Mobil gépek mechatronikája

Doba Dániel Károly
Közlekedés- és Járműirányítási Tanszék
2025. szeptember 15.

Robot Operating System - ROS

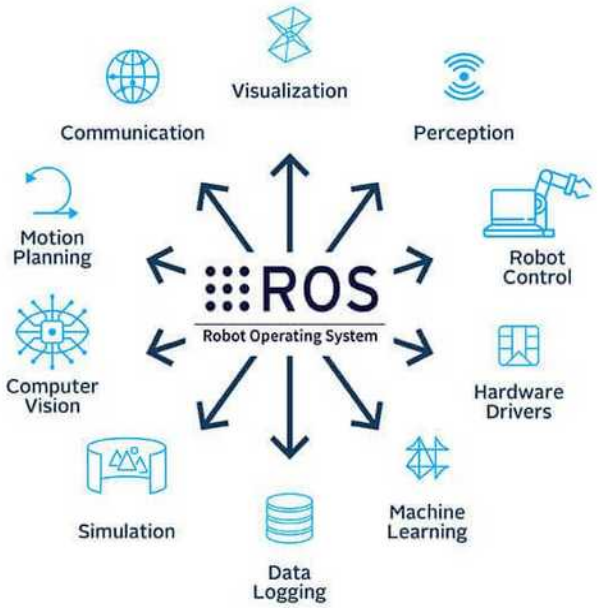
- Mi az a ROS?
- ROS története
- ROS koncepció
- Támogatott program nyelvek
- Fájl rendszer és kód fordítás
- ROS elemek
  - Node
  - Topic
  - Launch fájlok
- Rosbag
- Időbélyegek, időzítések
- Vizualizáció
- Multimachine, multimaster - Hálózati kapcsolatok
- Transformáció - tf2
- Robotok leírása URDF formában, xacro
- Gazebo
- Dinamikus konfigurálás
- ROS robotok

Mi az a ROS?

- A ROS nem egy valódi operációs rendszer.
- A ROS egy ún. „middleware” szoftverkörnyezet, amit elsődlegesen robot fejlesztések felgyorsítására hoztak létre. A ROS biztosítja az alapvetően szükséges elemeket moduláris formában.
- A ROS fő erőssége a nyílt forráskód, és kifejezetten bátorít a kód újrahasználhatóságra.
- A ROS-t elsődlegesen kutatás-fejlesztési környezetben hasznosítják, de egyre több az ipari alkalmazás.

ROS koncepció

1. Csövezetékelés - Plumbing: egyszerűsített kommunikációs megoldás, könnyebb skálázhatóság .
2. Eszközök - Tools: adott rendszerek konfigurálására, indítására és megállítására, monitorozására, megjelenítésére, tesztelésére, mérési adatok tárolására.
3. Képességek - Capabilities: nagy mennyiségű könyvtár tartozik hozzá, mellyek fókusza a különböző robot műveletek - mozgás, érzékelés, beavatkozás - megvalósítása.
4. Ökoszisztéma - Ecosystem: Nagy a közösségi támogatotsága, jelentős hangsúlyt helyezve az integrációra és a dokumentációra.

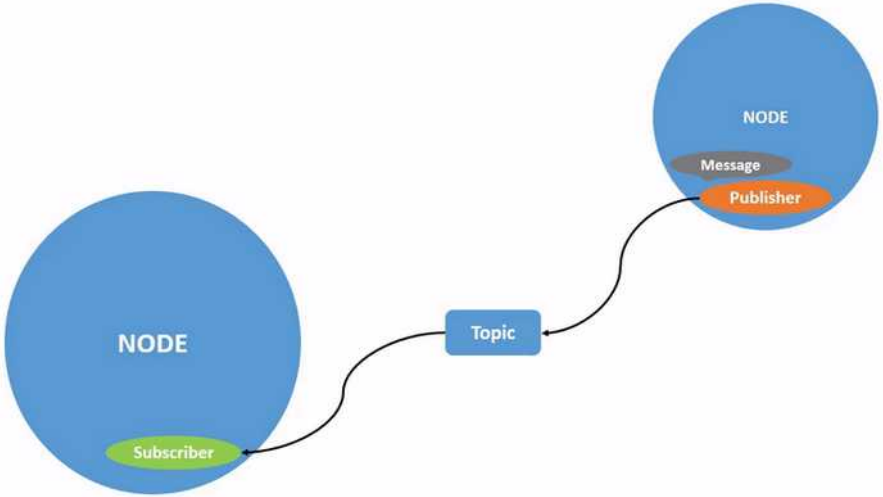


ROS történelem

- A ROS alapjait Standford-i Egyetem Phd hallgatói fektették le.
- 2007-ben az amerikai Willow Garage átvette fejlesztést és teljes értékű framework-é nőtte ki magát.
- 2010-ben került kiadásra az első hivatalos verzió, majd félévente új főverziók követték.
- 2013-ban Open Source Robotics Foundation (OSRF) átvette a karbantartási és fejlesztési feladatokat, és ekkor állnak át évenkénti frissítési ciklusra. Minden második évben kiadott verziók az Ubuntu Linux rendszerekhez hasonlóan kibővített támogatási ciklussal rendelkeznek, mely 5 évet jelent.
- 2017-től az OSRF Open Robotics néven fut tovább.
- 2017-ben kiadták a ROS 2-t, aminek az egyik legfontosabb újítása, hogy támogat valós idejű megoldásokat.
- 2020-ban kiadták az utolsó ROS 1 verziót, a Noetic Ninjemys-t, amely 2025-ig volt támogatott.
- A továbbiakban a ROS 2-ből lesznek új verziók.

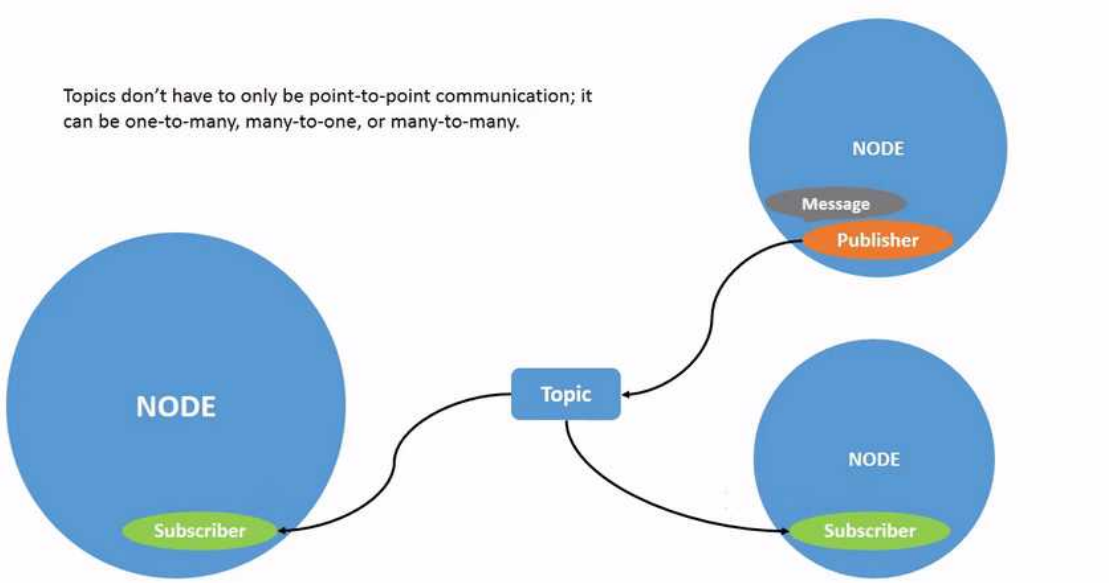
ROS alkotó elemek I.

- A **node**-k egyedi, egymástól független folyamatok, melyek a rendszert alkotják. Egy node általában egy jól körül határolt feladatért felel ezzel is segítve egy moduláris rendszer kialakítását. Minden node saját paraméter szerverrel rendelkezik, melynek kezelésére külső forrásból is van lehetőség.
- A **topic**-ok a node-k közötti kommunikációs csatornák azonosítója, melyek adott üzenet leírással/definícióval rendelkeznek. Az üzenet küldő regisztrálja a topic-ot a többi node-nál és az esemény vagy idő vezérlésnek megfelelően küldi az üzeneteket. A fogadó fél feliratkozik az adott topic-ra és képessé válik az üzenetek fogadására. Lehetőség van több küldő és fogadó node definiálására is ugyanarra a topic-ra.
- A gyakorlatban az üzent továbbítás peer-to-peer kommunikációt jelent, a kommunikáció kezelését egy **Data Distribution System (DDS)** rendszer felel. A DDS valósítja meg, hogy a kommunikációnak minőségi jellemzői legyenek: megbízhatóság, buffer, újraküldési lehetőség, stb.
- A **/rosout** egy rendszer szintű logger topic.



