Robot operációs rendszerek és fejlesztői ökoszisztémák

Autonóm robotikai funkciók

Gincsainé Szádeczky-Kardoss Emese

2025. november 17.

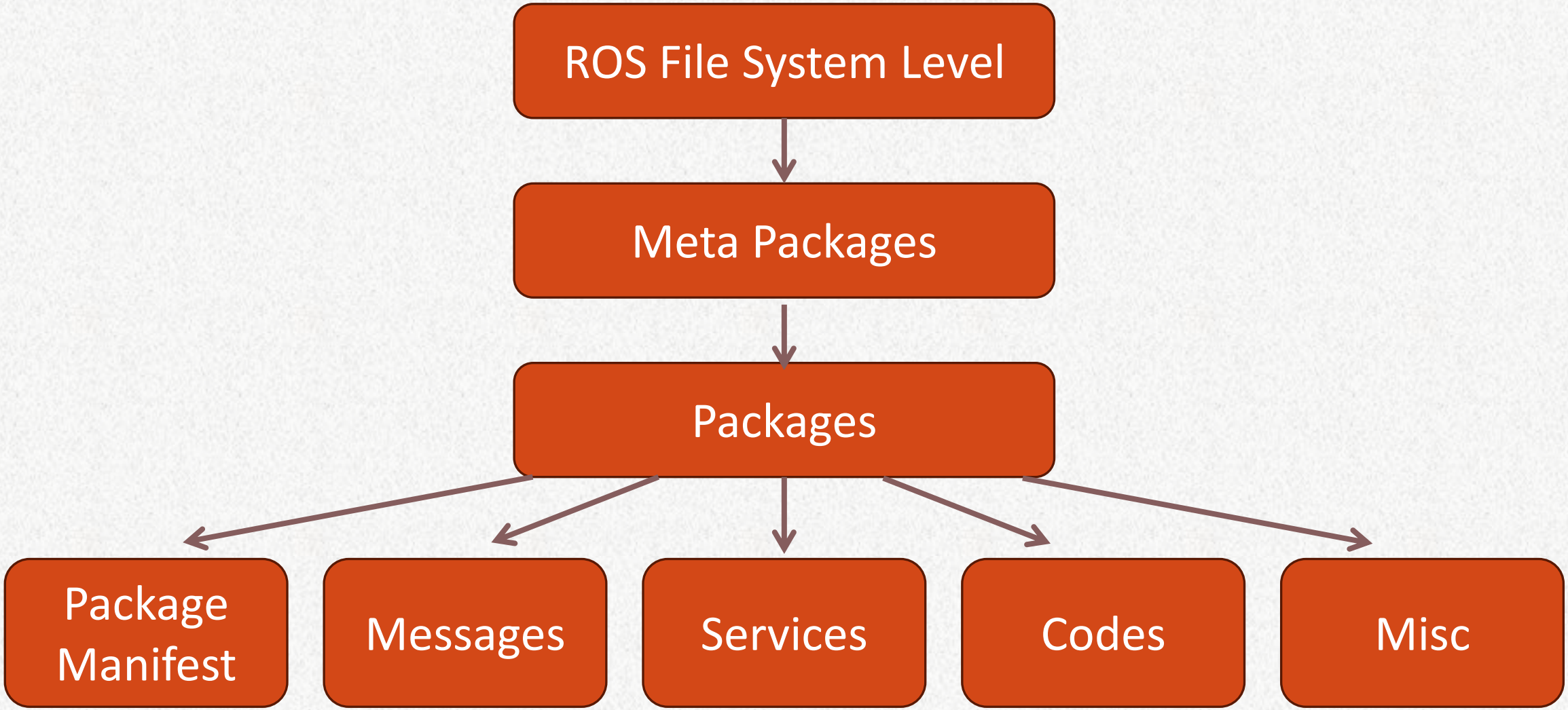
KUKA

MŰEGYETEM 1782

iit

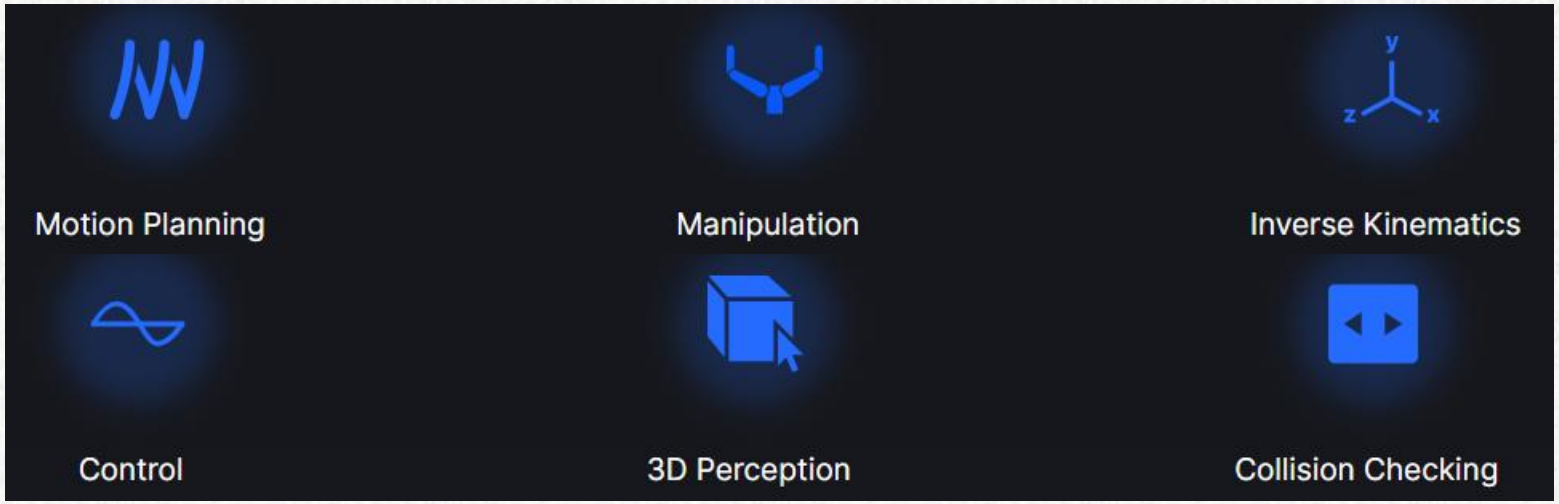
Tartalom

Visszatekintés


ROS File System Level
Meta Packages
Packages
Package Manifest
Messages
Services
Codes
Misc

MoveIt

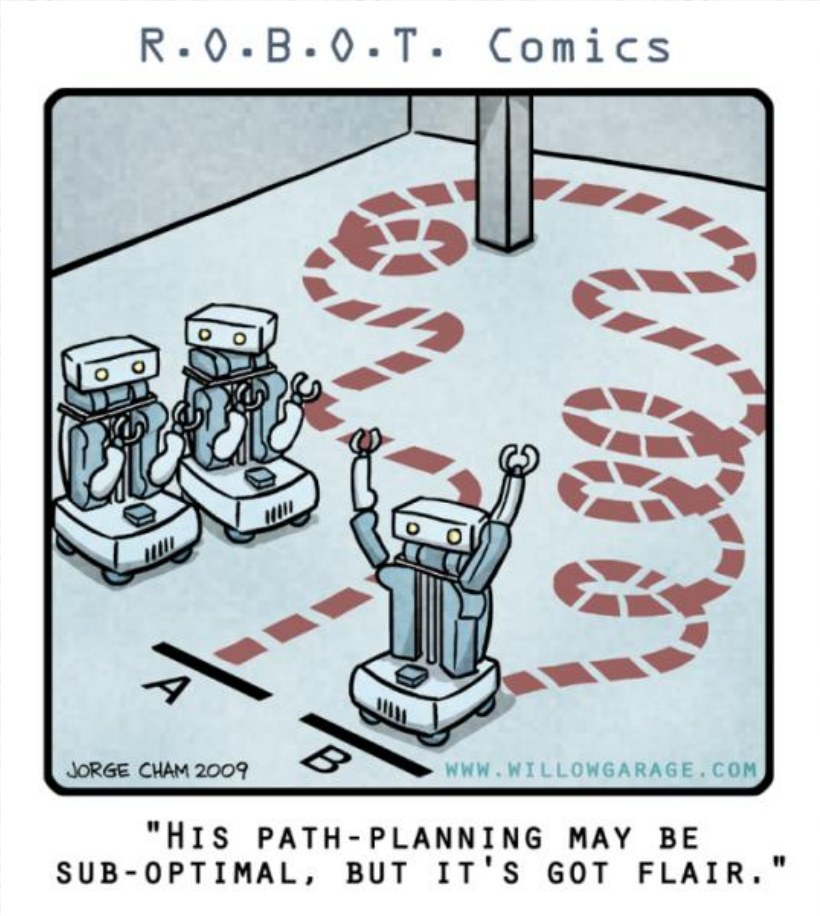
- https://moveit.ai/
- MoveIt Motion Planning Framework
  - Robotok mozgástervezése
  - Megfogás tervezés
  - Inverz kinematika: végberendezés kívánt pozíciójából (sebességéből) csuklóváltozók meghatározása
  - Robotok irányítás
  - Környezet érzékelés (perception)
  - Ütközés detektálás
- Free, open source



Navigation stack

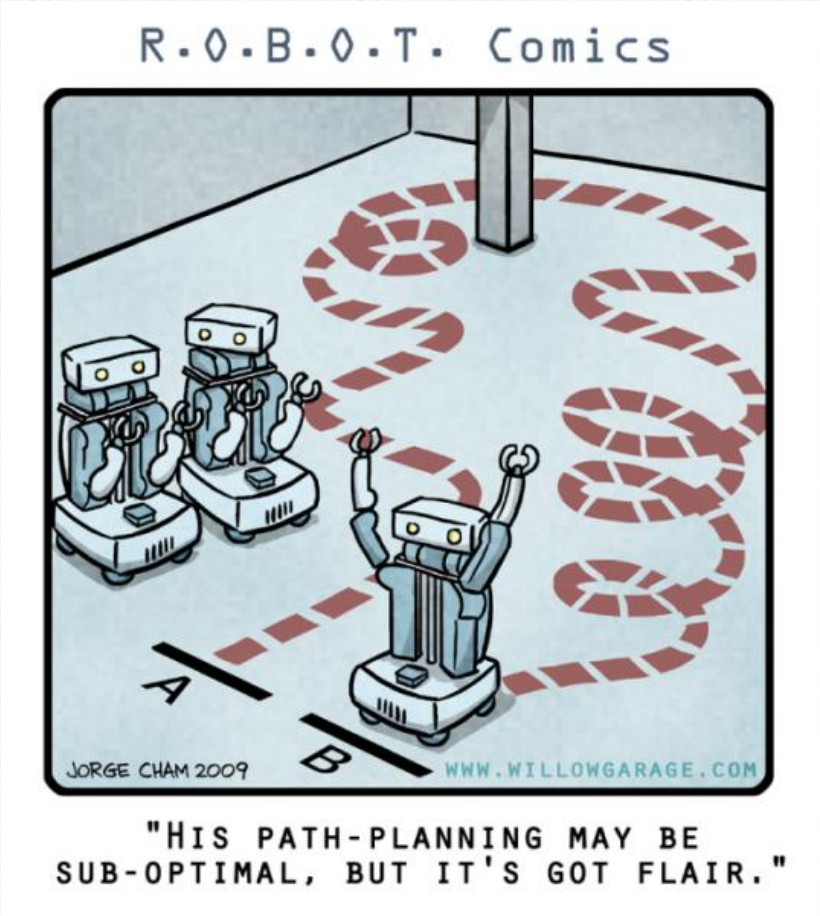
Navigation stack

- ROS1 stack 2D navigáció mobilis robotok számára
- Open source
- Cross platform
- Különböző robot modellek
- Különböző szenzorok
- 2D, 3D környezet reprezentáció
- Autonóm navigációs és térképezési feladatok megoldása



Navigation stack

- Bemeneti információk:
  - odometria
  - szenzor adatok
  - célpozíció
- Kimenet:
  - sebesség parancsok a mobilis robot számára
  - amik ütközésmentes mozgást biztosítanak
- Követelmények:
  - robot ROS-t futtasson
  - tf transzformációs fa
  - megfelelő típusú szenzor adatok a megfelelő topic-okon



