Workshop “ROS voor Engineers” - deel 1

auteur: Eric Dortmans (e.dortmans@fontys.nl)

Handige tips voor je begint

Je kunt een nieuw terminal window openen met de toetscombinatie: *CTRL-ALT-t*.

Alle commando’s ondersteunen *tab-completion*, d.w.z. zodra je de *TAB* toets indrukt probeert Ubuntu of ROS wat je al hebt ingetikt te completeren.

Je kunt door eerder ingevoerde commando’s heenlopen met behulp van de *pijltjes-toetsen*. Een eerder ingevoerd commando kun je aanpassen en/of opnieuw uitvoeren door het indrukken van de *ENTER* toets.

Je kunt een programma stoppen door in het betreffende terminal window *CTRL-c* te tikken. Je kunt ook het hele terminal window afsluiten.

Enkele veel gebruikte Linux commando’s als je met ROS werkt:

- *cd* of *cd ~* (ga naar je home directory)
- *cd ~/catkin_ws* (ga naar de top van je workspace; hier moet zijn vaak om *catkin_make* te doen)
- *cd ~/catkin_ws/src* (ga naar de src directory van je workspace; hier staan je packages)

ROS installatie & update

Als het goed is heb je de Indigo versie van ROS geinstalleerd in Ubuntu. Open een nieuw terminal window en voer het volgende commando in om je ROS versie te checken:

```
rosversion -d
```

Actualiseer Ubuntu en ROS alsvogt:

```
sudo apt-get update && sudo apt-get -y upgrade
```

Installeer nu `git`, een handig tool voor versiebeheer en voor het downloaden van ROS projecten van GitHub:

```
sudo apt-get -y install git
```

Je bent nu klaar om je ROS oefeningen te beginnen.

Creeren van een ROS workspace

Bij het installeren van ROS heb ook je automatisch een aardig aantal standaard packages geinstalleerd. Vaak wil je echter zelf nieuwe packages aanmaken of packages van anderen hergebruiken. Je hebt daarom een zogenaamde *workspace* nodig waar je die packages dan in kunt maken of downloaden en bouwen

Creeer een ROS workspace, bijvoorbeeld *catkin_ws* (maar de naam van de workspace is vrij te kiezen). Zo kun je zo een workspace creeren, als je er nog geen hebt:

```
source /opt/ros/indigo/setup.bash
```

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ~/catkin_ws
catkin_make
```

Maak deze workspace nu je default workspace (want je kunt meerdere workspaces hebben):

```
source ~/catkin_ws/devel/setup.bash
```

Om dit commando automatisch te laten uitvoeren in elk nieuw terminal window wat je opent, kun het alsvolgt toevoegen aan je *.bashrc* file. Die staat in je home directory:

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

Als je wilt weten wat er in je *.bashrc* file staat kun je die file op je scherm printen:

```
cat ~/.bashrc
```

Je kunt deze file ook openen in een teksteditor (Ubuntu default: gedit):

```
gedit ~/.bashrc
```

Zo kun je kijken hoe de ROS omgevingsvariabelen zijn ingesteld:

```
env | grep ROS
```

Downloaden en installeren van ROS packages

Er zijn bij installatie van ROS ook een heleboel standaard packages geinstalleerd. Een daarvan is *turtlesim*, een 2D simulatie package, gemaakt voor educatieve doeleinden. Laten we eens kijken of dat package inderdaad beschikbaar is.

Is het package `turtlesim` geinstalleerd?

```
rosversion turtlesim
```

Waar is het geinstalleerd?

```
rospack find turtlesim
```

Er zijn ook heel veel packages niet geinstalleerd, maar wel beschikbaar als binary download (zogenaamde debian packages).
Welke packages zijn er beschikbaar om je ROS installatie uit te breiden?

```
 apt-cache search ros-indigo-
```

Dat zijn er erg veel! Een overzicht kun je hier vinden.

Laten we het package *arbotix_python* downloaden en installeren. We zijn niet geinteresseerd in de sources, maar willen alleen het package installeren en gebruiken. Laten we kijken of het betreffende ROS package beschikbaar is als Debian (.deb) installatiepakket. Welke debian packages zijn er met de tekst *arbotix* in hun naam?