Forrás: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

ROS alkotó elemek II.

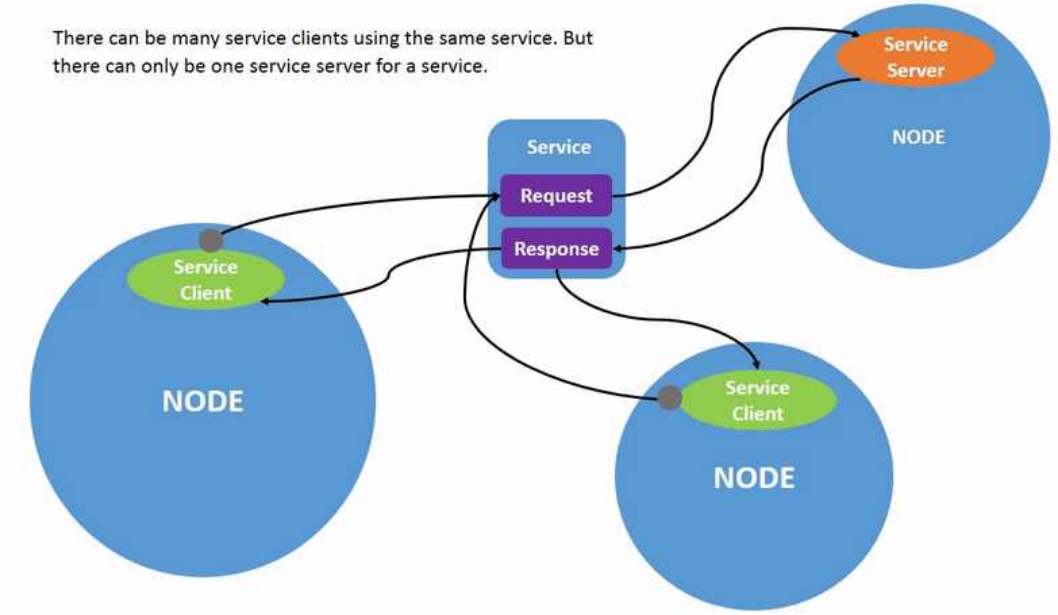


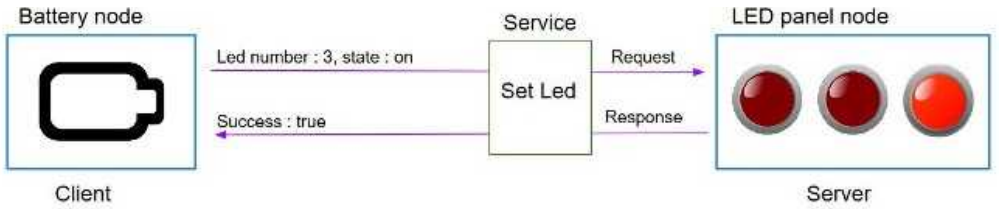
Forrás: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

ROS alkotó elemek III.

- A **service**-k olyan speciális node-k, melyek képesek közvetlen kapcsolatot kialakítani a kérelmező féllel és egy kérés-válasz kommunikációs eseményt lefolytatni. Ezzel lehetőség van az adott kérés és a válaszként jött üzenet közötti kapcsolat definiálására.

  Hátrányuk, ha a válasz üzenetre sokat kell várni, akkor a kérelmező fél folyamata áll.

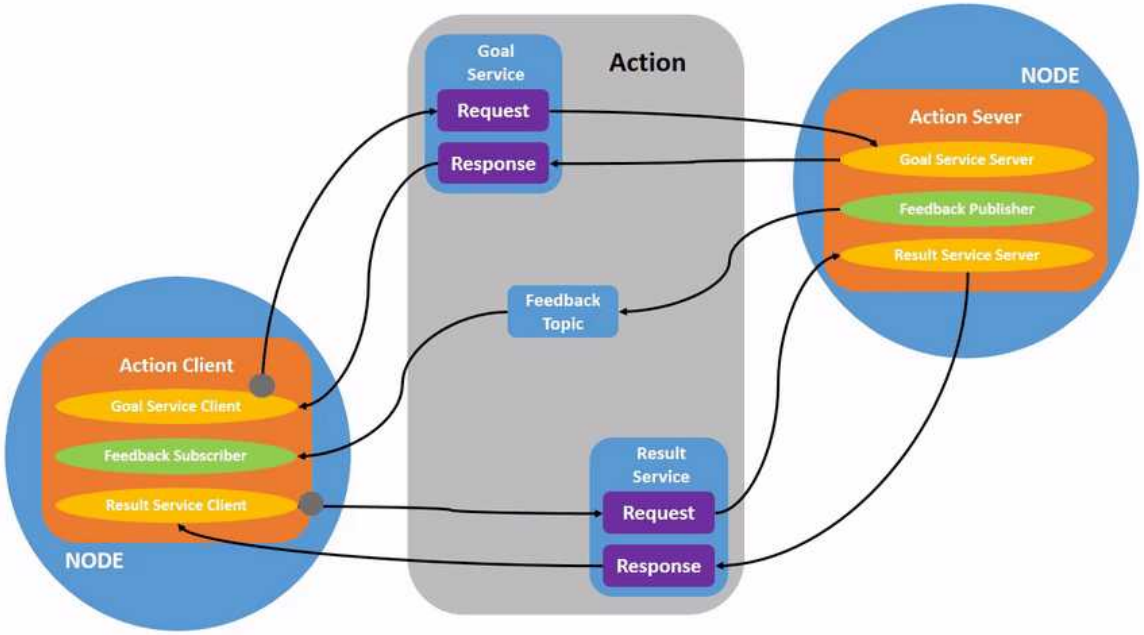




Forrás: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

ROS alkotó elemek IV.

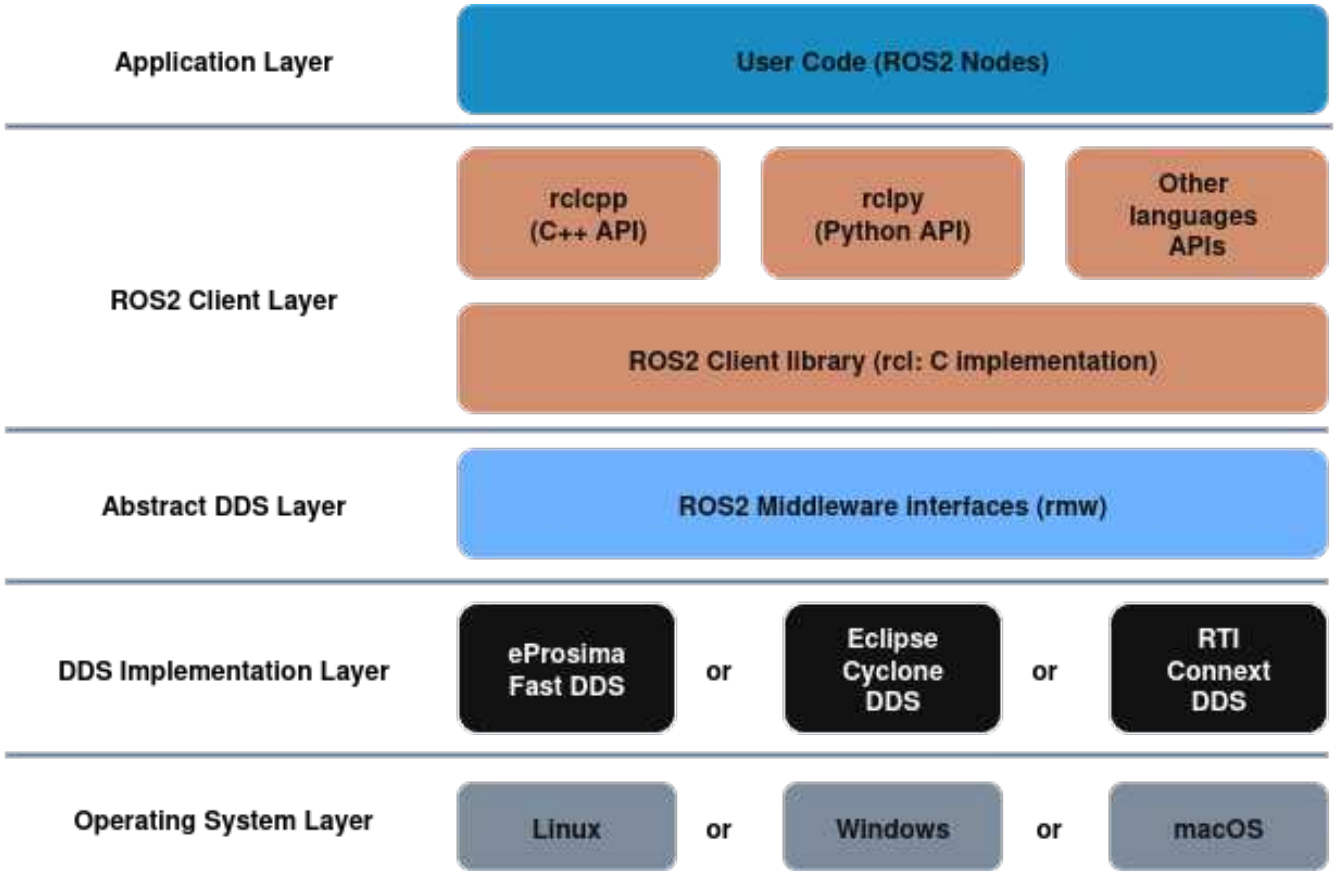
- Az **action**-k kiküszöbölik az információ nélküli várakozást, és van lehetőség indikátor üzenetek definiálására, melyeket többek között a kért folyamat állapotának indikálására lehet felhasználni.
  Ez aszinkron kommunikációt jelent, de a válasz esetében egyértelműen felállítható a kapcsolat.
  További előny, hogy van lehetőség arra, hogy akár a kérő oldal képes legyen visszavonni a kérést.