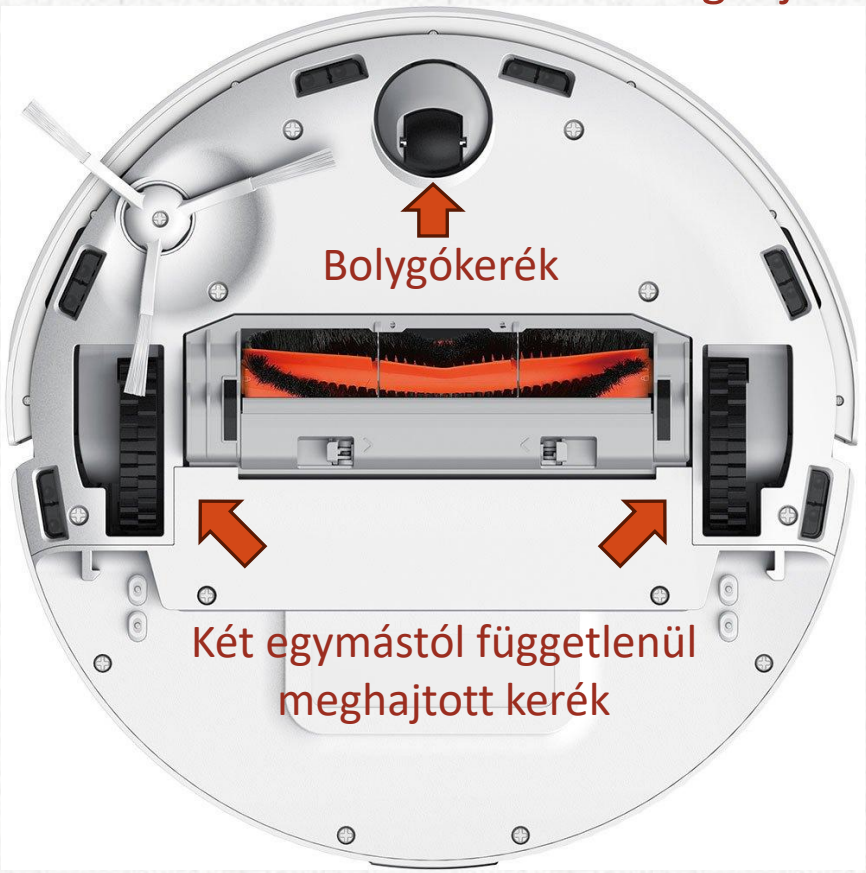
Navigation stack - robot

Hardver követelmények:

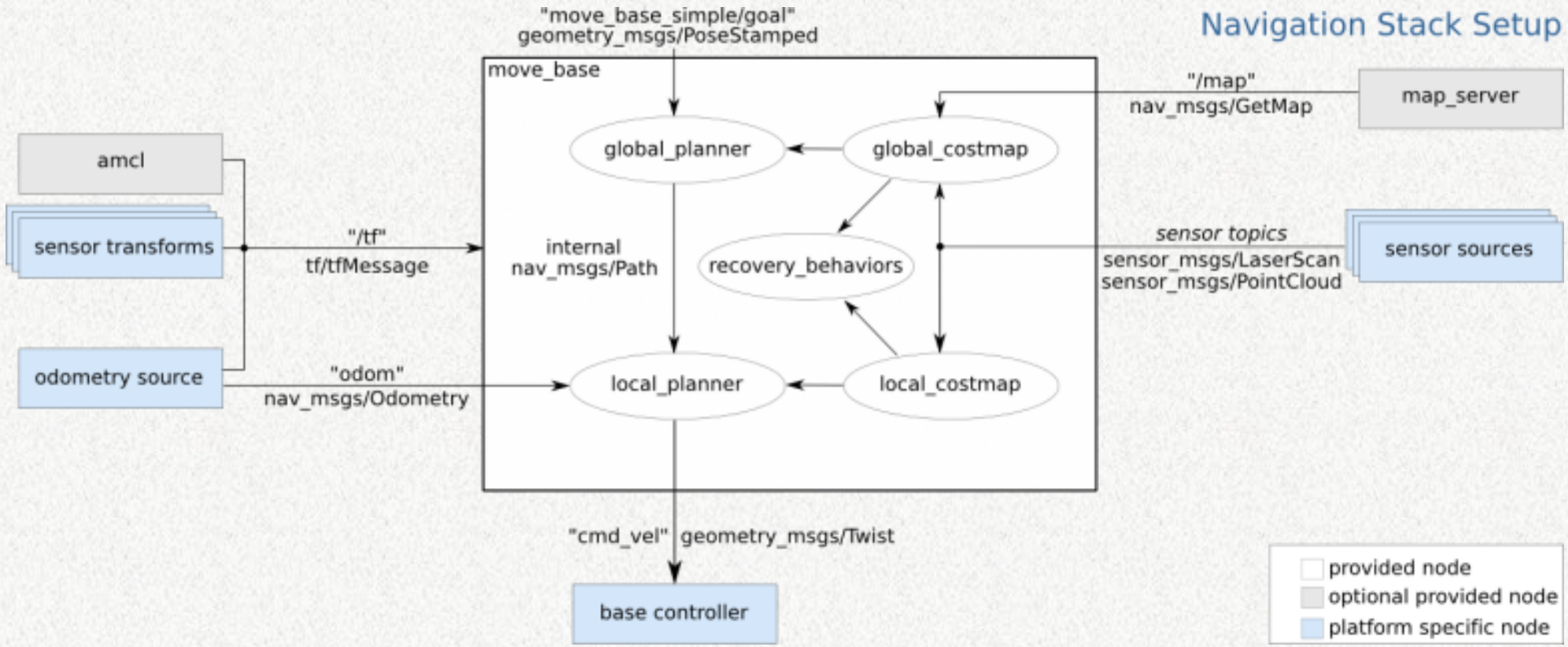
- Differenciális meghajtású robot vagy omindirekcionális robot
- Robot irányítható sebesség parancsokkal (x irányú sebesség, y irányú sebesség, szögsebesség)
- Sík lézer szkenner használata térképezéshez és lokalizációhoz
- Nagyjából kör vagy négyzet alapterületű robotokra hatékony.

