Erik Oppedijk is trainer/consultant bij Info Support

Robots in de Microsoft-wereld

AAN DE SLAG MET MICROSOFT ROBOTICS STUDIO

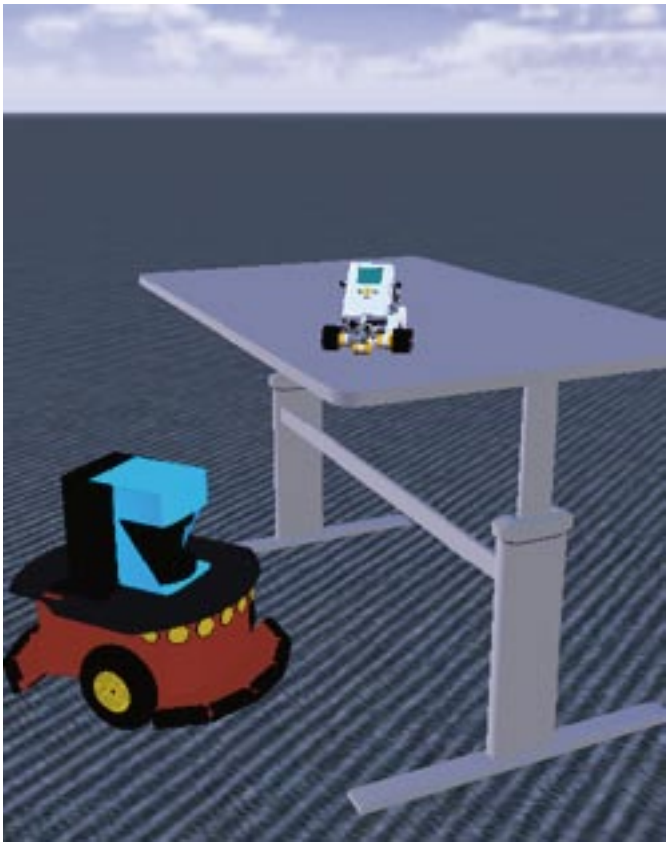
Voor het aansturen en programmeren van robots waren we tot nog toe altijd overgeleverd aan de software van de leveranciers van de diverse onderdelen, zoals motoren, robotarmen en afstandsensoren. De integratie van deze onderdelen leverde altijd interessante uitdagingen op, voornamelijk om alle onderdelen asynchroon te laten werken, bijvoorbeeld om tijdens het rijden voorwerpen te vermijden. Voor de deelname aan de RoboChallenge 2007 hebben wij gekeken naar Microsoft Robotics Studio met de mogelijkheid voorgaande problemen op te lossen.

De RoboChallenge is een jaarlijks terugkerende wedstrijd waaraan iedereen kan deelnemen. Het doel is het bouwen van een autonome robot, die geheel zelfstandig in een arena van 5 x 5 meter zijn weg moet vinden en objecten moet oppakken. De objecten zijn 'sterren' (gekleurde kerstballen) waarbij er per missie verschillende kleuren zijn die de robot wel en niet moet pakken. In één van de missies weet de robot bijvoorbeeld alleen maar dat de kleur die hij moet pakken aan de overkant aanwezig is, en op zijn eigen helft hangen juist de kleuren die strafpunten opleveren. Van eerdere deelnames weten we dat er een paar belangrijke zaken zijn om überhaupt een goede robot te kunnen maken. Onderdelen zoals een solide platform, strategiemodule, visionsysteem voor balherkenning, plus nog een heleboel elektronicazaken, zoals controllerboards, batterijen, servo's, sensoren en dergelijke, moeten ook allemaal correct functioneren. Bij het werken met een echte robot hebben we al diverse malen ondervonden dat het integreren van al deze zaken veel werk is, waarbij één fout het hele testen en kalibreren stillegt. Ook het toevoegen van multithreading geeft een hoop extra interessante uitdagingen voor het programmeren.