Forrás:
https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html

Architektúra és támogatott nyelvek

- Támogatott nyelvek :
  - C++ 17
  - Python 3
  - Matlab/Simulink
  - Ada, C. Java és Android, .NET, Javascript, Rust, Flutter, Dart
- Ezek közül a pirossal jelöltek, amiket a fejlesztők közvetlenül támogatnak.
- Az architektúra kialakításával igyekeztek a különböző nyelvek implementációit egységesíteni. Maradt egy-két nyelv specifikus megoldás implementációs szinten.
- A ROS2 hálózat egyes elemei programozási nyelvtől függetlenül képesek kommunikálni egymással.



Forrás:
https://docs.ros.org/en/humble/Concepts/Basic/About-Client-Libraries.html
https://docs.ros.org/en/humble/Concepts/Advanced/About-Internal-Interfaces.ht

Data Distribution Service - DDS

- Cél: a megbízható, valós idejű és jól skálázható kommunikáció.
- Adat központú küldés-fogadás
- Decentralizált, jól skálázható működés
- Alacsony latency
- Kommunikációs megoldások:
  - UDP
  - TCP
  - SHM – shared-memory

- QoS - Quality of service:
  - Megbízhatóság
  - Perzisztencia
  - Bufferelés
  - Utolsó üzenet érkezésétől eltelt idő
  - Üzenet élettartam
  - Kapcsolat élettartam

Forrás: https://design.ros2.org/articles/ros_on_dds.html
https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Quality-of-Service-Settings.html

Fájl rendszer

- /opt/ros/... : ROS elemek telepítési mappája, ahová a ROS PPA-ból (Personal Package Archive) települnek a különböző komponensek. Ezeket szükséges hozzáadni .bashrc-hez: “source /opt/ros/humble/setup.bash”
- workspace/:
  - build/: a fordítási folyamat során keletkezett fájlok
  - install/: a node-k, header-ek, üzenet típusok fordított fájljainak és hivatkozásainak a helye. Továbbá minden konfigurációs és script fájl ide **MÁSOLÓDIK BE(!)**.Az itt található setup.bash fájlt is hozzá kell adni az elérési úthoz a fenti módon.
  - src/:

    Package(s)/:
    - include/: c++ header fájlok helye
    - src/: forrás kód fájlok helye, általában c++/python
    - msg/: egyedi üzenetek definíciója
    - script/: script fájlok helye, általában python/bash fájlok
    - launch/: launch fájlok helye
    - CMakeLists.txt: a fordítási és telepítési információkat tartalmazza az adott szoftver csomagra nézve.
    - package.xml: meta információkat, függőségi viszonyokat ír le, melyeket a fordítási folyamat során használ fel a ROS.

Node

Ament és colcon

- Az **ament** a ROS2-hoz tartozó, de attól független, CMake makrókra és Pythonra épülő fordító rendszer. Ennek parancssori interface-e a **colcon**.
- Egyszerűsíti a workspace szintű alkalmazás fordítási és kezelési folyamatokat. Pl.: fájlrendszer.
- ***colcon build*** :
  - adott workspace-n található összes package felderítése és lefordítása
  - package.xml alapján egymásra épülési sorrend meghatározása, fordítási sorrend hozzáigazítása.
  - a beállított fájlok másolása az install/ mappába
- ***colcon build –symlink-install*** *:*
  - A beállított fájlok symbolic-linkelésre kerülnek az install mappa megfelelő helyére.

Package terminal parancsok

- ROS2 alatt a szoftverek „package”-ekbe vannak szervezve. Ezek tartalmazhatnak node-okat, könyvtárakat, dataseteket, konfig fájlokat, üzenet típusokat stb.

- ***ros2 pkg –h***: rövid leírás arról, hogy hogyan lehet használni a *ros2 pkg* parancsot.
- ***ros2 pkg list***: kilistázza az aktuálisan elérhető package-k nevét és helyét.
- ***ros2 pkg prefix [package name]***: kiírja az elérési útját az adott package-nek.
- ***ros2 pkg create --build-type [ament_cmake/ament_python] [package name]***: egy package generálása
-

Node terminal parancsok

- ***ros2 node –h***: rövid leírás arról, hogy hogyan lehet használni a *rosnode* parancsot.
- ***ros2 node list [-a]***: kilistázza az aktív node-ok nevét és elérhetőségi pontját.
- ***ros2 node info* [node_name]**: kilistázza az alapvető információkat az adott node-dal kapcsolatban, pl.: küldött és fogadott topic-ok.

- ***ros2 run [package_name] [node_type]***: egy node elindítására alkalmas, egyedi név generálódik.
  - ros2 run writer_package writer.py
  - ros2 run writer_package writer_node

Topic I.: Üzenet felépítés

Az üzenetek különböző mezőket tartalmazhatnak, melyek típussal és azonosítóval vannak ellátva.

- Alap építő típusok:
  - int8, int16, int32, int64 (plus uint*)
  - float32, float64
  - bool
  - string
  - time, duration
  - Más üzenet definíciók
  - Array: [] vagy [x], ahol x a méretét határozza meg. Python-ban ezek az elemek listaként, c++-ban std::vector-ként vannak értelmezve.
- std_msgs/Header üzenet:
  - **time** *stamp*
  - **string** *frame_id*
- nav_msgs/Path üzenet:
  - **std_msgs/Header** *header*
  - **geometry_msgs/PoseStamped[]** *poses*

Topic II.: Terminal parancsok

- ***ros2 topic –h***: rövid leírás arról, hogy hogyan lehet használni a *ros2 topic* parancsot.
- ***ros2 topic list***: kilistázza a regisztrált topic-okat
- ***ros2 topic echo* [topic_name]**: az adott topic tartalmát jeleníti meg
- ***ros2 topic hz* [topic_name]**: kiértékel egy adott üzenet fogadási gyakoriságát egy adott vizsgáló alak mentén. Az alapértelmezett vizsgáló ablak mérete 50000 db üzenet.
- ***ros2 topic info* [topic_name] -v**: az adott topicról kilistázza az alábbi információkat: típus, küldő és fogadó felek nevét
- ***ros2 topic pub* [topic_name] [topic_type] [data]**:
- ***ros2 topic bw* [topic_name]**: egy becslést ad topic által felhasznált sávszélességre, függ az elérhető hálózattól, valamint attól, hogy képes-e lépést tartani az üzenet mennyiségétől.
- ***ros2 interface show* [topic_type]**: megjeleníti az adott típus típusdefinícióját.

Launch I.