Omnidirekcionális meghajtás

Differenciális meghajtás

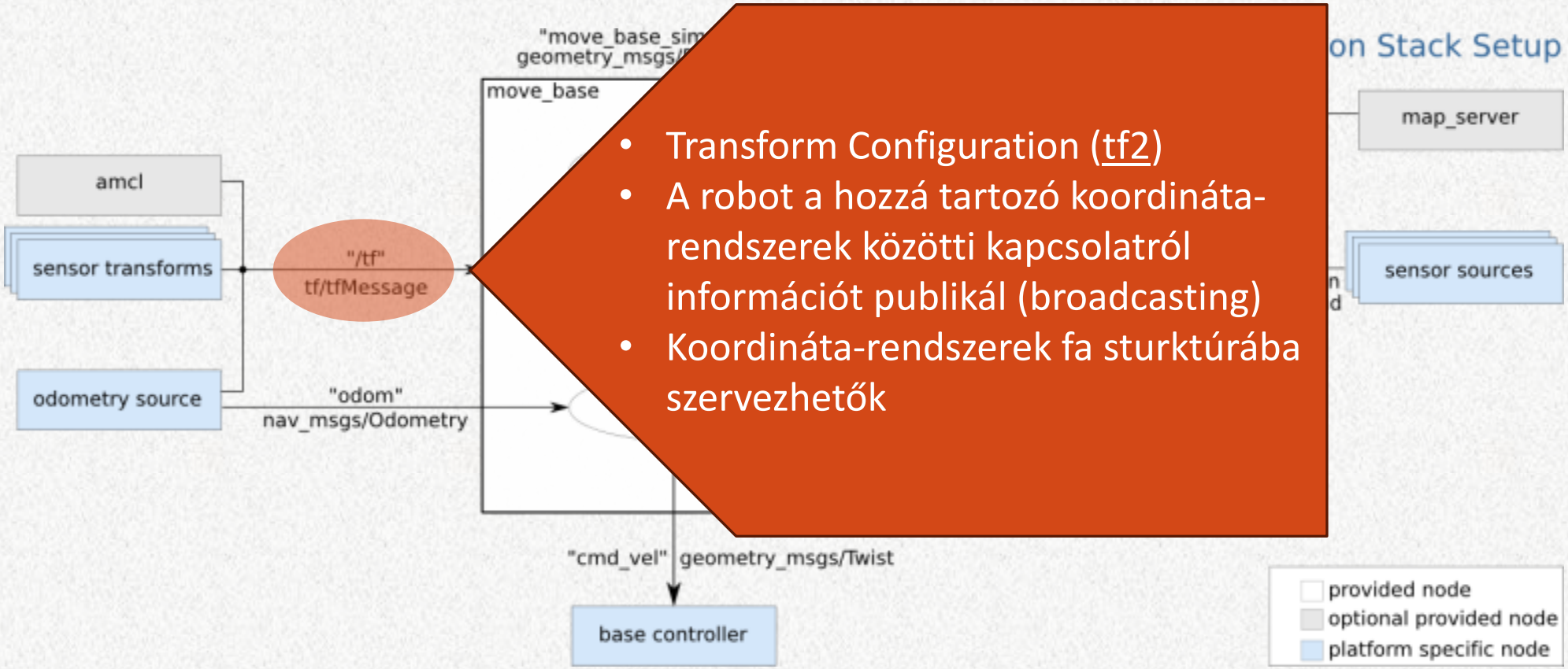


Robot setup



http://wiki.ros.org/navigation/Tutorials/RobotSetup

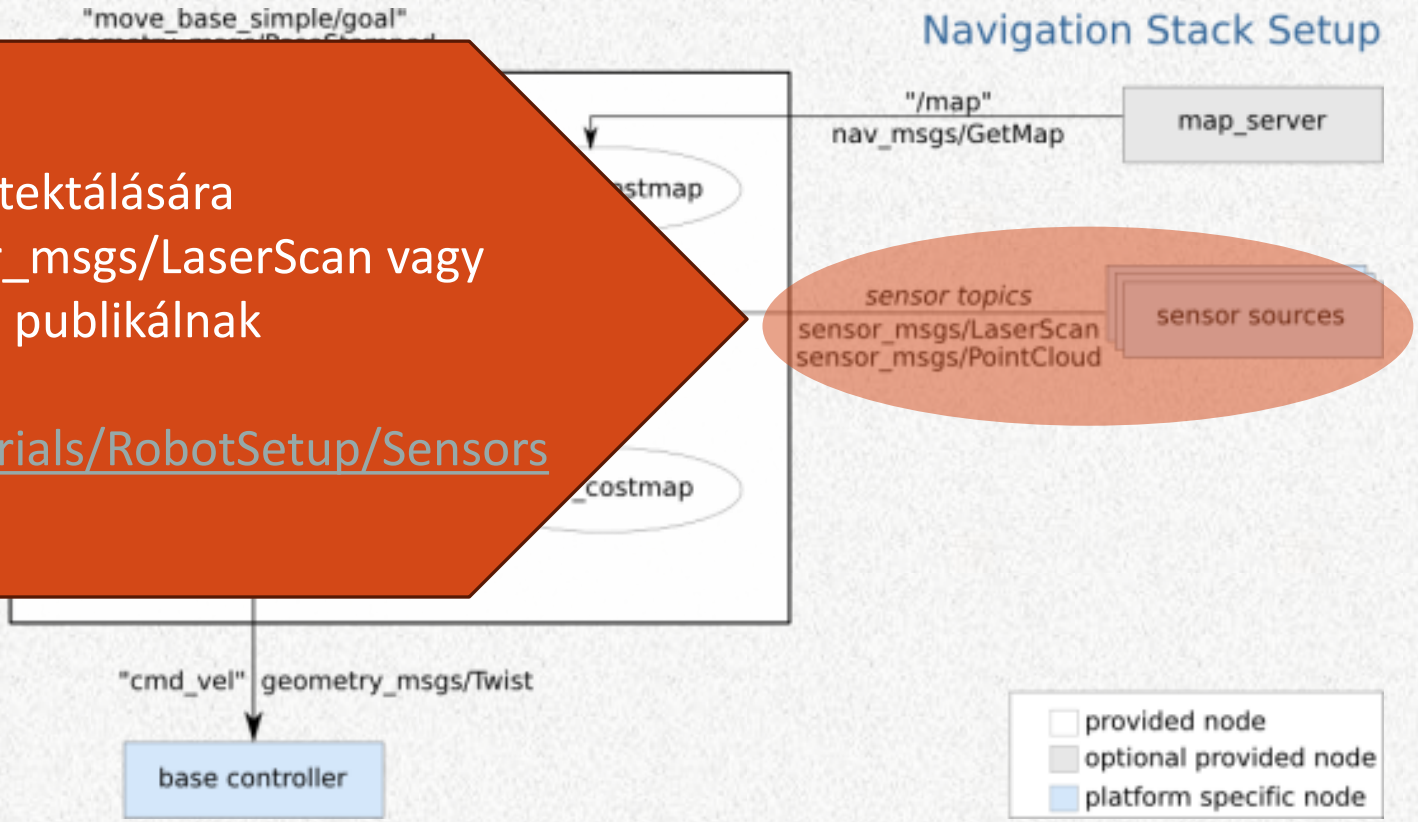
Robot setup - /tf



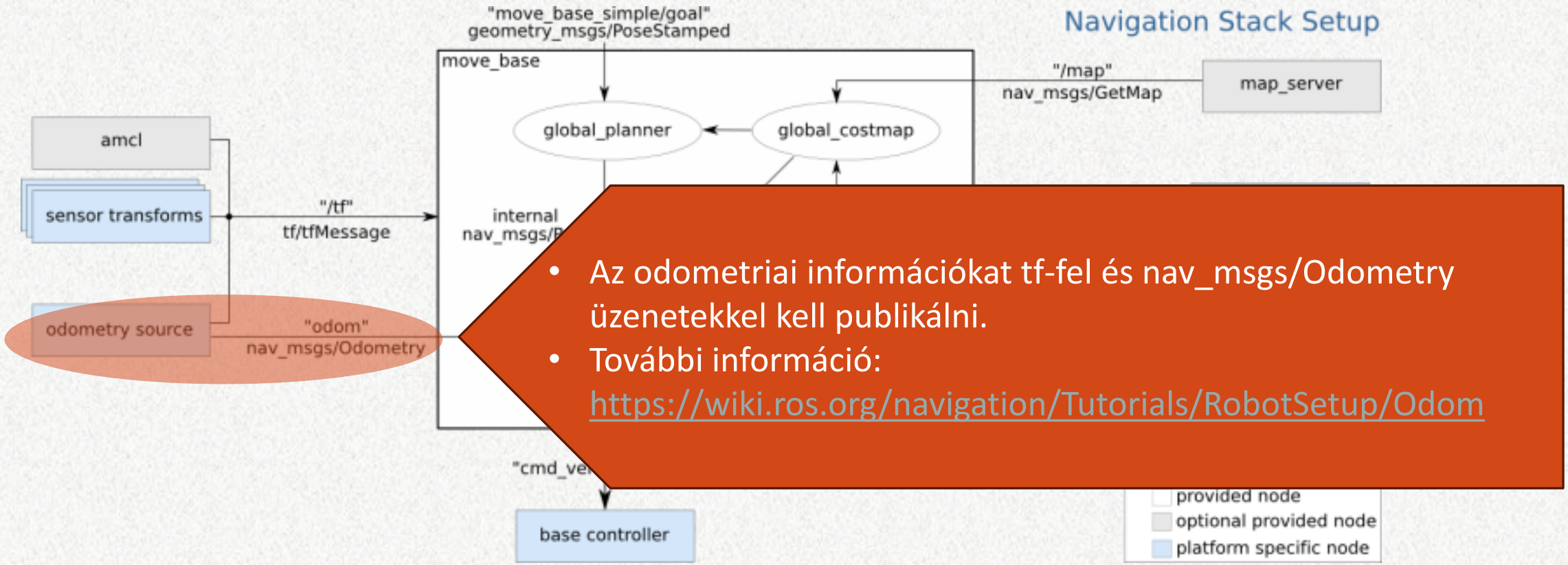
- Transform Configuration (tf2)
- A robot a hozzá tartozó koordináta-rendszerek közötti kapcsolatról információt publikál (broadcasting)
- Koordináta-rendszerek fa sturktúrába szervezhetők

Robot setup – Szenzor információ

- Szenzorok használata a környezet detektálására
- A szenzorok ROS-on keresztül sensor_msgs/LaserScan vagy sensor_msgs/PointCloud üzeneteket publikálnak
- További információ: https://wiki.ros.org/navigation/Tutorials/RobotSetup/Sensors

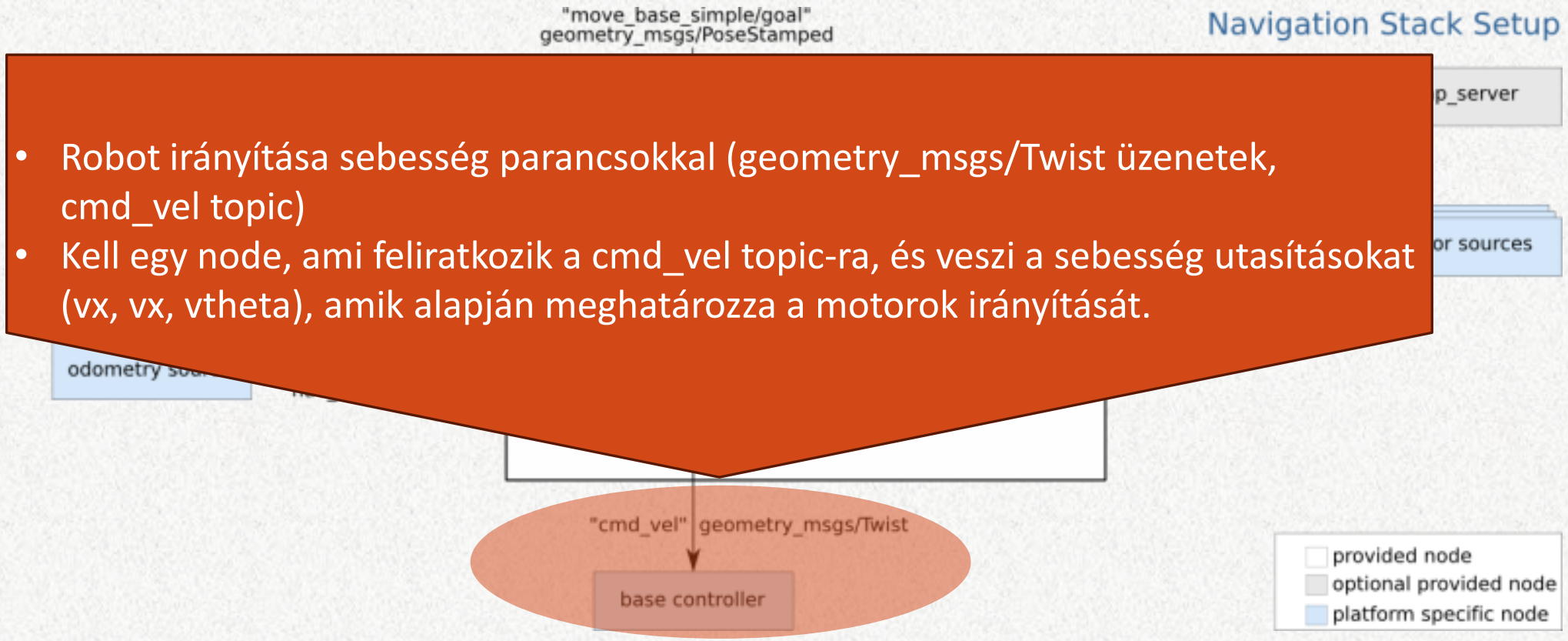


Robot setup - odometria



- Az odometriai információkat tf-fel és nav_msgs/Odometry üzenetekkel kell publikálni.
- További információ: https://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom

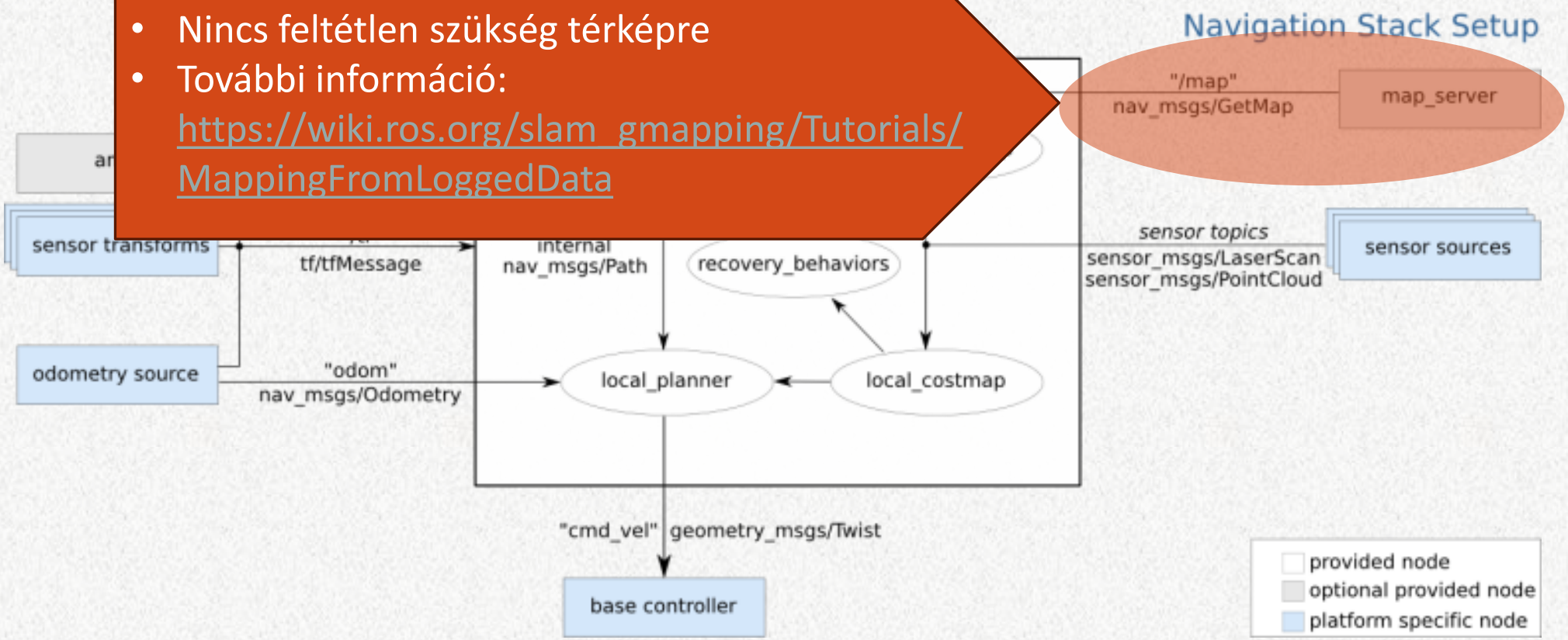
Robot setup – bázis irányítása



- Robot irányítása sebesség parancsokkal (geometry_msgs/Twist üzenetek, cmd_vel topic)
- Kell egy node, ami feliratkozik a cmd_vel topic-ra, és veszi a sebesség utasításokat (vx, vx, vtheta), amik alapján meghatározza a motorok irányítását.

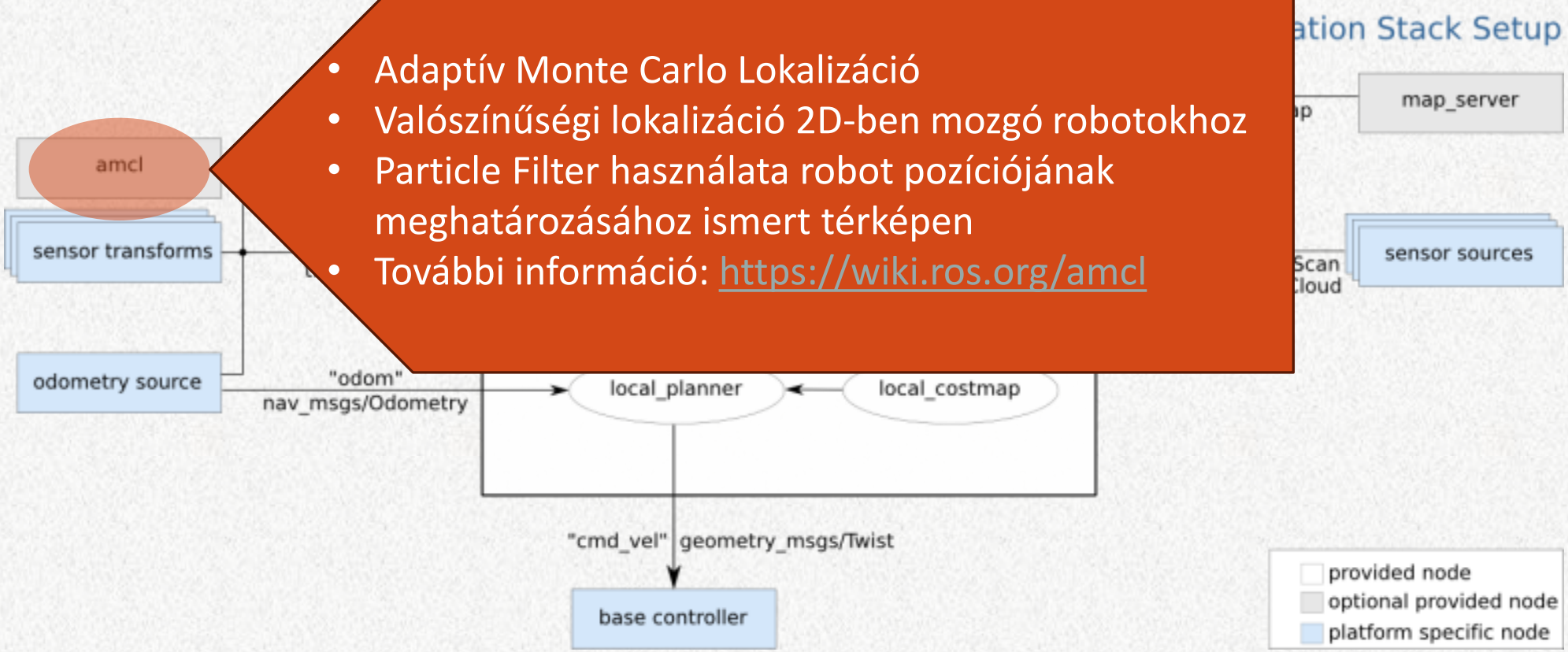
Robot setup – Térkép

- Nincs feltétlen szükség térképre
- További információ: https://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData



Robot setup - amcl

- Adaptív Monte Carlo Lokalizáció
- Valószínűségi lokalizáció 2D-ben mozgó robotokhoz
- Particle Filter használata robot pozíciójának meghatározásához ismert térképen
- További információ: https://wiki.ros.org/amcl



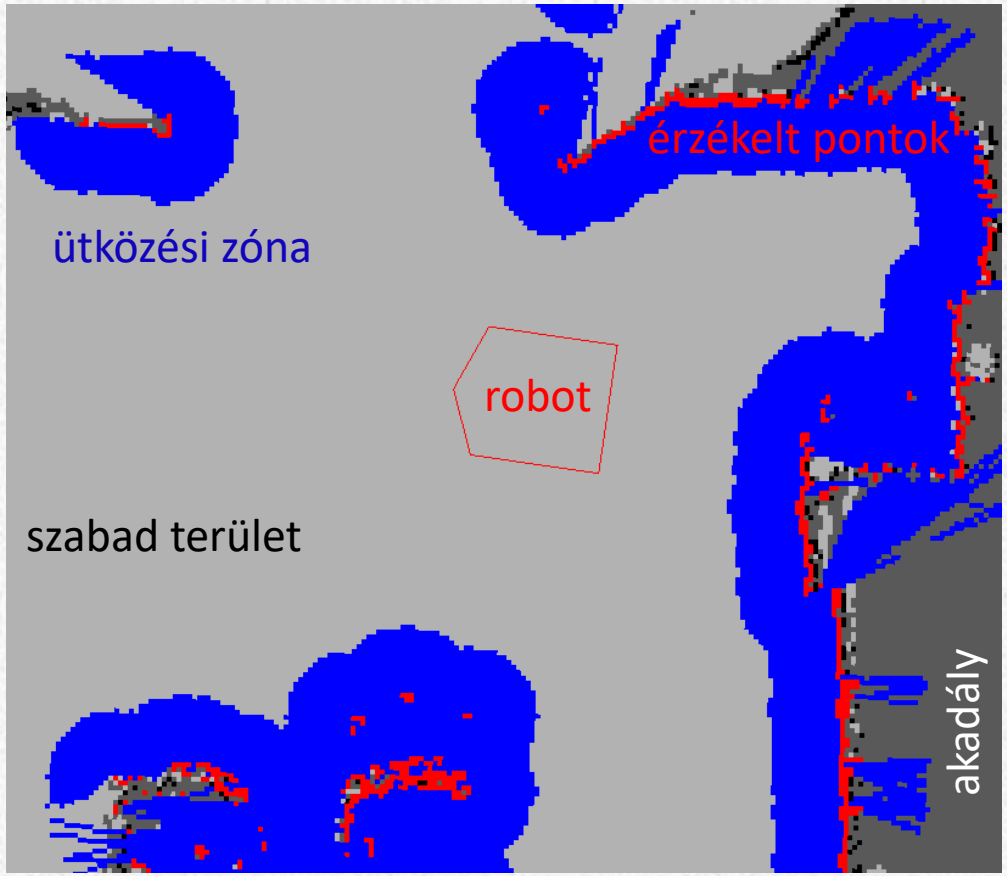
Navigation Stack Setup

- Package létrehozása a konfigurációs és launch állományok számára a megfelelő függőségekkel
- Launch fájl létrehozása (my_robot_configuration.launch)
  - Szenzor(ok) ROS driver-ének megfelelő node(ok) indítása
  - Odometria számítását végző node indítása
  - Traszformációk kezeléséhez szükséges node indítása
  - A node-okhoz tartozó paraméterek beállítása

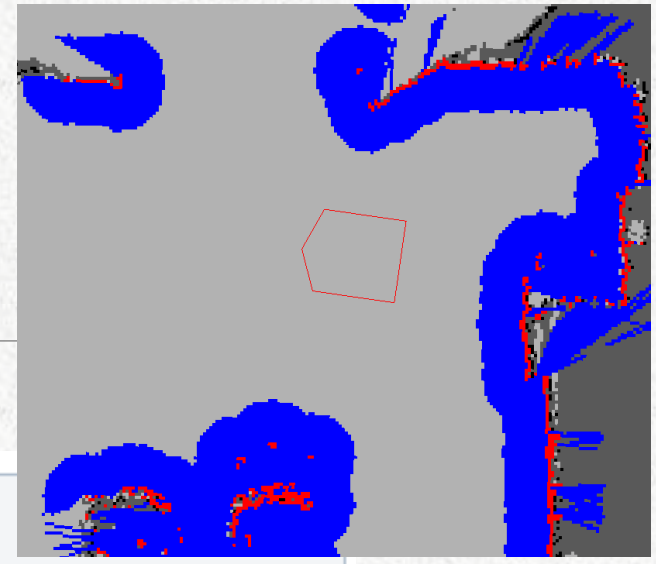
```
<launch>

   <node pkg="sensor_node_pkg" type="sensor_node_type" name="sensor_node_name" output="screen">
    <param name="sensor_param" value="param_value" />
 </node>
 <node pkg="odom_node_pkg" type="odom_node_type" name="odom_node" output="screen">
    <param name="odom_param" value="param_value" />
 </node>
 <node pkg="transform_configuration_pkg" type="transform_configuration_type" name="transform_con
figuration_name" output="screen">
    <param name="transform_configuration_param" value="param_value" />
 </node>

</launch>
```

Navigation Stack Setup (folyt.)

- Package létrehozása a konfigurációs és launch állományok számára a megfelelő függőségekkel
- Launch fájl létrehozása (my_robot_configuration.launch)
- Költségtérképek (costmap) beállítása – információ az akadályokról
  - Közös beállítások (lokális és globális) – costmap_common_params.yaml
  - Globális beállítások – global_costmap_params.yaml
  - Lokális beállítások – local_costmap_params.yaml



Costmap – szenzor hatótáv

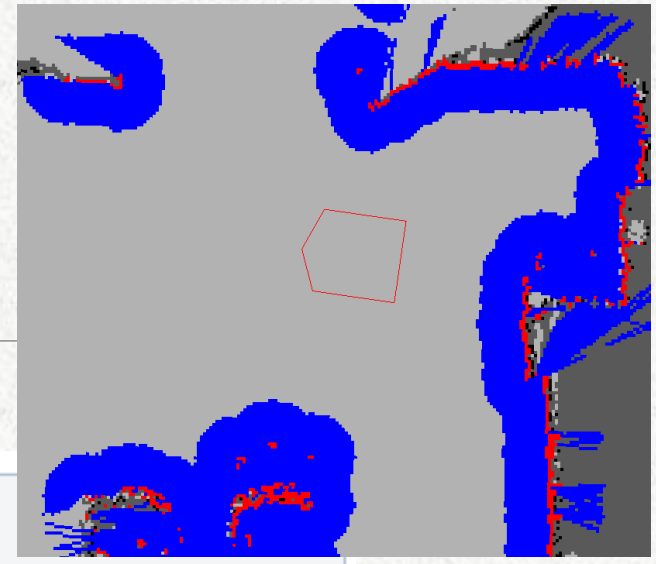


costmap_common_params.yaml

Akadályt akkor vesz fel a térképre, ha ezen távolságon belül van.

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking:
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Costmap – szenzor hatótáv



costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking:
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Szabad területet ezen távolságon belül jelöl be a térképen.

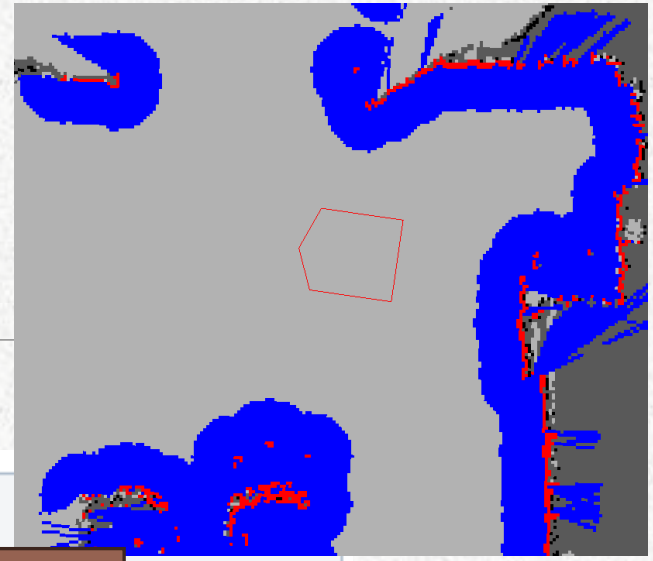
Costmap – robot alakja

costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking:
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Ha a robot alapterülete egy sokszög, akkor itt helyezkednek el a csúcsok az origóba helyezett robotnak.

Costmap – robot alakja

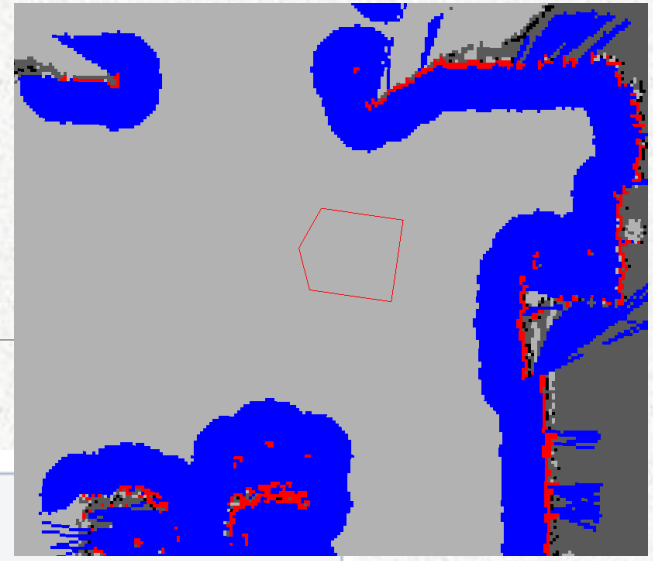


costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking:
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Ha a kör alakú lenne a robot, ez lenne a sugara.

Costmap – robot alakja



costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], .
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_s

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking:
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Az akadály max ilyen távolságra generál költséget.

Costmap – szenzorok

costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan,
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Costmap megalkotásához használt szenzorok

Costmap – szenzor paraméterek

costmap_common_params.yaml

```
obstacle_range: 2.5
raytrace_rang
footprint: [
#robot_radius
inflation_rad

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking:
true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, markin
g: true, clearing: true}
```

Szenzorok paramétereinek megadása (szenzor koordináta-rendszere; az adat típusa; a topic neve, ahova ad; a térképhez hozzáad/elvesz akadályokat)

Costmap – globális konfigurációk

global_costmap_params.yaml

```
global_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 5.0
  static_map: true
```

- a costmap ebben a koordináta-rendszerben értelmezendő
- a robothoz tartozó koordináta-rendszer
- ilyen frekvenciával frissül a costmap [Hz]
- A map_server által biztosított térképpel inicializáljunk?

Costmap – lokális konfigurációk

local_costmap_params.yaml

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

- global_frame, robot_base_frame, update_frequency, static_map jelentése ugyanaz, mint a globális costmapnél

Costmap – lokális konfigurációk

local_costmap_params.yaml

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

- Ilyen gyakran publikál vizuális információt [Hz]
- Mozgó robot mindig a costmap közepén helyezkedik el?
- Costmap szélessége [m], magassága [m] és felbontása [m/cella]

Navigation Stack Setup (folyt.)