Microsoft Robotics Studio

Vanuit de robot-community was er nog geen generieke tool om veel van deze problemen op te lossen. Dit is waar Microsoft Robotics Studio (MSRS) om de hoek komt kijken. Ten eerste kunnen alle hardwareleveranciers hun software nu als een service aanbieden binnen MSRS (eigenlijk hebben we het hier dan over Hardware As A Service), verder biedt Robotics Studio een complete Concurrency and Coordination Runtime (CCR) aan voor bijvoorbeeld het parallel afhandelen van alle taken. Daarnaast beschikt het over een volledige 3D-simulatieomgeving (inclusief zwaartekracht en collision detection) om gesimuleerde robots in een nagebouwde wereld al gedeeltelijk te testen, zie afbeelding 1. Voeg hier nog een centrale logging, service repository (via http) en een visuele programmeeromgeving aan toe en je hebt het complete platform om (robot) hardware aan te sturen.

Vanuit dit centrale platform is het nu veel gemakkelijker om een robot te maken, gebaseerd op standaardonderdelen van diverse leveranciers. Combineer dit met ingebouwde ondersteuning voor webcam's en GPS-devices en de wereld ligt open. Natuurlijk moet er voor eigen hardware een zelfgeschreven service gemaakt worden, maar daarna kan deze service meedraaien in MSRS. Dit zal in eerste instantie nogal wat extra werk zijn, maar steeds meer leveranciers beginnen hier code voor aan te bieden. Het belangrijkste is dat de hardwareservices geschikt zijn om multithreaded te werken, iets waar nog niet alle huidige COM DLL's en drivers aan voldoen. De gehele omgeving is gebaseerd op generieke contracten, zoals generieke drive, webcam, motor, bumpers, laser range finders, en dergelijke. Onze robot spreekt tegen de generieke laag en at runtime kiezen we vanuit een config-file (manifest) welke specifieke implementatie er gebruikt moet worden. Zo kunnen we dus de logica om rond te rijden en obstakels te vermijden eerst testen met een gesimuleerde robot en daarna dezelfde code starten op onze real-life robot.



Afbeelding 1. Een volledige 3D-simulatieomgeving om robots te kunnen testen

Het opstarten van de simulatieomgeving

Na het downloaden en compleet installeren van Microsoft Robotics Studio 1.5 vinden we in het startmenu een Microsoft Robotics Studio Command Prompt. In deze MSRS Command Prompt starten we een Decentralized System Service (DSS) op en laten deze luisteren naar een bepaalde poort: Dsshost /p:50000.
Start een webbrowser op en ga naar http://localhost:50000/ om alle services en de control panel te bekijken. Robotics Studio gaat zelf alle services in de MSRS/bin/services/ af en voegt deze toe aan de control panel. Klik op 'Control Panel' om alle beschikbare services te zien, typ in de search box '2' in. Er zijn nu nog vijf services over. Selecteer bij de 'Simulation Tutorial 2' service het samples\Config\SimulationTutorial2.manifest.xml-bestand uit de dropdown. Dit manifestbestand bevat alle gerelateerde services die ook opgestart moeten worden, in dit geval een dashboardapplicatie. Klik op 'Create' om de simulatieomgeving en een dashboard op te starten. Er worden nu twee applicaties opgestart, een generiek dashboard om robots aan te sturen(zowel real-life als gesimuleerd) en de simulatieomgeving. Vul in de dashboardapplicatie als machinenaam 'localhost' in en druk op 'Connect'. Dubbelklik op '(P3DXMotorBase)/simulateddifferentialdrive/{guid}', klik op de 'Drive'-knop en beweeg de grootste van de twee robots met de cirkel boven de 'Stop'-knop. Om door de simulatieomgeving te bewegen, kunnen we de pijltoetsen gebruiken, en met 'F8' kun je naar de volgende camera springen, om bijvoorbeeld mee te kijken met onze robot.

Status van services bekijken

We gaan terug naar de webbrowser (http://localhost:50000/) en bekijken hier onder 'Service Directory' de '/simulatedwebcam/{guid}'. Deze webcam is op de grootste robot gemonteerd, door in deze service op 'Start' te klikken, krijgen we beelden binnen vanaf de robot terwijl deze rondrijdt (beweeg de robot via de dashboardapplicatie). Kijk ook bij de overige services in de 'Service Directory': deze geven alleen in XML hun huidige status weer, maar de webcam-service gebruikt een XSL-T om een betere opmaak te tonen met de beelden van de gesimuleerde webcam.