- Launch fájl: egy konfigurátor fájl, mely az adott teszt esetre tartalmaz egy vagy több node konfigurációját XML vagy python formában.

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="false"/>

  <node pkg="py_pubsub" exec="talker" name="talker"/>

  <node pkg="py_pubsub" exec="listener" name="listener">
    <param name="use_sim_time" value="$(var use_sim_time)"/>
  </node>

  <group unless="$(var use_sim_time)">
    <include file="$(find-pkg-share py_pubsub)/launch/system.launch.py">
      <arg name="use_sim_time" value="$(var use_sim_time)"/>
    </include>
  </group>

</launch>
```

Launch II.: python launch


```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.substitutions import LaunchConfiguration

def generate_launch_description():
  return LaunchDescription([
    Node(
      namespace='test',
      package='py_pubsub',
      executable='talker',
      name='talker',
      parameters=[{'use_sim_time': LaunchConfiguration('use_sim_time')}],
      output='screen'
    ),

    Node(
      namespace='test',
      package='py_pubsub',
      executable='listener',
      name='listener',
      parameters=[{'use_sim_time': LaunchConfiguration('use_sim_time')}],
      output='screen'
    ),
  ])
```

Paraméterek

- Minden node-nak saját paraméter szervere van
- Többi node felé nyitott információ áramlás a meglévő paraméterek írására/olvasására
- Nincs interface/callback a paraméter változások kezelésére
- Nem deklarált paraméterek létrehozása korlátozott

Idő, időbélyeg, időzítés

- A számítógép rendszer óráját használja, mely a szabványosított Unix-idő formátumot használja. Kezdő időpontjától, UTC szerinti 1970. Január 1. 00:00:00.0 óta eltelt másodperceket számolja. Ez a ROS-ban az ún. ”wall-time” vagy „wall-clock”
- Szimuláció vagy rosbag visszajátszás esetén van lehetőség egy új idő forrás definiálására, ebben az esetben minden node a /clock topicból veszi az időt.