```
apt-cache search ros-indigo- | grep arbotix
```

Blijkbaar heet het betreffende Debian pakket *ros-indigo-arbotix-python*. Nu we dit weten is installeren een koud kunstje:

```
sudo apt-get install ros-indigo-arbotix-python
```

Checken of het is gelukt:

```
rospack find arbotix_python
```

Zoals je ziet is dit pakket toegevoegd aan de ROS installatie en niet aan je workspace.

Soms willen we ROS packages downloaden die alleen in source vorm beschikbaar, of waarvan we de sources willen bekijken en wellicht aanpassen. Die packages willen we niet in binaire vorm toevoegen aan onze ROS installatie, maar downloaden in source vorm in onze workspace.

Laten we wat handige ROS voorbeeld packages van GitHub downloaden in je workspace.

```
cd ~/catkin_ws/src
git clone https://github.com/dortmans/ros_examples.git
cd ~/catkin_ws
catkin_make
```

Als het goed is kan ROS die packages nu vinden.

```
rospack find agitr
```

Je kunt ook naar de directory gaan waar dat package staat, bijvoorbeeld om wat aan te passen:

```
roscd agitr
```

En zo ga je weer terug naar je homedirectory:

```
cd
```

nodes, topics, messages

Start de node *listener* uit het package *beginner_tutorials*

```
rosrun beginner_tutorials listener
```

Waarom werkt het niet? Hoe los je dat op?

Open een nieuw terminal window (ctrl-alt-t) en voer het volgende commando in:

```
rosnode
```

Zo zie je wat je allemaal met dit commando kunt doen.

Kijk welke nodes er nu draaien:

```
rosnode list
```

Welke topics zijn er aangemaakt?

```
rostopic list
```

Bekijk de ROS Graph:

```
rqt_graph
```

Nu gaan we topics bekijken. Daar is ook een commando voor:

```
rostopic
```

Op welk topic is de node *listener* geabonneerd, d.w.z. naar welk topic luistert hij?

Welk type message mag er op dit topic gepubliceerd worden?

```
rostopic type *topic_naam*
```

Waaruit bestaat zo'n type message?

```
rosmsg show *message_type*
```

Je kunt in Linux ook de output van een commando doorgeven aan het volgende commando door middel van een *pipe* teken (|).

```
rostopic type *topic_naam* | rosmsg show
```

Probeer handmatig een message op dit topic te publiceren:

```
rostopic pub *topic_naam* *message_type* *message_inhoud*
```

Tip: gebruik de *TAB*-toets na het intikken van het message type.

Wat gebeurt er nu in het terminal waar je de *listener* node hebt opgestart?

ROS is gemaakt om robots te besturen en niet om chat programma's te maken. In de praktijk zijn messages complexer en adverteren en publiceren nodes op meerder topics. Bovendien hebben nodes vaak parameters en kunnen ze behalve via messages ook via services aangestuurd worden. Laten we daarnu eens een meer complexe node bekijken, een 2D robot simulator.

Start de node *turtlesim_node* uit het package *turtlesim*

Welke nodes draaien er nu?

Welke topics zijn er aangemaakt?

Bekijk de ROS Computation Graph.

Op welke topics is de node *turtlesim* geabonneerd? Op welke nodes publiceert hij?

Welk type message wordt er gepubliceerd op het topic */turtle1/pose*

Met het rostopic commando kun je nog veel meer bekijken. Gebruik dit commando om de volgende vragen te benatwoorden.

Met welke frequentie (rate) wordt op dit topic gepubliceerd?

Kijk eens welke messages er momenteel gepubliceerd worden op dit topic?

Wat denk je dat de gepubliceerde data betekent?

Welk type message is geassocieerd met het topic */turtle1/cmd_vel*?

Wat gebeurt er bij het volgende commando?

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist "{linear: {x: 2.0}}"
```

En wat bij dit commando?

```
rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist "{angular: {z: 1.57}}"
```

Teken met de turtle (ongeveer) een vierkant.

Wat gebeurt er als we het volgende commando uitvoeren?

```
rostopic pub -r 10 /turtle1/cmd_vel geometry_msgs/Twist "{linear: {x: 2.0}, angular: {z: 1.8
```