- Package létrehozása a konfigurációs és launch állományok számára a megfelelő függőségekkel
- Launch fájl létrehozása
- Költségtérképek (costmap) beállítása – információ az akadályokról
- Base Local Planner beállításai
  - magas szintű irányítás (sebesség utasítások) meghatározása a robot számára
  - base_local_planner_params.yaml
    - sebesség, gyorsulás határértékek
    - holonom mozgás?

```
TrajectoryPlannerROS:
  max_vel_x: 0.45
  min_vel_x: 0.1
  max_vel_theta: 1.0
  min_in_place_vel_theta: 0.4

  acc_lim_theta: 3.2
  acc_lim_x: 2.5
  acc_lim_y: 2.5

  holonomic_robot: true
```

Navigation Stack Setup (folyt.)

- Package létrehozása a konfigurációs és launch állományok számára a megfelelő függőségekkel
- Launch fájl létrehozása
- Költségtérképek (costmap) beállítása – információ az akadályokról
- Base Local Planner beállításai
- Launch fájl létrehozása a Navigation Stack számára (move_base.launch)
  - master indítás
  - map_server node indítása
  - AMCL indítása
  - move_base node indítása
  - paraméterek beállítása (yaml fájlok alapján)

```
<launch>

   <master auto="start"/>
 <!-- Run the map server -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find my_map_package)/my_m
ap.pgm my_map_resolution"/>

 <!--- Run AMCL -->
    <include file="$(find amcl)/examples/amcl_omni.launch" />

   <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find my_robot_name_2dnav)/costmap_common_params.yaml" command="load" ns="g
lobal_costmap" />
    <rosparam file="$(find my_robot_name_2dnav)/costmap_common_params.yaml" command="load" ns="l
ocal_costmap" />
    <rosparam file="$(find my_robot_name_2dnav)/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find my_robot_name_2dnav)/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find my_robot_name_2dnav)/base_local_planner_params.yaml" command="load" /
>
 </node>

</launch>
```

Navigation Stack - futtatás

- 1. terminálban:

```
roslaunch my_robot_configuration.launch
```

- 2. terminálban:

```
roslaunch move_base.launch
```

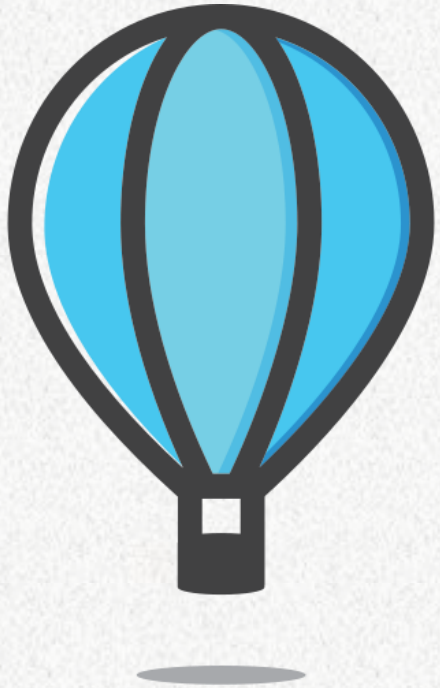
Cél beállítása:

- Grafikus felületen keresztül: rviz-zel
- Kódból: C++ megoldás

Van navigáció ROS2-ben?

Igen!

https://nav2.org/



N A V 2

NAV 2

NAV 2 szolgáltatásai

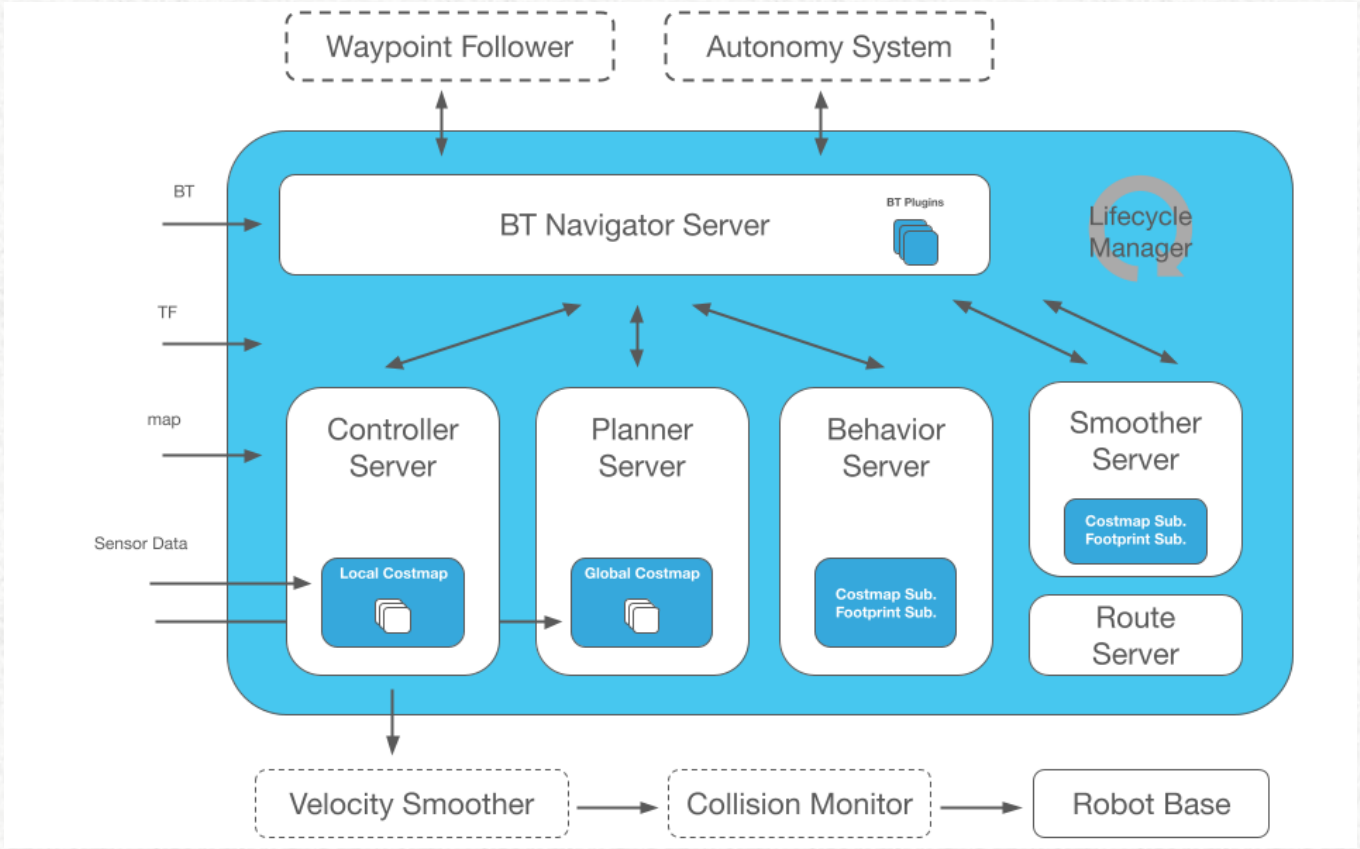
Navigációs keretrendszer főbb feladatai:

- Érzékelés
- Tervezés
- Irányítás
- Lokalizáció
- Vizualizáció
- Viselkedés

Különböző célú mozgások:

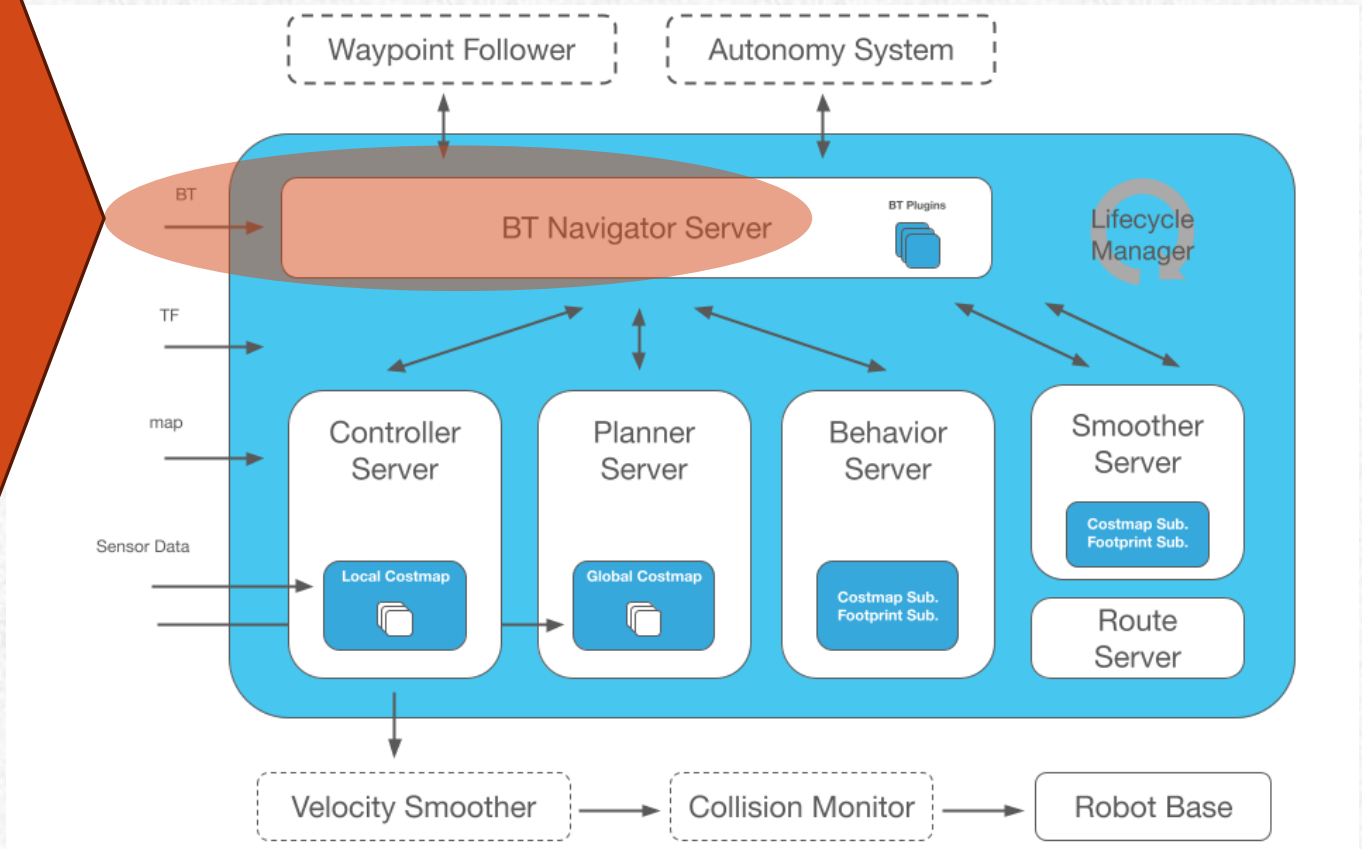
- Mozgás A pontból B pontba
- Objektum követés
- Területfedés (pl. felderítő robot vagy robotporszívó)