  - `<param name="/use_sim_time" value="true"/>`
- Bármelyik node publisholhat a /clock topicba, de egyszerre csak egy
  - ros2 bag play --clock a felvett időt küldi a /clock topicba

Feladat	Python

Time osztály	`now1 = node.get_clock().now()`
	`now2 = node.get_clock().now()`
	`seconds = now1.nanoseconds / 1e9`

Duration osztály	`d = rclpy.Duration(3600)`
Rate osztály	`r = node.create_rate(10)` `r.sleep()`

Timer osztály	`def timer_callback(self):`
	`  pass`
	`self.create_timer(d.nanoseconds / 1e9, self.timer_callback)`

Rosbag

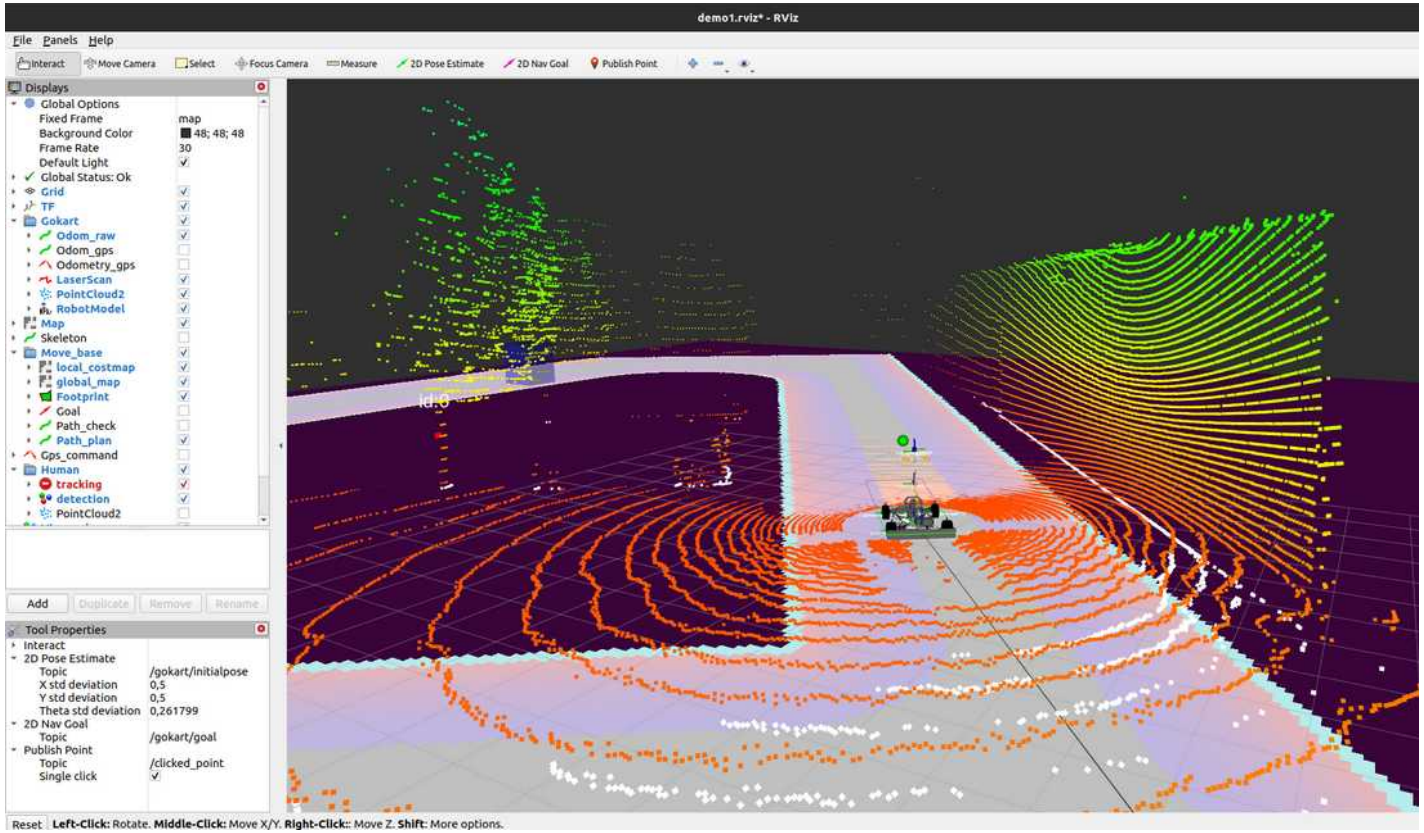
- Egy eszköz, melynek segítségével a node-k közötti kommunikáció időbélyeggel ellátva felvehető és később az visszajátszható.
- Javasolt tömörítés .7zip formátumba vagy a *rosbag compress* használatával.

- ***ros2 bag record [-a]/[topic_name] -o [filename]***: adott vagy összes([-a])topicra felcsatlakozva minden üzenetet elment egy adott fájlba.

- **ros2 bag play [filename]**: visszajátssza az adott file tartalmát tartva az relatív időbeliséget.
  - --clock: az eredeti időbélyeget publisholja a /clock topicba.
  - -l: periodikusan újra akarjuk játszatni ugyanazt a bag fájlt.
  - --topic [topic_name]: ha csak adott topic-okat akarunk lejátszatni.

Vizualizáció

- ***ros2 run rviz2 rviz2***: vizualizációs alkalmazás
- ***ros2 run rqt_graph rqt_graph***: aktív node-k és topic-ok közötti kapcsolat rendszerből épít egy kapcsolati gráfot
- ***ros2 run rqt_plot rqt_plot***: adott topic adott jelének időbeli megjelenítése
- ***ros2 run plotjuggler plotjuggler***: adott topic adott jeleinek megjelenítése, Matlab szintű plotok készítése
- ***Floxglove***



Topic: Quality of Service

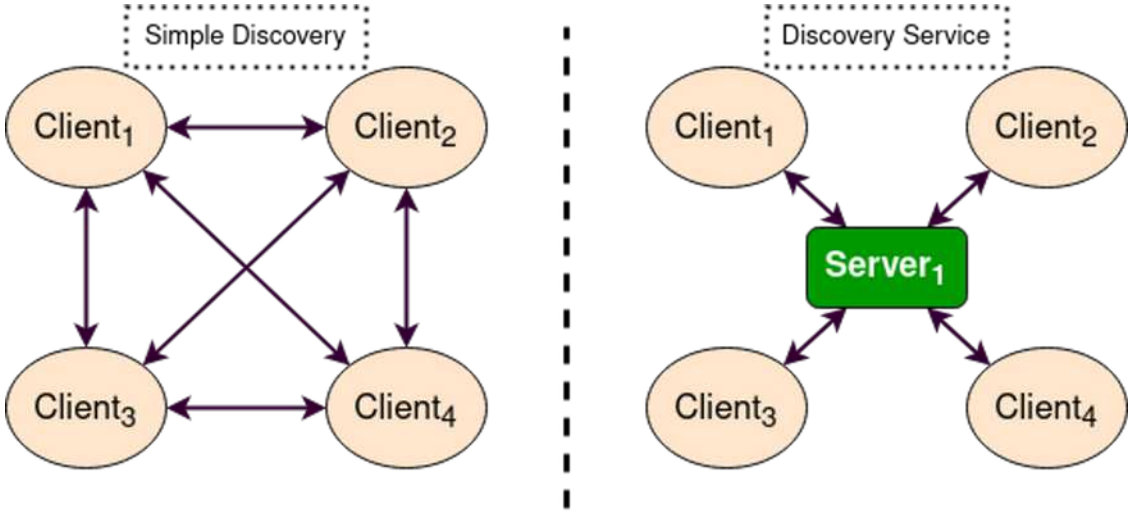
- History:
  - **Keep last**: az utolsó N darab eltárolása
  - **Keep all**: az összes üzenet eltárolása
- Reliability:
  - **Best effort**: lehet, hogy elveszik az üzenet