En wat is het verschil als je dit commando uitvoert:

```
rostopic pub -r 10 /turtle1/cmd_vel geometry_msgs/Twist '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

Tot nu toe hebben we slechts een applicatie met enkele node gebruikt. In de praktijk zal een applicatie uit meerdere nodes bestaan. We hebben de turtle laten bewegen door ROS commando's vanuit de terminal. Dat is genoeg om de node te testen maar natuurlijk geen geintegreerde applicatie. Stel we willen de turtle kunnen besturen met de pijltjestoetsen van het toetsenbord. Dan moeten nog een node opstarten, namelijk een die input van het toetsenbord omzet in besturingscommando's

Start daarom vanuit een nieuw terminal window de *turtle_teleop_key* node uit het package *turtlesim*. Je weet nu vast hoe dat moet.

Kijk in de computation graph hoe de *turtle_teleop_key* node gekoppeld is aan de *turtlesim_node*:

Bekijk met behulp van *rostopic echo* welke messages *turtle_teleop_key* publiceert als je een bepaalde pijltjestoets indrukt.

services

Nodes wisselen niet alleen informatie uit via topics. Een node can ook een dienst (service) van een andere node aanroepen. Vanuit een terminal window kunnen we zo'n service testen met behulp van het *rosservice* commando.

Welke services biedt de node *turtlesim*?

```
rosservice list
```

Welke argumenten heeft de set_pen service?

```
rosservice args /turtle1/set_pen
```

In meer detail:

```
rosservice type /turtle1/set_pen | rossrv show
```

Roep de set_pen service aan (met *rosservice call*) om de pen van de turtle rood te maken.

Beweeg de turtle om te kijken of het werkt.

Zet de pen uit. Beweeg de turtle om te kijken of het werkt.

Reset turtlesim:

```
rosservice call /reset
```

parameters

Nodes kunnen ook parameters lezen. Die parameters moeten dan wel gepubliceerd zijn voor het opstarten van de node. Dat gebeurt meestal in opstartscripts (launch files).

Welke parameters zijn er momenteel gedefinieerd?

```
rosparam list
```

Dump alle parameters in YAML formaat in een file

```
rosparam dump dump.yaml
```

Late we die file eens bekijken:

```
cat dump.yaml
```

Verander de waarde van parameter /background_b

```
rosparam set /background_b 0
```

Clear de turtlesim node zodat de parameters opnieuw ingelezen worden

```
rosservice call /clear
```

Maak nu een nieuw turtle aan:

```
rosservice call /spawn 2 2 1.57 "r2d2"
```

Welke topics zijn er nu?

```
rostopic list
```

Laat die nieuwe turtle bewegen.

Opnemen en afspelen van messages

Soms willen we alles wat er op een topic wordt gepubliceerd opslaan in een zogenaamde *bag* file om later nog eens te kunnen afspelen. Dit is erg handig voor het testen!

Laten we de messages op het topic *turtle1/cmd_vel* eens een tijdje opnemen:

```
rosbag record -O /tmp/turtle1 /turtle1/cmd_vel
```

als je wilt stoppen tik je *CTRL-C* in het terminal window waar je dit commando hebt gestart .

Reset turtlesim om met een schone lei te beginnen:

```
rosservice call /reset
```

Speel de opgenomen `bag` nu eens af:

```
rosbag play /tmp/turtle1.bag
```

Wat gebeurt er?

launch files

Tot nu toe hebben we steeds alle nodes (plus natuurlijk roscore) stuk voor stuk vanuit een nieuw terminal window moeten opstarten. Dat is natuurlijk niet handig. Gelukkig heeft ROS een mechanisme om een verzameling nodes in een keer te lanceren met behulp van het commando *roslaunch*. Welke nodes met welke parameters moeten worden opstart moeten we natuurlijk wel eerst specificeren. Zo'n specificatie heet een *launch file*.

Bestudeer de launchfiles van het *agitr* package. Met het volgende commando open je betreffende launch directory met de Ubuntu filemanager (*nautilus*):

```
nautilus `rospack find agitr`/launch
```

Probeer ze uit met het *roslaunch* programma:

```
roslaunch agitr *launchfile_naam*
```

We gaan nu zelf een nuttige launch file maken voor het opstarten van benodigde nodes om realtime beeld van een webcam op ons scherm te laten zien. Tegenwoordig hebben veel laptops zo'n webcam boven aan het scherm zitten voor b.v. Skype.

Installeer een USB camera driver package:

```
sudo apt-get -y install ros-indigo-usb-cam
```

Dit package bevat een node voor de aansturing van de USB webcam.

We hebben ook een node nodig die images kan lezen en op het scherm kan ztten. We gebruiken daarvoor de *image_view* node van het *image_view* package. Dit package is al standaard geinstalleerd.

We willen nu beide nodes tegelijk opstarten met de benodigde parameters. Daarvoor moeten we een launch file maken. We zullen eerst een nieuw package creeren om die launchfile in te zetten. Laten we het *usb_camera* noemen.