NAV 2 – Tipikus blokkdiagramm



– Viselkedési fák

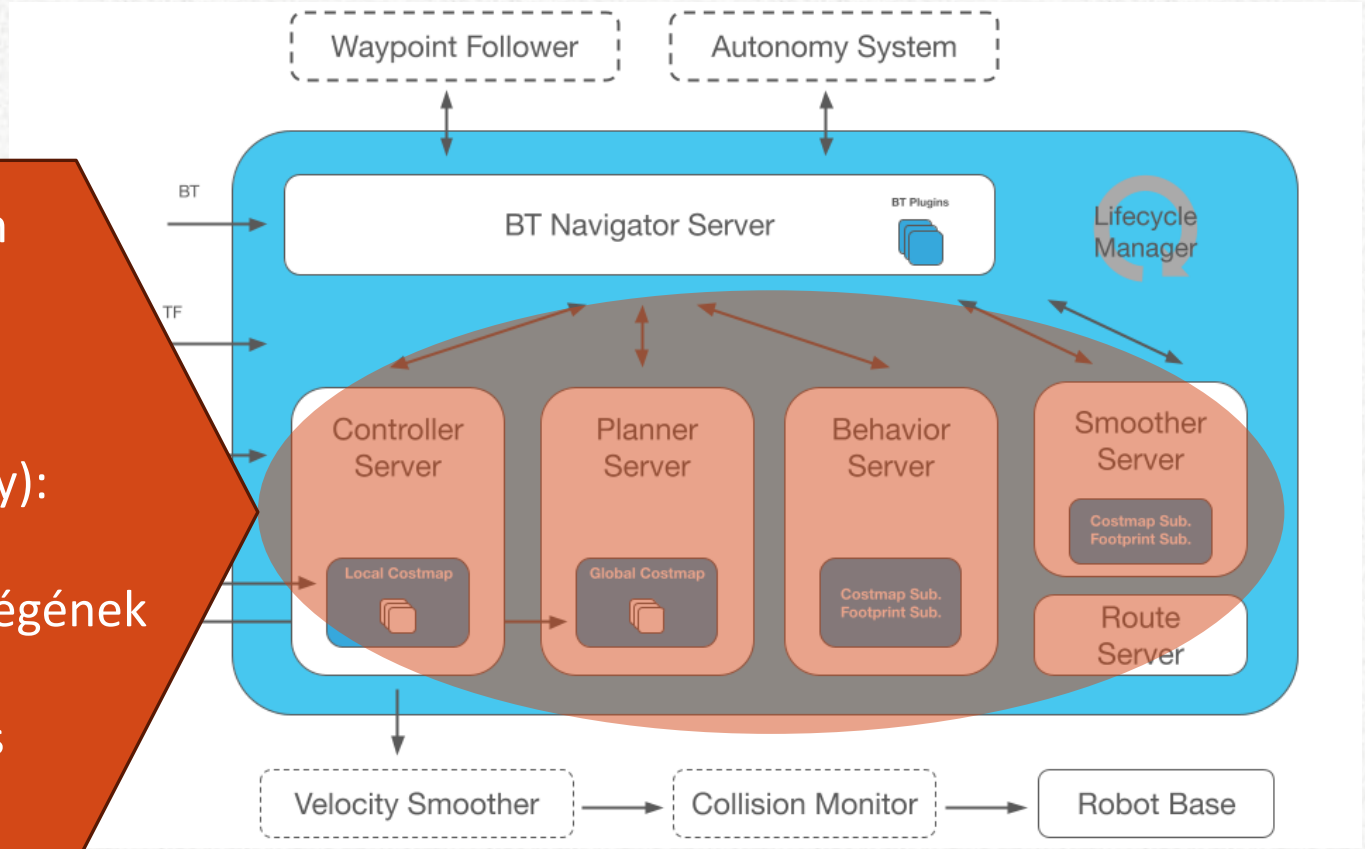
BT: Behavior tree
- xml fájl
- független, moduláris szerverek összehangolása
- BT-szerver kommunikáció ROS interface-eken keresztül (service, action)
- különböző BT-k különböző feladatok-hoz



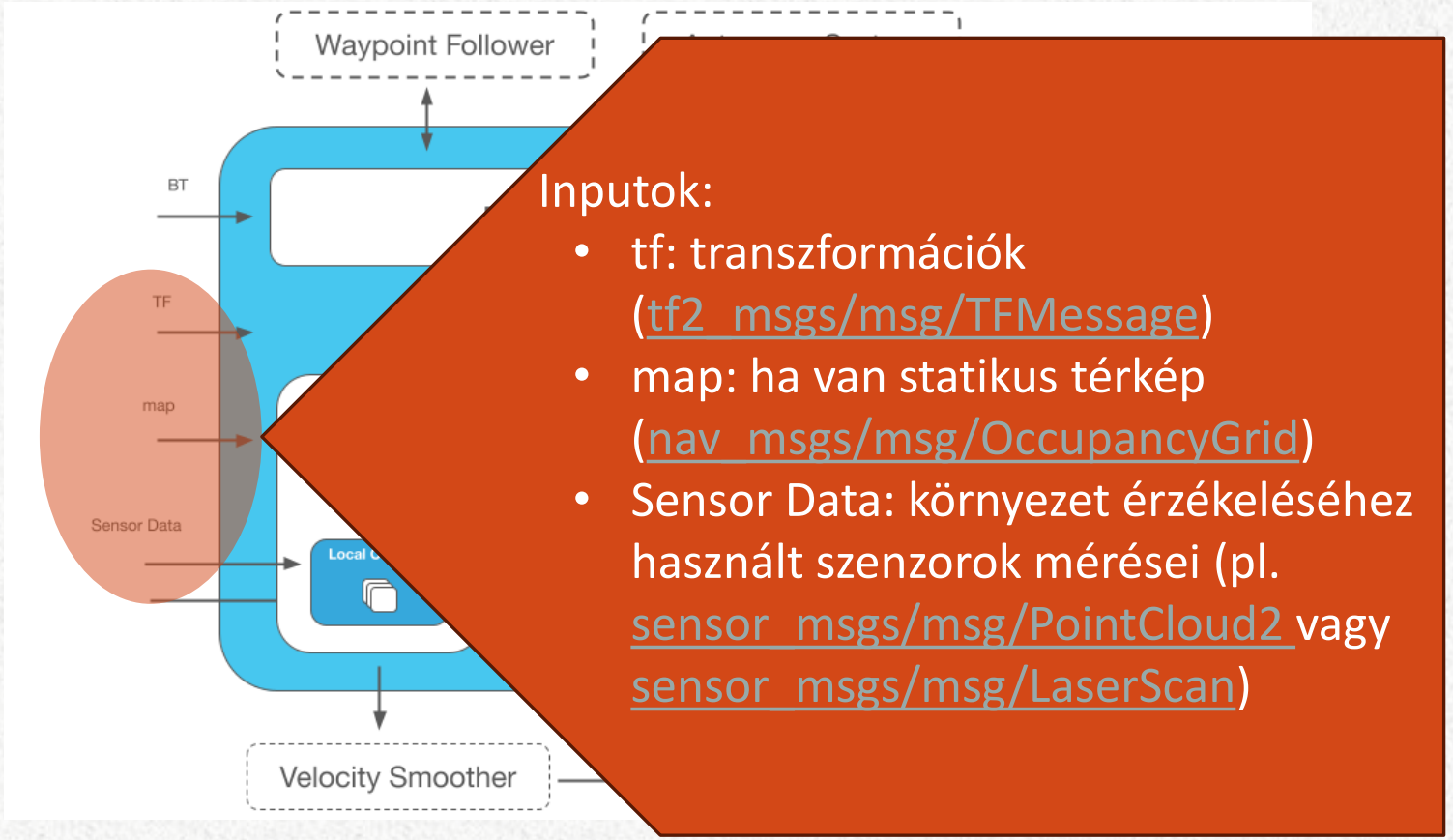
NAV 2 – Navigation Servers

Független, moduláris action szerverek:

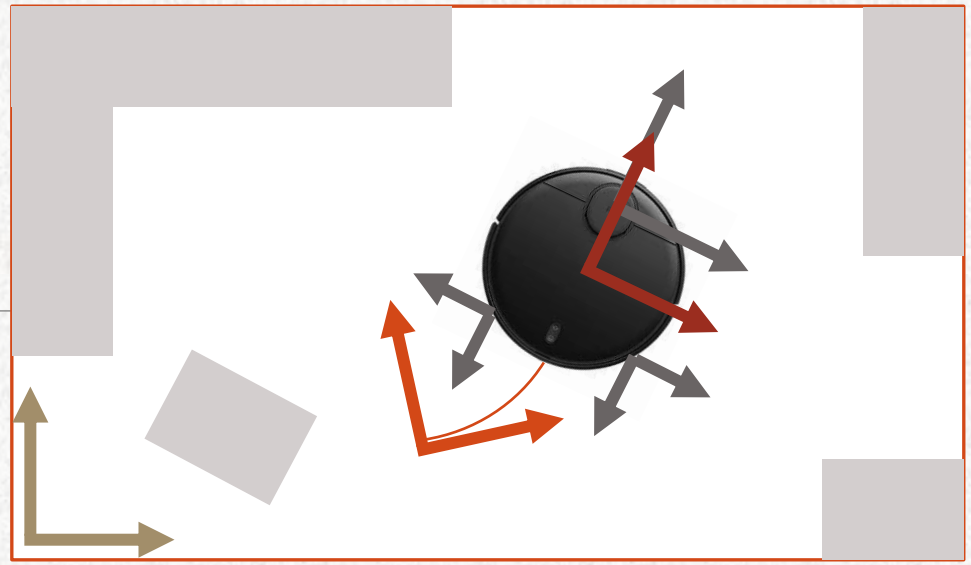
- Controller: irányít
- Planner: pályatervezés
- Behavior (vagy Recovery): hibakezelés
- Smoother: pálya minőségének javítása
- Route: útvonal számítás navigációs gráfból



NAV 2 – Bemeneti adatok

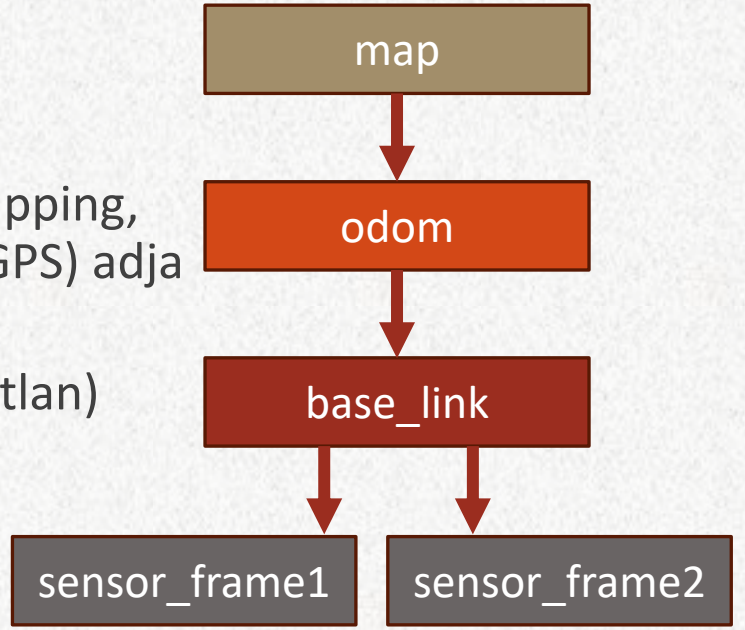


TF fa



Navigációs keretek:

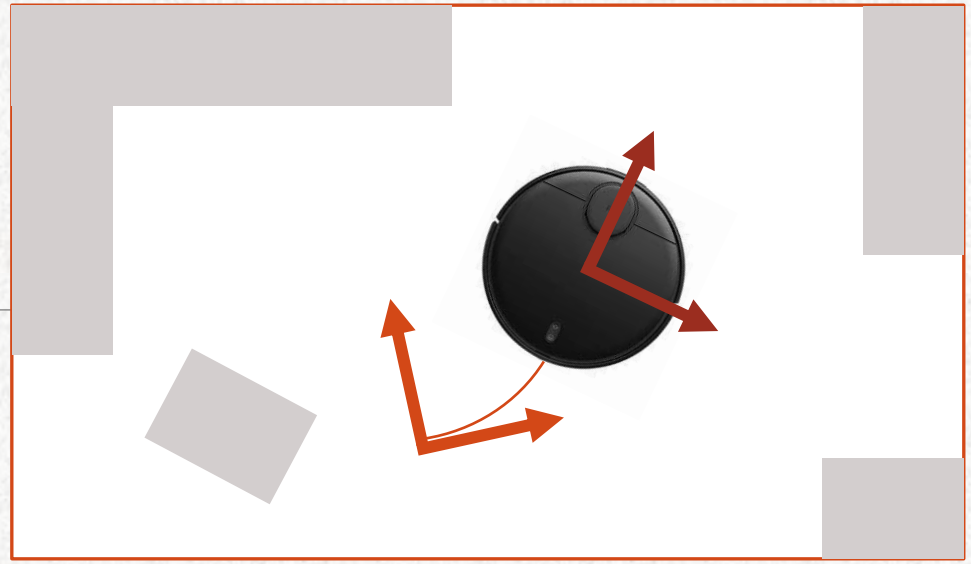
- map: térképhez rendelt keret
- odom: odometriai számítások kerete
- base_link: robothoz rögzített keret
- sensor_frames: érzékelők keretei (annyi, ahány érzékelő van)



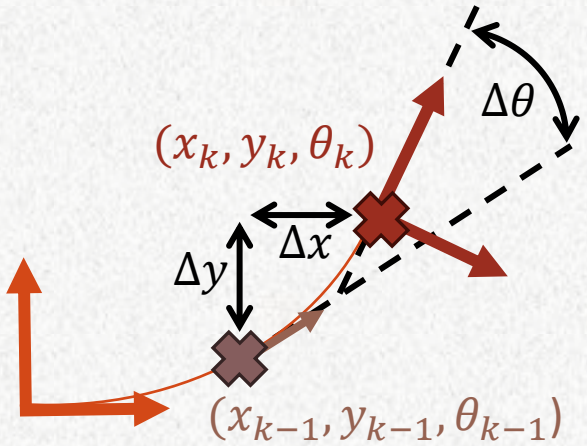
Transzformációk:

- map – odom: lokalizáció (pl. SLAM-Simultaneous Localization and Mapping, AMCL-Adaptive Monte-Carlo Localization) , pozícionáló rendszer (pl. GPS) adja meg
- odom – base_link: odometriai számítások alapján (hosszútávon pontatlan)
- base_link – sensor_frames: a robot felépítéséből következő statikus transzformációk (pl. URDF-ben lehet megadni)