  - **Reliable**: újra próbálkozik az üzenet küldéssel
- Durability:
  - **Transient local**: új subscriber megkapja az utolsó elküldött üzenetet is
  - **Volatile**: új subscriber csak új üzeneteket kapja meg

- **Deadline**: Minimum üzenet küldési gyakoriság egy topic-ra
- **Lifespan**: Maximum várható eltelt idő a küldés és fogadás között
- **Liveliness**: Van-e adat forgalom egy adott idő ablakon belül

Forrás:
https://docs.ros.org/en/rolling/Concepts/Intermediate/About-Quality-of-Servi
ettings.html

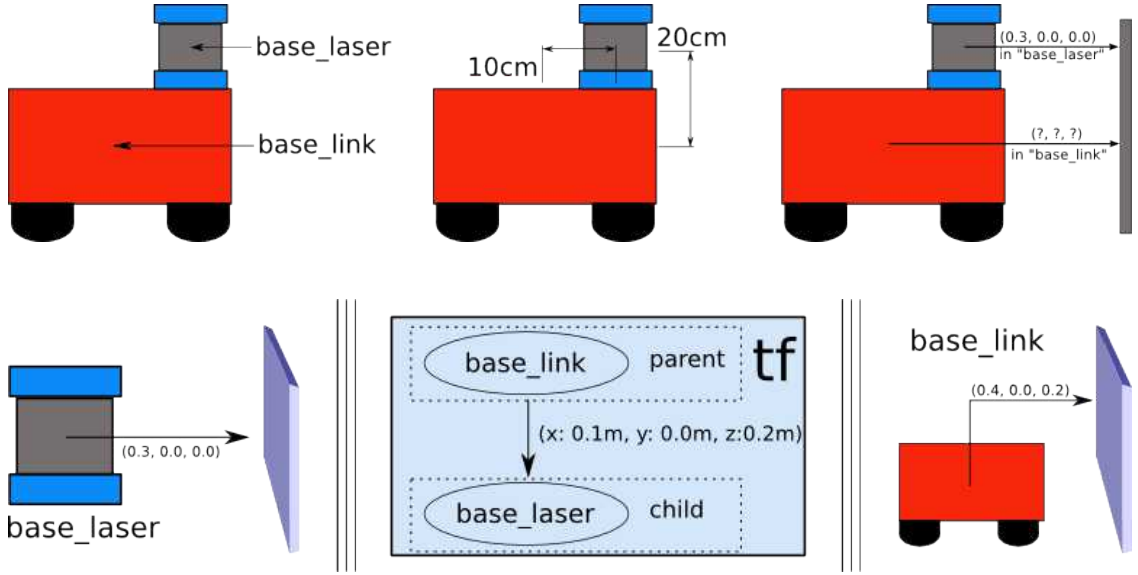
ROS2 környezetek közötti kapcsolat

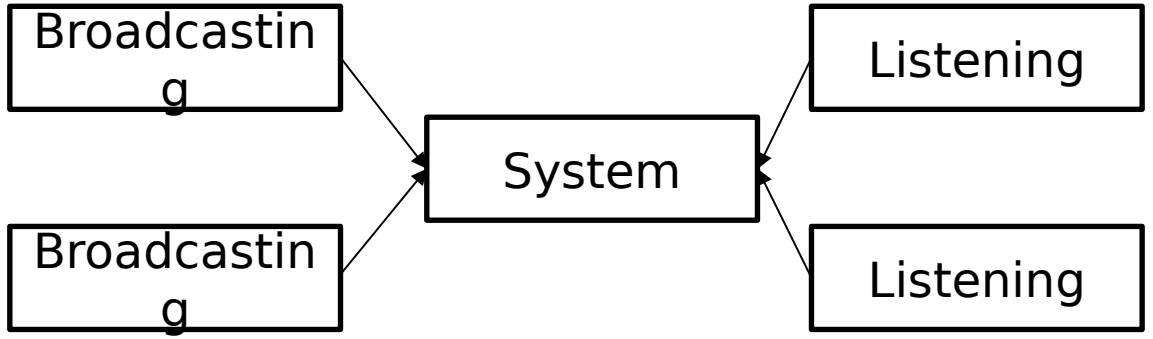
- Domain ID:
  - Forrás: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Domain-ID.html
- Dinamikus feltérképezés:
  - Forrás: https://docs.ros.org/en/rolling/Tutorials/Advanced/Improved-Dynamic-Discovery.html
- Discovery server:
  - Forrás: https://docs.ros.org/en/humble/Tutorials/Advanced/Discovery-Server/Discovery-Server.html



ROS – Transzformációk: tf2 könyvtár

- A transzformációk célja a kapcsolatbiztosítása a különböző koordinátarendszerek között, azaz pontok vagy vektorok koordinátatranszformációja.
- Tf2 a második generációja a transzformáció könyvtár, ami képes időben nyomon követni egyszerre több koordináta rendszer relatív viszonyát.
- A tf2 fa struktúrában kezeli az egyes koordináta rendszerek közötti kapcsolatokat, így a tér bármely pontja egyszerűen leírható bármelyik koordináta rendszer szerint bármely múltbeli időpontban. A fa struktúra alapján egy koordináta rendszernek csak egy szülője lehet.



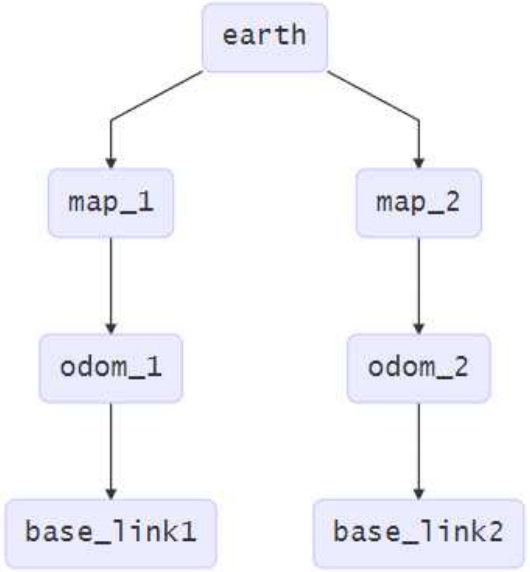


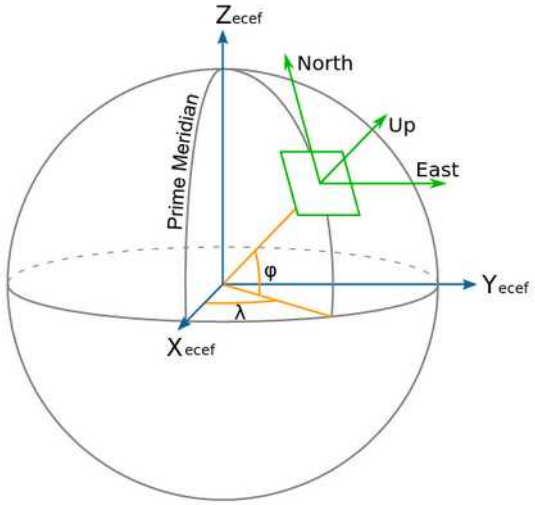
Forrás: http://wiki.ros.org/tf#static_transform_publisher

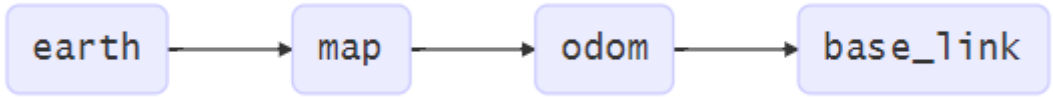
ROS – Tf2 II.: koordináta-rendszerek I.

Általános nevezékek:

- earth: Eath Centered, Earth Fixed (ECEF) koordináta rendszer, pl.: ECEF/GPS koordináták(WGS84 rendszerWGS'84)