```
cd ~/catkin_ws/src
catkin_create_pkg usb_camera std_msgs rospy roscpp
cd ~/catkin_ws
catkin_make
```

Even checken of ROS ons package nu kan vinden:

```
rospack find usb_camera
```

In ons nog vrij lege package maken we een directory *launch*. Daar gaan we onze launch file in zetten.

```
roscd usb_camera
mkdir launch
```

Waarschijnlijk handiger om nu deze directory met de Ubuntu filemanager te openen:

```
nautilus `rospack find usb_camera`/launch
```

Maak nu in deze launch directory een nieuwe file aan met de naam *usb_camera.launch* en geef deze file de volgende inhoud:

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="mjpeg" />
    <!-- <param name="pixel_format" value="yuyv" /> -->
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen"
    <!-- <remap from="image" to="/usb_cam/image_raw"/> -->
    <param name="autosize" value="true" />
  </node>
</launch>
```

Let op: meestal is het webcam device in Ubuntu geregistreerd als */dev/video0*, maar het kan ook bijvoorbeeld */dev/video1* zijn.

Let op: niet alle webcams zijn hetzelfde. Soms moet je als pixel_format *mjpeg* gebruiken en soms *yuyv*. Even uitproberen dus.

Test de launchfile:

```
roslaunch usb_camera usb_camera.launch
```

Bekijk de computation graph

```
rqt_graph
```

Welk structureel probleem zie je? Los het op door de launch file aan te passen. Als het je lukt kun je mooi naar jezelf kijken ;-)

rqt

Veel zaken die je kunt doen via losse commando's kun je ook doen via het programma *rqt*, de grafische user interface (GUI) van ROS. Dit programma kan ook de robot en verschillende messages visualiseren.

Start de *rqt* GUI:

```
rqt
```

Verken de verschillende plugins.

Enkele plugins (b.v. rqt_graph, rqt_image_view) ken je al want die hebben we los opgestart en gebruikt. Binnen rqt zijn alle plugins mooi geintegreerd.

Besturen van een echte mobiele robot

Tot nu toe hebben we alle nodes op onze eigen laptop gedraaid. Meestal draait een gedeelte van de ROS nodes echter op de robot zelf. De nodes op onze laptop moeten dan via het netwerk (b.v. Wifi) communiceren met de nodes die op de robot draaien.

ROS is gemaakt om gedistribueerd te draaien. Wel moet je weten waar de ROS Master node draait. Meestal draait die op de robot. Het is voldoende om de omgevingsvariabelen ROS_IP en ROS_MASTER_URI aan te passen. In ROS_IP moet het IP_ adres van je laptop komen. Dit kunnen we opvragen via het commando `hostname -I`. Let op: je laptop kan meerder IP adressen hebben (want elke actieve netwerk interface krijgt een eigen IP adres). Je kiest dan die van het netwerk waarmee je met de robot bent verbonden. Heb je maar een IP adres dan is het simpel:

```
export ROS_IP=`hostname -I`
```

Anders moet je zelf het goede IPadres invullen:

```
export ROS_IP=*IP_adres_van_je_laptop*
```

Nu moet je nog instellen hoe jouw laptop nodes de ROS Master op de robot kunnen bereiken:

```
export ROS_MASTER_URI=http://*IP_adres_van_de_robot*:11311
```

Deze exports kun je ook in je *.bashrc* zetten, dan worden ze automatisch uitgevoerd als je een nieuw terminal window opent:

```
gedit ~/.bashrc
```

Kijk of je met de ROS Master (die op de robot draait) kunt communiceren:

```
rostopic list
```

Het besturen van de robot gaat nu exact hetzelfde als bij de gesimuleerde robot in turtlesim.

Laat de robot door middel van een *rostopic pub* commando een rondje draaien.

Start het volgende handige programma om de robot met je muis te besturen:

```
arbotix_gui
```

Dit programma leest je muis acties (bewegen van de rode stip) en vertaalt ze in Twist messages op het cmd_vel topic.

De robot heeft ook een camera, zodat we zien waar we rijden. Maak het camerabeeld zichtbaar view de *Image View* plugin van rqt.

Open ook eens de *RViz* plugin van rqt en voeg een RobotModel display toe (gebruik Add). Je ziet dan de robot model gevisualiseerd. Daar komen we o.a. in volgende sessies op terug.

Referenties

- A Gentle Introduction to ROS
- ROS Tutorials
- YAML on the ROS command line
- rosparam
- roslaunch
- Testing: ROS USB Camera drivers
- usb_cam
- image_view
- rqt