900

5.3.4 De test met de bal in een hoek tussen de robot en de goal



Figuur 5.3 Test met de robot en een bal

Om de simulator te testen met een meer complexe dribbel is voor deze testopstelling gekozen. Praktijk en testen met de simulator hadden al uitgewezen dat dit een bijzonder moeilijk probleem oplevert. Voordat de bal in de richting van de goal moet worden geschoten, moet de bal ten opzichte van de robot een hoek van 90 graden naar links maken. Dit bleek een meestal onoverkomelijk probleem. De bal kwam meestal ergens stil te liggen in de buurt van de boarding, waar de robot er niets meer mee kon. Vaak deed de robot zelfs een poging om door de boarding heen te rijden. Incidenteel werd de bal met succes naar links bewogen in de richting van de goal waarna de dribbel richting de goal kon worden ingezet en worden gescoord. Als na 30 seconden de bal nog niet was gescoord dan werd de poging als gefaald beschouwd.
De simulator bleek moeite te hebben met deze opstelling en de virtuele robots hadden moeite om te scoren. De virtuele robots dribbelden vaak in de richting van de boarding met als gevolg dat de bal klem kwam te liggen tussen de robot en de boarding.
Aangezien de simulator een soort scheidsrechter heeft ingebouwd die de bal nadat deze 10 seconden heeft stilgelegen op de middenstip legt (ook in het echte spel wordt de bal op de middenstip gelegd als deze langdurig niet bewogen heeft), moesten pogingen waarin dit optrad buiten beschouwing worden gelaten. Als de bal namelijk eenmaal op de middenstip werd gelegd, dan kon de virtuele robot makkelijk richting de goal dribbelen en scoren.
Zowel de echte robot als de virtuele robot hebben een zeer laag succes percentage. In beide gevallen komt de bal meestal klem te liggen tegen de boarding.

Echte robot		**Simulator**	
Succes	Tijd (s)	Succes	Tijd
	>30	Goal	16
	>30		>30
Goal	18		>30
	>30		>30
	>30	Goal	18
	>30		>30
Goal	15	Goal	14
	>30		>30
	>30		>30
	>30		>30
Percentage	**Gemiddeld**	**Percentage**	**Gemiddeld**
20%	16.5	30%	16

Tabel 5.3 de resultaten van de tweede dribbeltest met oriëntatie –900

5.3.5 De test met de kicker

Om te bepalen welke grootheden gebruikt moeten worden in de simulator aangaande de snelheid die de bal ontvangt nadat hij is geschopt en de vertraging die de bal ondervindt bij het rollen, is de afstand die de bal aflegt nadat deze is geschopt gemeten en de tijd die deze onderweg was. Omdat de robot met gemak de bal het hele veld over kan schieten, is voor deze test uitgeweken naar de gang, waar het laboratorium aan grenst. Dit brengt met zich mee dat de test op een andere ondergrond heeft plaatsgevonden, dan waar een wedstrijd robosoccer zich op af speelt. De groene mat van het veld heeft een meer remmende werking op de bal dan het zeil op de vloer in de gang. Bij berekening van de beginsnelheid van de bal en van de vertraging is ervan uit gegaan dat het rollen van een bal éénparig vertraagd is.

Afstand (m)	**Tijd (s)**	**Gemiddelde v (mm/s)**	**V_0**	**Acceleratie (mm/s$^{2)}$**
17.5	23	760	1520	-66
17.7	26	680	1360	-52
16.2	22	740	1480	-67
15.9	21	750	1500	-71
16.7	22	760	1520	-69
17.9	26	690	1380	-53
19.5	27	720	1440	-53
Gemiddeld				
17.3	23.8	728	1456	-62

5.4 Algemeen

Zonder al te optimistisch te zijn kan worden geconcludeerd dat een virtuele robot voldoende weg heeft van een echte robot. Na de eerste test, de test met het rijden van de robot, kwam naar voren dat de virtuele robot, net als de echte, een afwijking vertoont in de rijrichting van ongeveer 1%. Dat de ruis in de y richting niet overeen komt met de geteste robot, is niet belangrijk: het gaat vooral om het balgedrag. Je wilt de simulator wel zo nauwkeurig mogelijk hebben, maar voor dit doel lijkt het wel goed.
Het balgedrag van de simulator bleek beter overeen te komen met de werkelijkheid. De dribbelende robot boekt ongeveer dezelfde resultaten als de dribbelende virtuele robot, met dien verstande dat de virtuele robot iets beter met de bal omgaat.
De parameters die bleken uit de proef met de kicker van de robot zijn hard in de simulator overgenomen.
Als een virtuele robot wordt bekeken op een computer scherm dan ziet zijn gedrag en zijn interactie met de bal er waarheidsgetrouw uit. Wat dit betreft kan de simulator in de toekomst een grote hulp zijn bij de ontwikkeling en het testen van de overige modules.

6. Conclusie

Na ruim een jaar met het onderzoek naar de simulator, het ontwerp en de implementatie van de software en het schrijven van dit verslag bezig te zijn geweest, kan op een project met wisselend succes worden teruggekeken. Aan de ene kant kan op realistische wijze een robot worden gesimuleerd, aan de andere kant is het uiteindelijke doel, het naspelen van een echte wedstrijd, nog niet bereikt.