- map: egy adott területet reprezentáló, fix koordináta rendszer, pl.: ENU koordináták, kész térkép(UTM rendszer)
- odom: a robot **vélt** mozgását leíró, fix koordináta rendszer
- base_link: az adott robot bázispontja, általában tömeg középpont, vagy valamelyik tengely középpont, a robothoz mereven rögzített

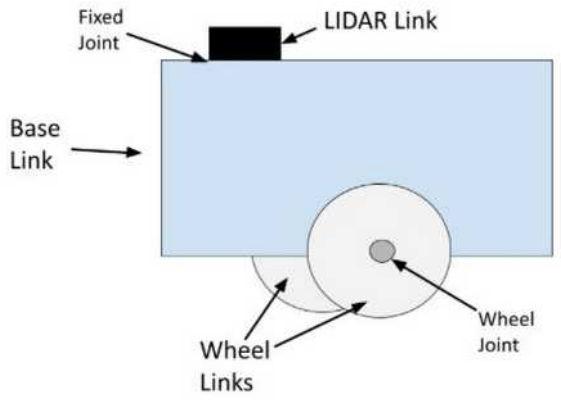


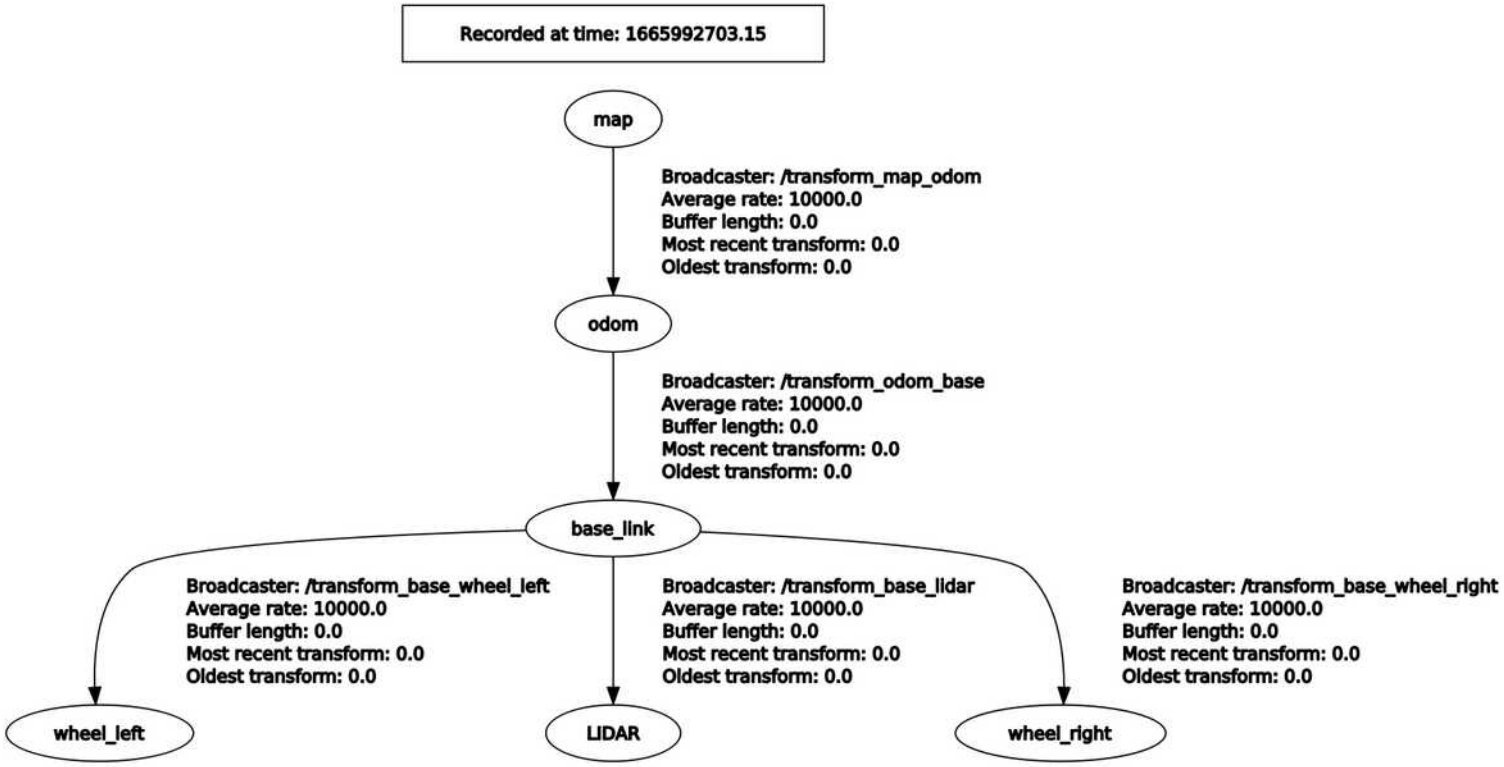




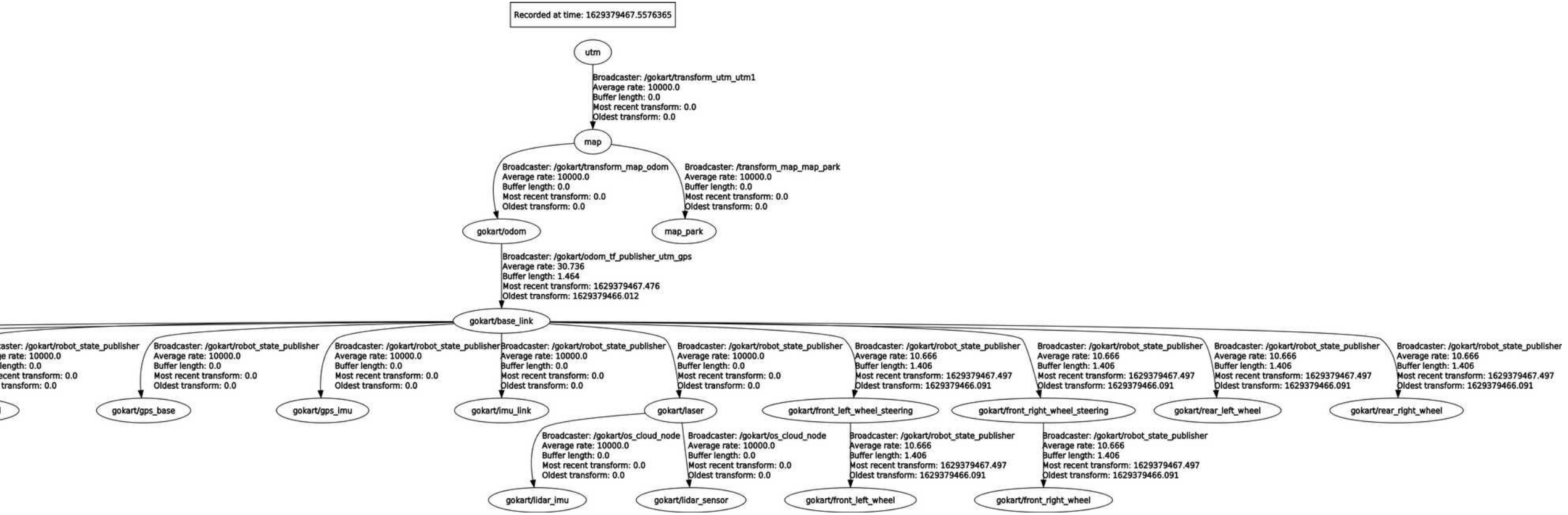
Forrás: https://www.ros.org/reps/rep-0105.html

ROS – Tf2 II.: koordináta-rendszerek II.





ROS – Tf2 II.: koordináta-rendszerek III.



ROS – tf2 terminál parancs

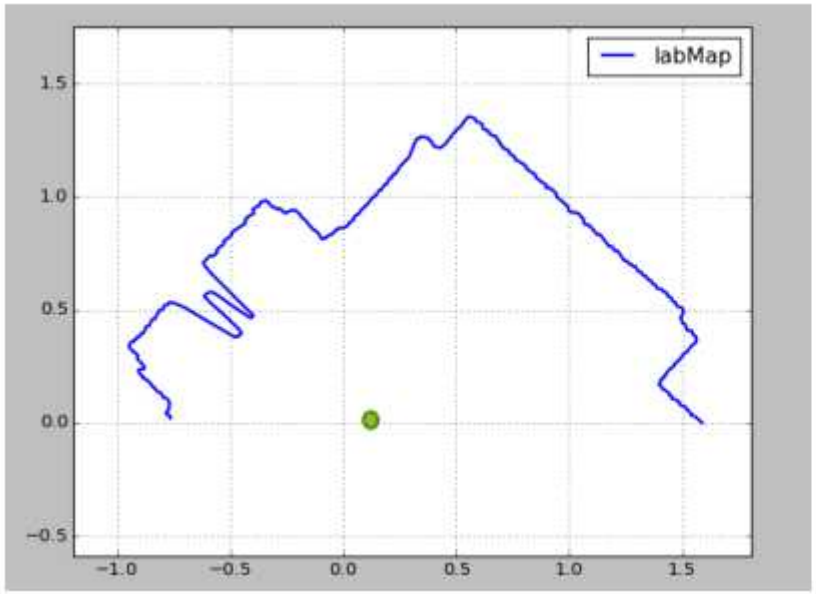
- A tf2 önálló package, tf2_ros néven, és beépülő modulként C++ és Python implementációja van.
- Terminálból:
  “ros2 run tf2_ros static_transform_publisher --x 0.0 --y 0.0 --z 0.0 --roll 0.0 --pitch 0.0 --yaw 0.0 --frame-id map --child-frame-id odom”
  - *6 számos esetben: x y z yaw pitch roll*
  - *7 számos esetben: x y z qx qy qz qw*

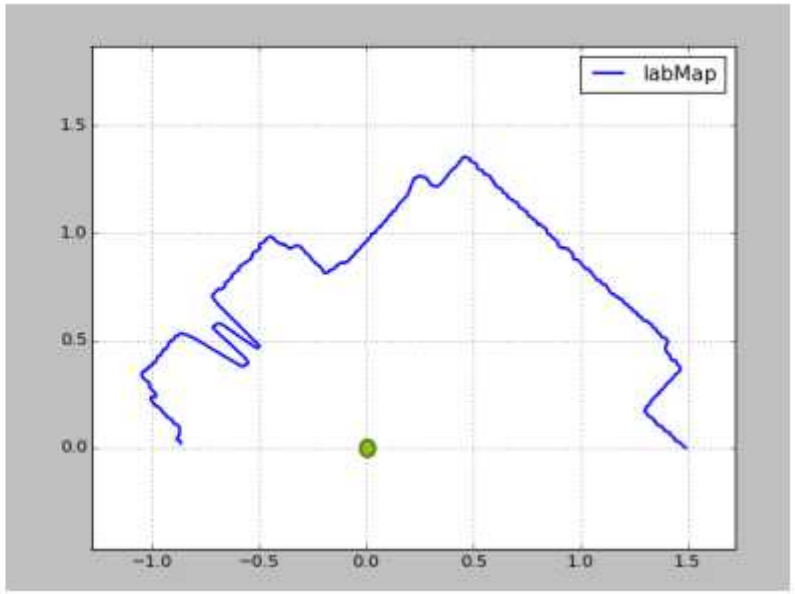
- Launch fájl:

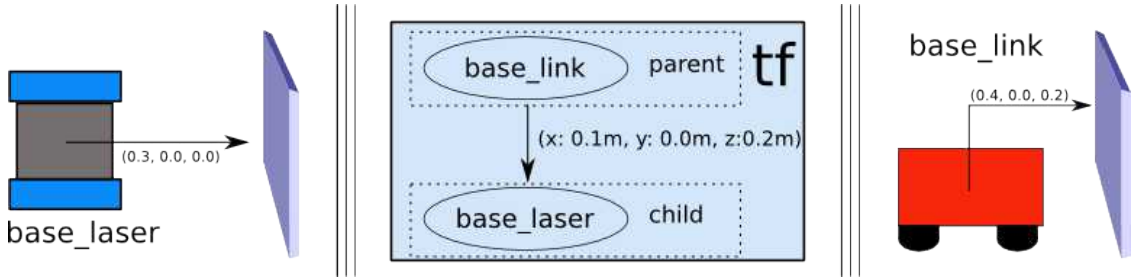
```
<node pkg="tf2_ros" exec="static_transform_publisher" name="test_broadcaster"
   args="--x 0.0 --y 0.0 --z 0.0 --roll 0.0 --pitch 0.0 --yaw 0.0 --frame-id map --child-frame-id
odom" />
```

- Vizualizáció:
  - rviz2
  - `ros2 run rqt_tf_tree rqt_tf_tree`

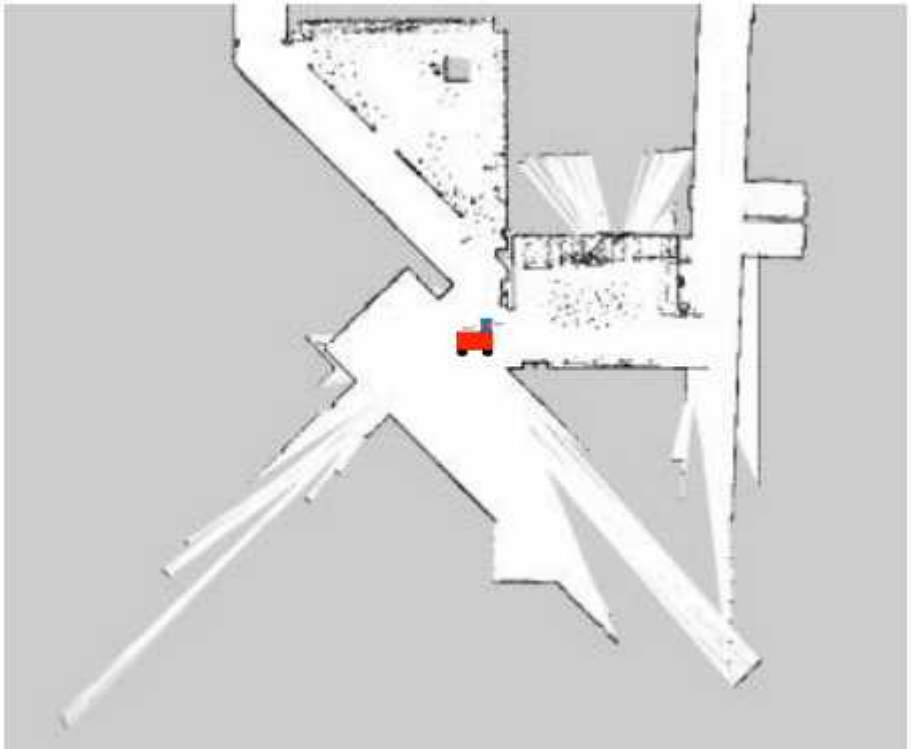
TF2 lidar pontok a gyakorlatban I.

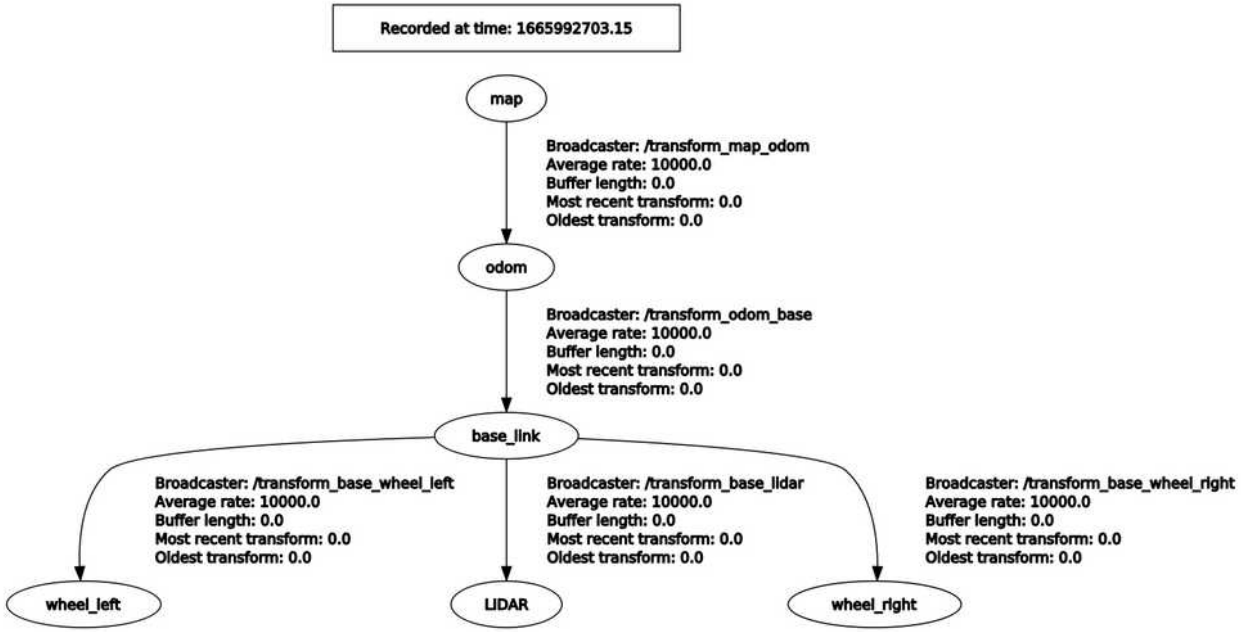




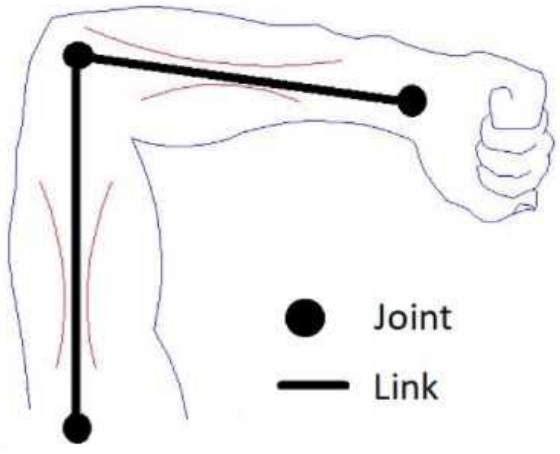


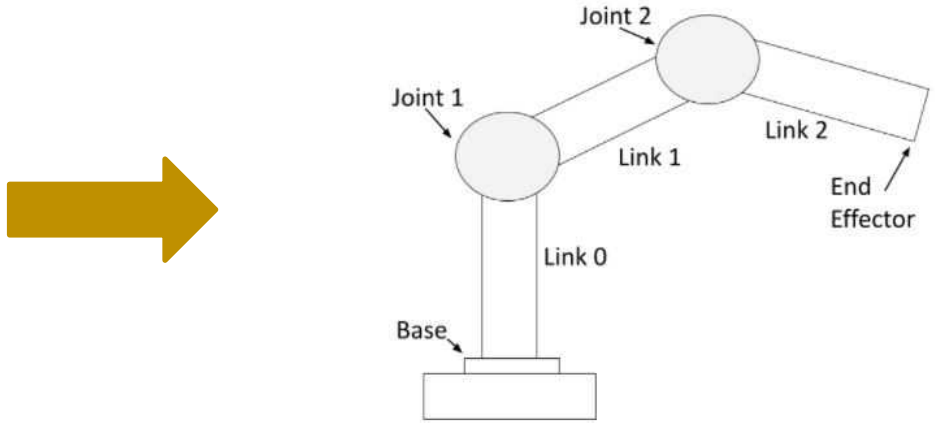
TF2 lidar pontok a gyakorlatban II.

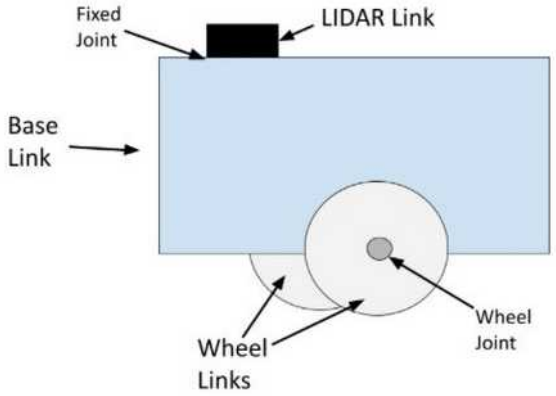




- Robot leíró - Unified Robotic Description Format (URDF)
  - XML formátumú leíró fájlok, melyek nyelvezete ki lett egészítve ros specifikus makró nyelvvel, xacro néven.







ROS – Robotok II.


