2. Hogyan kezdjünk el a ROS1-gyel dolgozni?

Kiegészítő anyag

Robot operációs rendszerek és fejlesztői ökoszisztémák

BMEVIIIAV55

Összeállította: Gincsainé Szádeczky-Kardoss Emese

szadeczky.emese@vik.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem

Irányítástechnika és Informatika Tanszék

2024

Telepítés

A ROS (Robot Operating System) a neve ellenére nem egy operációs rendszer, ezért először a megfelelő OS-ről kell gondoskodnunk. A különböző ROS disztribúciók különböző operációs rendszerekkel használhatók, ezekről a honlapon lehet tájékozódni (https://www.ros.org/).

ROS1-ből már csak a Noetic Ninjemys a támogatott verzió, ehhez az Ubuntu 20.04 (Focal) operációs rendszer használata ajánlott. A további támogatott operációs rendszer és a telepítési útmutatók szintén elérhetők a hivatalos honlapon: http://wiki.ros.org/noetic/Installation. Ha számunkra az a kényelmesebb, akkor virtuális gépre is telepíthető a megfelelő operációs rendszer és a kiválasztott ROS disztribúciónk (pl. VMware Workstation).

Érdemes még egy fejlesztői környezetet is telepíteni (pl. Visual Studio Code), és megfelelően konfigurálni, hozzáadni a fejlesztést segítő kiegészítőket. Továbbá szükség van még egy compiler-re is (pl. build-essential package).

Ahhoz, hogy parancssorból elérhessük a ROS szolgáltatásait, a következő utasításra van szükségünk minden egyes új terminál ablak megnyitásakor:

```
ros_user@ubuntu:~$ source /opt/ros/noetic/setup.bash
```

Ha nem akarjuk ezt minden újabb terminál megnyitásakor begépelni, a `.bashrc` állomány végére is beszúrhatjuk:

```
ros_user@ubuntu:~$ echo "source /opt/ros/noetic/setup.bash" >> .bashrc
```

Ha majd saját package-eket akarunk futtatni, akkor a fentiekhez hasonlóan be kell majd állítani a munkakörnyezetet.

Első próbálkozások

A telepítés ellenőrzése és ismerkedés gyanánt érdemes pár node-ot futtatni, például egy terminálban indítsunk egy ROS Mastert:

```
ros_user@ubuntu:~$ roscore
```

A második terminálban a `turtlesim` package-ből a `turtle_sim` node futtatható:

```
ros_user@ubuntu:~$ rosrun turtlesim turtlesim_node
```

A harmadik terminálban ugyanebből a package-ből a `turtle_teleop_key`-t is indítsuk el:

```
ros_user@ubuntu:~$ rosrun turtlesim turtle_teleop_key
```

Ha utóbbi terminálban a billentyűzet nyilait nyomkodjuk, a kis teknős mozog a `turtle_sim` node grafikus felületén. (Leállítani a node-okat például a Ctrl+C billentyűkombinációval lehet.)

Catkin workspace és saját package létrehozása

Ha saját package-eket akarunk létrehozni, ahhoz szükség van egy úgynevezett Catkin munkatérre (pl. `catkin_ws`). A package-ek a munkatéren belül egy `src` nevű könyvtárba kerülnek majd. Ezért hozzuk létre ezeket a mappákat:

```
ros_user@ubuntu:~$ mkdir -p ~/catkin_ws/src
```

A munkatér könyvtárába belépve érdemes a – még üres – munkateret buildelni a `catkin_make` utasítással. Figyeljünk arra, hogy ezt az utasítást mindig a catkin workspace mappából (`catkin_ws`) adjuk ki!

```
ros_user@ubuntu:~$ cd ~/catkin_ws
ros_user@ubuntu:~/catkin_ws$ catkin_make
```

A build során az `src` mellé új könyvtárak jöttek létre. Ezeket kilistázhatjuk:

```
ros_user@ubuntu:~/catkin_ws$ ls
```

A `devel` mappába is érdemes belenézni. Itt is van egy `setup.bash` nevű állomány. Ezt kell source-olni minden megnyitott terminálban, ha ott a saját package-ből szeretnénk állományokat futtatni:

```
ros_user@ubuntu:~/catkin_ws$ ls devel
ros_user@ubuntu:~/catkin_ws$ source devel/setup.bash
```

Ha nem szeretnénk ezt a sort minden megnyitott terminálba begépelni, akkor itt is működik, hogy kiegészítjük a `.bashrc` állományt:

```
ros_user@ubuntu:~/catkin_ws$ cd ..
ros_user@ubuntu:~$ echo "source ~/catkin_ws/devel/setup.bash" >> .bashrc
```

A catkin munkatér `src` mappájából létre tudunk hozni egy új package-et. Ennek meg kell adni a nevét, és hogy milyen további package-eket, library-ket használ (dependencies). Hozzunk most létre egy `my_pkg` nevű package-et, amiben használni fogjuk a `roscpp` szolgáltatásait:

```
ros_user@ubuntu:~ $ cd catkin_ws/src
ros_user@ubuntu:~/catkin_ws/src$ catkin_create_pkg my_pkg roscpp
```

Az utasítás hatására létrejöttek új fájlok (`package.xml`, `CMakeLists.txt`) és új mappák (`include`, `src`). Tegyünk eleget a megjelenő kérésnek, és nézzük meg a manifest fájlt:

```
ros_user@ubuntu:~/catkin_ws/src$ gedit my_pkg/package.xml
```

A fájl elején meg lehet adni számos adatot, az állomány végén pedig a függőségek szerepelnek. Ha a package létrehozásakor (`catkin_create_pkg` utasítás) esetleg nem adtunk meg minden szükséges függőséget, akkor azt később itt lehet beállítani.

Ha C++ nyelven szeretnénk node-okat implementálni, akkor a forráskódot a `my_pkg/src` könyvtárába mentsük. A `CMakeLists.txt` állományt is módosítani kell ilyenkor. Ha kész vagyunk a forráskóddal, build-elni a már korábban látott `catkin_make` utasítással kell a `catkin_ws` könyvtárból:

```
ros_user@ubuntu:~/catkin_ws/src$ cd ..
ros_user@ubuntu:~/catkin_ws$ catkin_make
```

Az implementáció, fordítás, futtatás részleteit a következő alkalomhoz készített kiegészítő anyag tartalmazza.