Het robosoccer project is zeer omvangrijk. Omdat de simulator en de stubs met zeer veel modules moeten communiceren, was het essentieel om vrij gedetailleerd van de werking van alle modules op de hoogte te zijn. Van niet alle modules was voor dit doeleinde bruikbare documentatie beschikbaar. Ook was er niemand die voldoende kennis over alle modules in zich verenigde. Dit had tot gevolg dat het zeer veel tijd heeft gekost om voldoende kennis te vergaren over het project om de simulator te kunnen ontwerpen en implementeren.

Het simuleren van een robot kan succesvol genoemd worden. De Motion Stub reageert naar behoren op commando's van de PLS Module. De simulator genereert input voor de Vision Stub en voor de ODM Stub. Wat nog aandacht verdient in de toekomst is het algoritme waarmee de ruis in de beweging van de virtuele robot wordt toegevoegd. Nu is gewerkt met factoren die invloed hebben op de x en y richting, in een egocentrisch coordinatenstelsel. Het verdient aanbeveling in de toekomst geen ruis toe te passen op de positie, maar op de snelheden. Een afwijking in de lineaire snelheid heeft dan een afwijking in de afgelegde weg tot gevolg en een afwijking in de hoeksnelheid heeft een zijwaartse afwijking tot gevolg in de baan van de robot en een afwijking in de oriëntatie. Dit is meer realistisch dan tot nu toe is gebruikt in de simulator.
Meer realistisch is het ook om ervan uit te gaan dat de Vision Module niet perfect werkt. De Vision Stub zou dan ook ruis moeten gaan genereren. Deze zou dan moeten worden toegevoegd zoals beschreven in hoofdstuk 3.

Een ander belangrijk onderdeel, het simuleren van de bal, is ook ruim voldoende geslaagd. De bal reageert op de Kick Stub. Als er tegen de bal wordt getrapt dan krijgt deze een bepaalde snelheid in een bepaalde richting. Als de bal in de bak van een virtuele robot belandt dan reageert de bal hierop op een manier die lijkt op de dribbel van een echte robot. Dit houdt in dat ook de virtuele robot de bal kan verliezen, net als de echte robot, en dat hij er zeer veel moeite mee heeft om dribbelend met de bal te draaien.

Wat niet geslaagd is aan dit project is dat niet, zoals van tevoren beoogd, een hele wedstrijd gesimuleerd kan worden. Dit heeft twee oorzaken: ten eerste is het message systeem hier niet voor geschikt. Een gesimuleerde omgeving zorgt voor zoveel meer netwerkverkeer dan een echte omgeving dat op het moment dat er meerdere virtuele robots werden ingelogd, het message systeem de grote hoeveelheid messages niet meer aankon. Er wordt in de toekomst gekeken of met een ander message systeem, SPLICE genaamd, betere resultaten gehaald kunnen worden. Het zou zaken ook een stuk makkelijker maken als niet per virtuele robot een apart IP adres benodigd is, zodat alle virtuele robots op één computer zouden kunnen draaien.
De tweede oorzaak was dat de overige software ook niet was ingericht voor een dergelijke toepassing. Op het moment dat er een echte of virtuele robot wordt opgestart

met de software die is ontwikkeld door Amsterdam en Delft, wordt deze robot door de andere robots gezien als teamgenoot. Zij communiceren met elkaar als teamgenoten en geven informatie over hun wereldbeeld door aan alle andere robots die opgestart zijn. Ook de Team Skills zorgt voor communicatie tussen robots onderling. Er moeten dus wijzigingen worden aangebracht, zodanig dat bij het opstarten van een robot kan worden aangegeven of deze bij team A of B hoort, zodat tijdens de wedstrijd de robots niet met tegenstanders communiceren. Dit zou door een coach moeten kunnen worden ingesteld.

Al met al is dit een goed begin geweest voor een wedstrijdsimulator. Als de verbeteringen worden aangebracht zodat wel een wedstrijd in volledige bezetting kan worden gespeeld, is dit een waardevol instrument bij het instellen en perfectioneren van de TMS Module en de PLS Module. Deze simulator zal dan in de toekomst bij kunnen dragen aan een hogere kwaliteit van het voetbalteam Clockwork Orange.

Literatuur

[1] Dr. Robin Murphy,University, Center for Robot-Assisted Search and Rescue, http://www.csee.usf.edu/~murphy

[2] Prof Ricardo Cassinis, Robots for Demining, http://www.ing.unibs.it/~cassinis/main.htm

[3] Bas Terwijn, Coach Documentation, Intern rapport, IAS sektie Instituut voor informatica, Universiteit van Amsterdam

[4] Matthijs Spaan, Team play among soccer robots, Masters Thesis AI, Universiteit van Amsterdam, april 2002

[5] Werner Altewischer, Implementation of robot soccer player skills using vision based motion, Masters Thesis, Department of Applied Physics, Faculty of Applied Science, TU Delft, dec 2001

[6] Raymond Dondervoort, Evaluation of the world module of Clockwork Orange, Masters Thesis AI, Universiteit van Amsterdam, april 2002

[7] Jurjen Caarls, Fast and accurate robot vision for vision- based motion, Masters Thesis, Department of Applied Physics, Faculty of Applied Science, TU Delft, aug 2000

[8] Will van Geest, Description of the message system, Intern rapport, ITS, TU Delft

[9] Gerhard Kraetzschmar, Home Page of RoboCup Middle Size Robot League, http://smart.informatik.uni-ulm.de/ROBOCUP/f2000/index.html

[10] Veloso, Manuela / Pagello, Enrico / Kitano, Hiroaki RoboCup-99 : robot soccer world cup III : [third conference held in Stockholm, Sweden, 27.06 - 06.08.1999] Berlin Springer, 2000