```
<?xml version="1.0"?>

<robot name="gokart" xmlns:xacro="http://www.ros.org/wiki/xacro">

    <link name="$(arg agent)/base_link">
        <visual>
            <origin xyz="1.035 0 -0.14" rpy="1.57 0 1.57" />
            <geometry>
                <mesh scale="0.001 0.001 0.001" filename="package://gokart_system/meshes/electric_go_kart_simplified.stl" />
            </geometry>
            <material name="grey"/>
        </visual>
        <collision>
            <origin xyz="0.55 0 0.3" rpy="0 0 0" />
            <geometry>
                <box size="1.7 1.2 0.02" />
            </geometry>
        </collision>
    </link>

    <link name="$(arg agent)/dummy">
        <inertial>
            <mass value="50.0" />
            <origin xyz="0.5 0 -0.1" rpy="0 0 0" />
            <xacro:solid_cuboid_inertia w="1.0" h="0.6" d="0.03" m="50.0"/>
        </inertial>
    </link>

    <joint name="dummy_link_joint" type="fixed">
        <origin xyz="0.0 0 0" rpy="0 0 0" />
        <parent link="$(arg agent)/base_link" />
        <child  link="$(arg agent)/dummy" />
    </joint>
</robot>
```

ROS – Robotok III.


```
<?xml version="1.0"?>

<robot name="gokart" xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:include filename="$(find gokart_system)/urdf/rear_wheel.urdf.xacro" />

    <xacro:macro name="solid_cuboid_inertia" params="w h d m">
    <inertia  ixx="${m*(h*h+d*d)/12}" ixy = "0" ixz = "0"
                      iyy="${m*(w*w+d*d)/12}" iyz = "0"
                      izz="${m*(w*w+h*h)/12}" />
    </xacro:macro>

    <xacro:arg name="agent" default="gokart" />
    <xacro:arg name="lidar_type" default="sick"/>

    <xacro:property name="lidar" value="$(arg lidar_type)"/>

    <material name="red">
        <color rgba="${255/255} ${0/255} ${0/255} 1.0"/>
    </material>

    <link name="$(arg agent)/dummy">
        <inertial>
            <mass value="50.0" />
            <origin xyz="0.5 0 -0.1" rpy="0 0 0" />
            <xacro:solid_cuboid_inertia w="1.0" h="0.6" d="0.03" m="50.0"/>
        </inertial>
    </link>

 </robot>
```

ROS – Robotok IV.


```
<?xml version="1.0"?>

<robot name="gokart" xmlns:xacro="http://www.ros.org/wiki/xacro">
    <gazebo>
        <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
            <robotNamespace>$(arg agent)</robotNamespace>
            <robotSimType>gazebo_ros_control/DefaultRobotHWSim</robotSimType>
            <legacyModeNS>true</legacyModeNS>
        </plugin>
    </gazebo>

    <gazebo reference="$(arg agent)/imu_link">
        <gravity>true</gravity>
        <sensor name="imu_sensor" type="imu">
            <always_on>true</always_on>
            <update_rate>100</update_rate>
            <visualize>true</visualize>s
            <topic>__default_topic__</topic>
            <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
                <topicName>imu/raw</topicName>
                <bodyName>$(arg agent)/imu_link</bodyName>
                <updateRateHZ>10.0</updateRateHZ>
                <gaussianNoise>0.01</gaussianNoise>
                <xyzOffset>0 0 0</xyzOffset>
                <rpyOffset>0 0 0</rpyOffset>
                <frameName>$(arg agent)/imu_link</frameName>
            </plugin>
        </sensor>
    </gazebo>

</robot>
```