Het maken van een eigen service

We gaan een eigen service maken die in de simulatieomgeving een robot simpel aanstuurt. We starten met MSRS door vanaf de Robotics Studio Command Prompt (vergeet onder Vista niet om dit met administratieve rechten op te starten) het volgende commando uit te voeren: dssnewservice /s:DemoRobot
(tip: Laat de naam niet op 'Service' eindigen, MSRS plakt dit er normaal zelf al achter).Nu hebben we een directory 'DemoRobot' gekregen, waaruit we vervolgens het gegenereerde bestand DemoRobot.csproj kunnen opstarten met Visual Studio 2005. We hebben in dit project nu drie bestanden, een manifestbestand met daarin alle relevante informatie om deze service op te starten, een types-bestand met daarin de mogelijke State van de robot en de interfacedefinitie van de service, en een DemoRobot.cs waarin we onze implementatie gaan schrijven. Als eerste voegen we een referentie toe naar 'RoboticsCommon.Proxy', waarin we onder andere de proxy naar het generieke 'Drive'-contract gaan vinden. In het bestand DemoRobot.cs vullen we in de 'Start'-methode de code uit codevoorbeeld 1 in. Deze code zal voor ons de robot in een rondje laten rondrijden. Deze code is gekoppeld aan een generiek 'Drive'-contract, zodat het onze service om het even is of we straks met een echte of een 'virtuele' robot gaan rondrijden. MSRS biedt op deze manier een hele mooie abstractielaag aan om de logica van de robot snel op verschillende systemen te testen. Onze service stuurt drive-berichten naar een partnerservice, via de _drivePort, deze partner wordt later opgestart vanuit een manifestbestand.

LET OP: nog niet alle methodes zijn in de simulatie geïmplementeerd, o.a. de RotateDegrees en DriveDistance werken nog niet!

Om onze 'virtuele' robot te gaan starten, maken we gebruik van een tweede reeds bestaand manifest, namelijk het manifest van ServiceTutorial2 dat we al eerder gebruikt hebben. Hierin is al een robot aanwezig met een drive-contract. We doen dit door naar het property-scherm van het project te gaan, de 'Debug'-setting te selecteren en onder de 'Start Options' voegen we aan het einde van de Command line-arguments het volgende toe: -m:'samples/config/simulationtutorial2.manifest.xml'

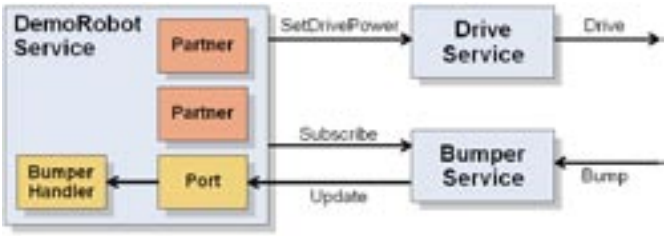
```
using drive = Microsoft.Robotics.Services.Drive.Proxy;
...
[Partner("drive",
   Contract = drive.Contract.Identifier,
   CreationPolicy = PartnerCreationPolicy.UseExisting)]
private drive.DriveOperations _drivePort
   = new drive.DriveOperations();

protected override void Start()
{
   base.Start();
   //SubscribeToBumpers();

   //Start driving, left wheel speed, right wheel speed
   _drivePort.SetDrivePower(0.9f,1.0f);
   LogInfo(LogGroups.Console, "Demo Robot Started");
}
```

Codevoorbeeld 1.

```
using bumper = Microsoft.Robotics.Services.ContactSensor.Proxy;

...

[Partner("bumper",
   Contract=bumper.Contract.Identifier,
   CreationPolicy=PartnerCreationPolicy.UseExisting)]
private bumper.ContactSensorArrayOperations _bumperPort
   = new bumper.ContactSensorArrayOperations();

private void SubscribeToBumpers()
{
   bumper.ContactSensorArrayOperations bumperNotificationPort
      = new bumper.ContactSensorArrayOperations();
   _bumperPort.Subscribe(bumperNotificationPort);

   //Start listening for notifications messages
   Activate(
      Arbiter.Receive
         <bumper.Update>
           (true, bumperNotificationPort, BumperHandler));
}

private void BumperHandler(bumper.Update notification)
{
   LogInfo(LogGroups.Console, "Bumper touched!");
   _drivePort.SetDrivePower(-0.9f, -1.0f);
}
```

Codevoorbeeld 2.

Nu we klaar zijn met het programmeren, wordt het tijd om de robot op te starten. Dit kan door bijvoorbeeld het project in debug mode op te starten. We zien de simulatieomgeving verschijnen, met daarin de gesimuleerde robot die rondjes door de omgeving begint te rijden. Normaal gesproken zouden we in ons eigen manifestbestand een robot opstarten, maar omdat er in de simulationtutorial2 al een robot met een 'Drive'-contract aanwezig is, wordt



Afbeelding 2. Extra onderdelen voor het ontvangen van berichten van de bumper

deze gebruikt. In de consoleapplicatie zien we ook de uitvoer van de loginfo: dit is een handige helperfunctie om centrale logging in te bouwen.

Reageren op botsingen

We gaan nu een 'Bumper'-contract gebruiken om botsingen te detecteren. Om de bumper te gebruiken, hebben we een aantal extra onderdelen nodig om berichten van de bumper te ontvangen, zie afbeelding 2. Hiervoor maken we een BumperHandler om de berichten vanaf de bumper te verwerken. Vervolgens moeten we zorgen dat de bumperservice weet dat onze handler aangeroepen moet worden. Hiervoor gebruiken we weer een partner (deze zal later vanuit een manifest weer ingevuld worden) en vanuit de Start()-methode roepen we een SubscribeToBumpers() aan. In deze functie maken we een eigen notificationPort voor het opvangen van de berichten van de bumper, waarop we een abonnement nemen. Als laatste starten we een aparte persistente taak om de bumperberichten te ontvangen vanaf de notificationPort en deze aan de BumperHandler door te geven, zie codevoorbeeld 2.

Verbeteringen

Een aantal verbeteringen dat we hier nog kunnen doorvoeren, zijn het slimmer reageren op botsingen, we kunnen in de service ook een 'state' vasthouden, met daarin bijvoorbeeld de richting, om te kunnen wisselen tussen vooruit en achteruit rijden bij een botsing. Verder zouden we ook Vision in kunnen bouwen, kijk voor een voorbeeld hiervan naar de Sumo- en Soccer-simulatieomgevingen die ook te downloaden zijn vanaf de Microsoft Robotics-site.

Hoe nu verder

Tot zover deze eerste introductie van Microsoft Robotics Studio versie 1.5. In de toekomst zullen we aandacht besteden aan de Concurrency and Coordination Runtime (CCR), die ook los gebruikt kan worden in projecten waar we concurrency-problemen willen oplossen. Microsoft heeft ondertussen al een flink aantal video's online gezet op de MSRS-site (zie referenties) met uitleg over de gehele omgeving. Ook de meegeleverde samples/tutorials bieden veel extra informatie. Het is verstandig deze in ieder geval ook door te werken om alle verschillende onderdelen en mogelijkheden tegen te komen. Daarnaast zijn er steeds meer partijen die drivers/software voor Robotics Studio aanbieden, waaronder ondertussen ook al een aantal vision-libraries.

Erik Oppedijk is trainer/consultant op het Kenniscentrum van Info Support. Hier houdt hij zich bezig met producten als ASP.NET en Microsoft BizTalk Server. Daarnaast is Erik ook ieder jaar actief bij de bouw van de Info Support-deelname aan de RoboChallenge-wedstrijden. Zijn e-mailadres is eriko@infosupport.com.

Referenties

MSRS-site met downloads: http://www.microsoft.com/robotics

Channel 9 info: http://channel9.msdn.com/wiki/default.aspx/Channel9.MSRoboticsStudio

Robotics team blog: http://blogs.msdn.com/MSRoboticsStudio/

Robo Challenge: http://www.robochallenge.nl/

CCR: http://msdn.microsoft.com/msdnmag/issues/06/09/ConcurrentAffairs/