Odometria

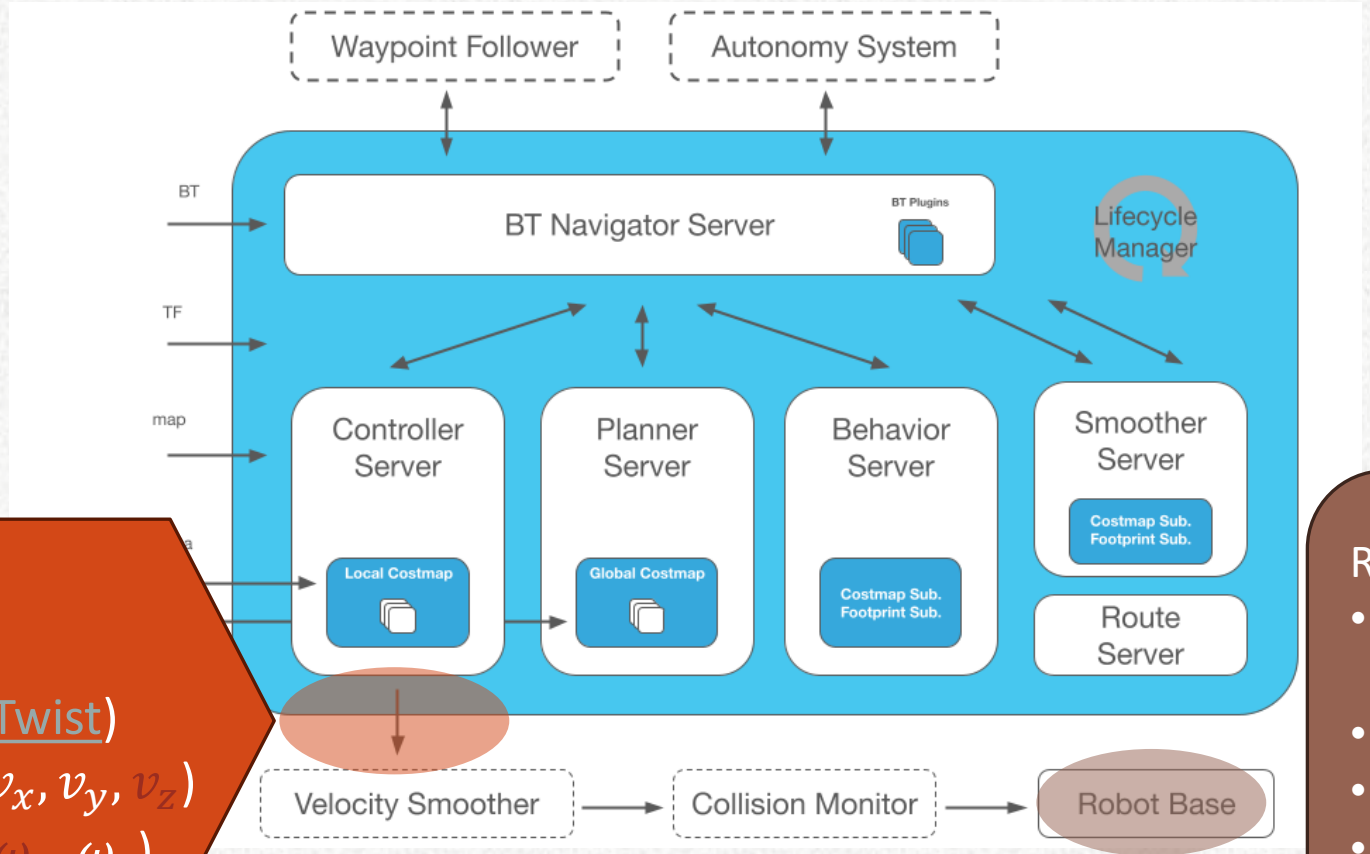


- Robot helyzetének számítása elemi elmozdulások összegzésével.
- Számítható például:
  - kerekek időegység alatti elmozdulásából
  - lépések alatti elmozdulásokból
  - IMU (Inertial Measurement Unit) által mért gyorsulás és szögsebesség adatokból
  - vizuális információból
- A hibával terhelt elemi elmozdulások összegzésekor a hiba is összeadódik. Hosszú távon alkalmazva pontatlan eredményt ad.
- Csak relatív pozíció és szöghelyzet adható meg vele.
- A számított mozgás folytonos, nincsenek benne diszkrét ugrások, ami a globális pozícionáló rendszerek esetén előfordulhat.



$$x_k = x_{k-1} + \Delta x$$
$$y_k = y_{k-1} + \Delta y$$
$$\theta_k = \theta_{k-1} + \Delta\theta$$

NAV 2 – Kimenet



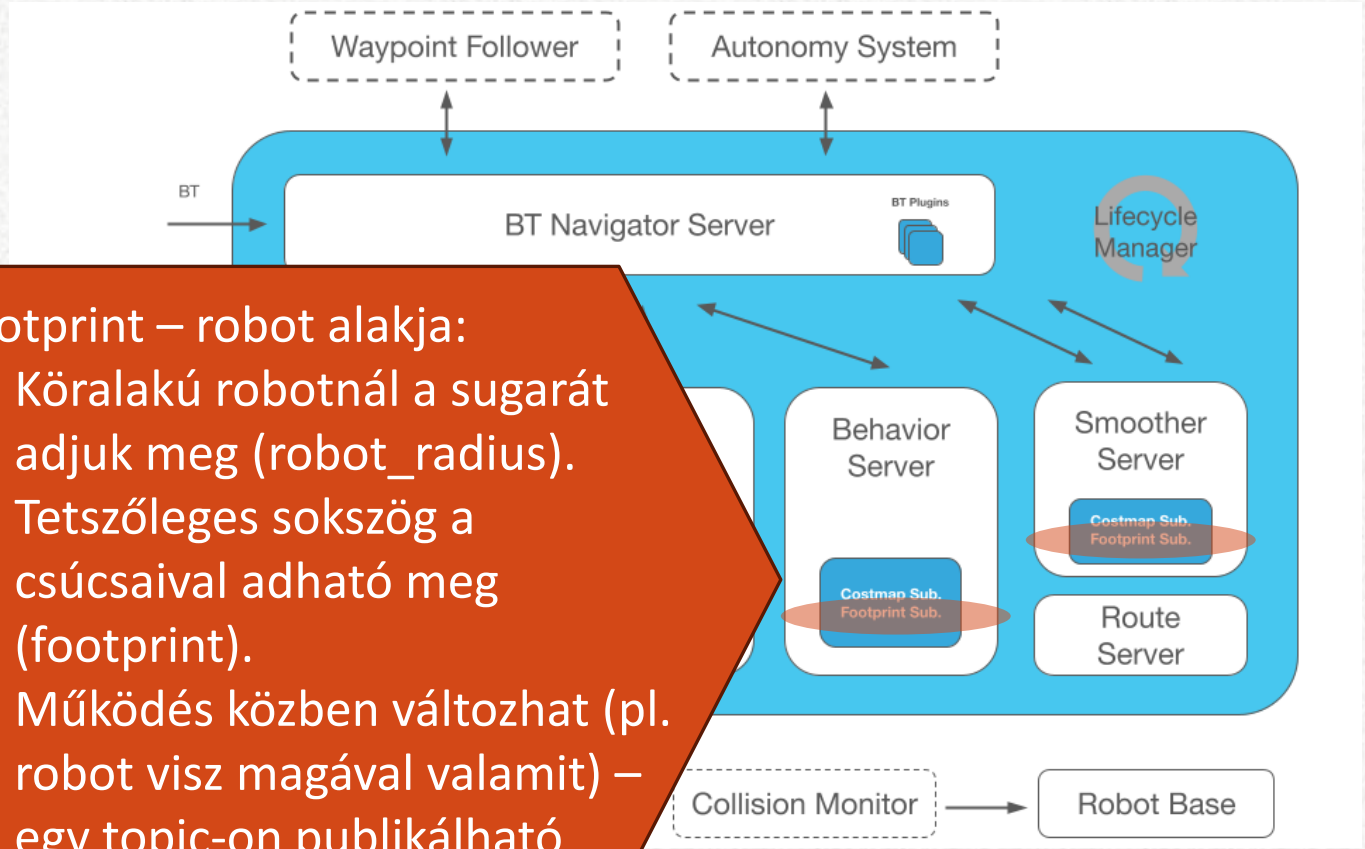
Kimenet:

- sebesség parancsok (geometry_msgs/msg/Twist)
  - lineáris sebesség (v_x, v_y, v_z)
  - szögsebesség ($\omega_x, \omega_y, \omega_z$)

Robot típusok:

- holonóm (omnidirekcionális)
- differenciális meghajtású
- lépegető
- autó-szerű (Ackermann)

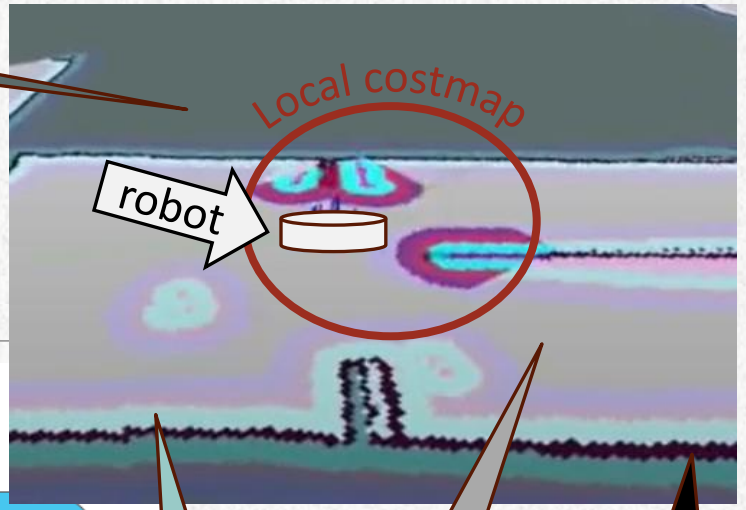
NAV 2 – Footprint

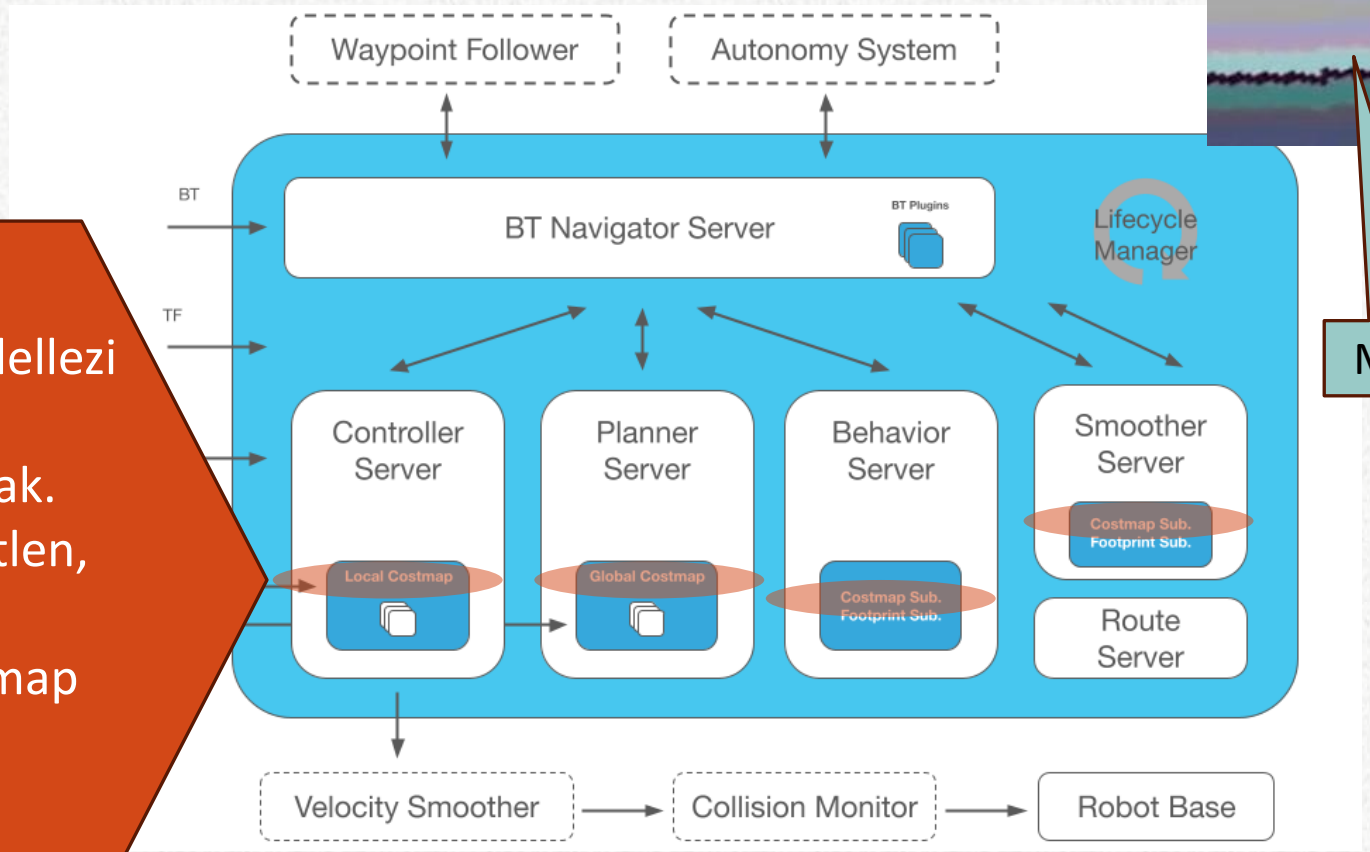


Footprint – robot alakja:

- Köralakú robotnál a sugarát adjuk meg (robot_radius).
- Tetszőleges sokszög a csúcsaival adható meg (footprint).
- Működés közben változhat (pl. robot visz magával valamit) – egy topic-on publikálható

NAV 2 – Costmap

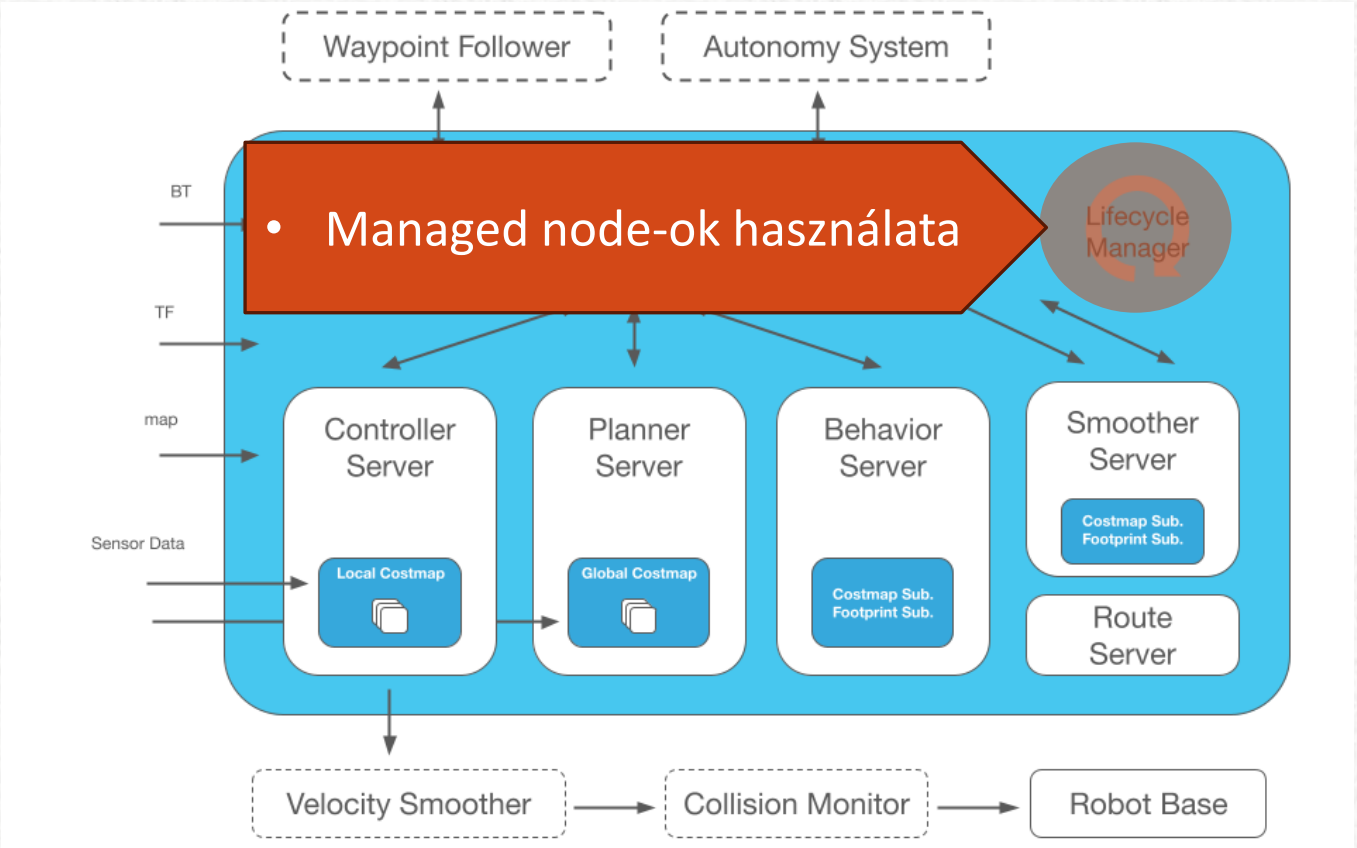




Costmap:

- Robot környezetét modellezi
- 2D grid, mely elemei költségeket tartalmaznak.
- Foglalt, szabad, ismeretlen, megnövelt mezők
- Lokális és globális costmap
- Több layer is lehet
- Filterekkel helyfüggő viselkedés valósítható meg.

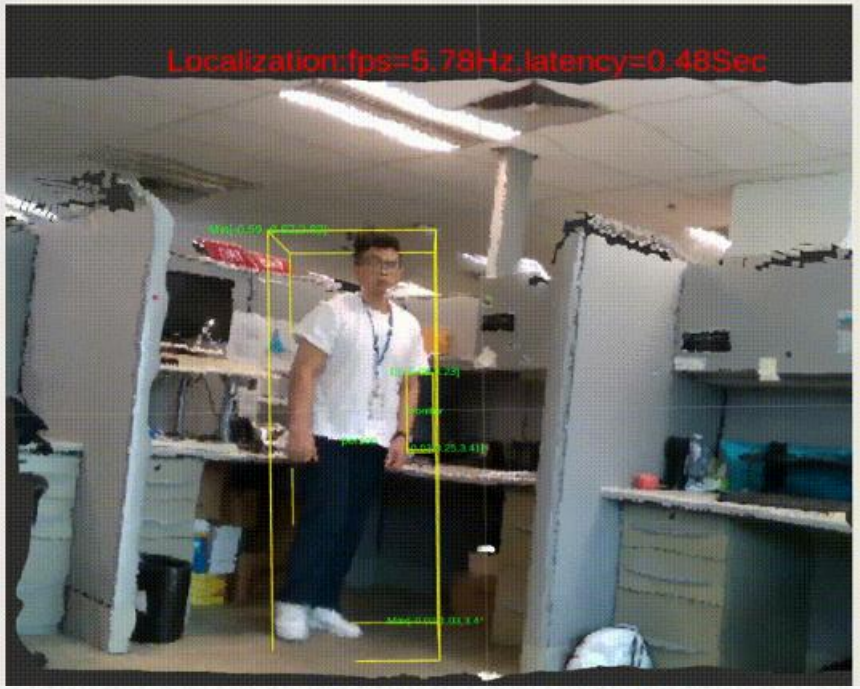
NAV 2 – Lifecycle



Gépi látás

Tipikus gépi látás feladatok

- Kamera kalibráció
- Sztereó képfeldolgozás, 3D látás
- Általános képfeldolgozási feladatok
- Szegmentáció
- Detekció
- Felismerés
- Követés
- Feature detektálás
- Transzformációk



https://dev.realsenseai.com/docs/object-analytics

image_common

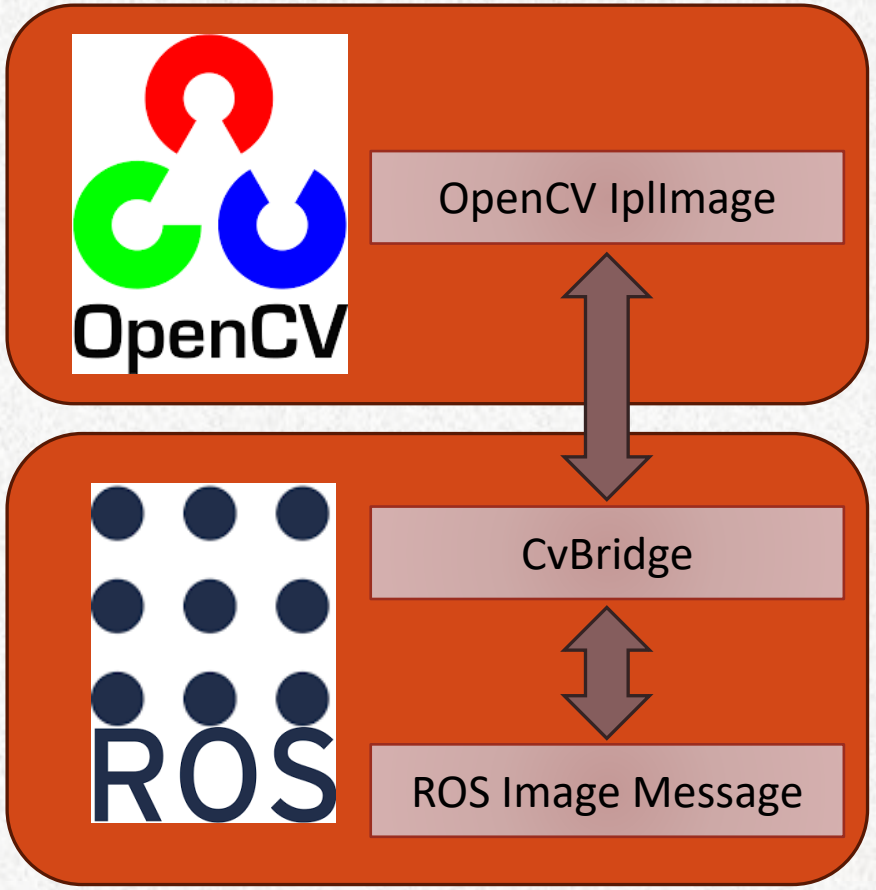
- Stack a képek kezeléséhez ROS1-ben
- Package-ek:
  - image_transport: Képek átvitele tömörített formában
    - publish-subscribe kommunikáció résztvevői használhatják (külön topic a különböző típusú átvitelekhez)
  - camera_calibration_parsers: Kamera kalibrációs paraméterek írása és olvasása
    - kamera driver használja
  - camera_info_manager: C++ interfész kamera kalibrációs információk mentéséhez, beállításához
    - kamera driver használja
  - camera_info_manager_py: Phython interfész (külön package, nem a stack része)
  - polled_camera: ROS interfész képek kéréséhez service híváson keresztül
- ROS2-ben is: image_common

image_pipeline

- Köztes lépések a kamerából érkező nyers kép és magas szintű vizuális feldolgozás között (kalibráció; mono, sztereó vagy mélységkép feldolgozás; vizualizáció)
- A stack package-ei:
  - camera_calibration: mono és sztereókamerák kalibrálása ROS rendszerben
  - image_proc: torzítás javítása és színkezelés mono kamera felvételén
  - stereo_image_proc: sztereó képek javítása
  - depth_image_proc: mélységképek feldolgozása (pl. pontfelhő előállítása)
  - image_view: képek (sztereó és diszparitás is) megjelenítése
  - image_rotate: kép elforgatása
  - image_publisher: egy kép vagy avi közzététele
- ROS2-ben is: image_pipeline

vision_opencv

- Stack a ROS1 – OpenCV interfészhez (valós idejű gépi látáshoz, magas szintű vizuális feldolgozáshoz)
- cv_bridge:
- Konvertálás ROS kép üzenetek és OpenCV képek között
- image_geometry:
- Python és C++ libraries
- kép és pixel geometriához kapcsolódó módszerek gyűjteménye
- sensor_msgs/CameraInfo paraméterei alapján
- ROS2-ben is: vision_opencv



vision_msgs

- Üzenet típusok gépi látáshoz
- Bounding Box: BoundingBox2D, BoundingBox2DArray, BoundingBox3D, BoundingBox3DArray
- Osztályozás: Classification2D, Classification3D
- Detekció: Detection2D, Detection2DArray, Detection3D, Detection3DArrray
- Objektum felismerés: ObjectHypothesis, ObjectHypothesisWithPose
- VisionInfo
- vision_msgs a github-on

perception_pcl

- PCL (Point Cloud Library) – ROS1 interfész stack-je
- pontfelhő és 3D geometriai feldolgozás
- pcl_ros: interfészek és eszközök ROS – PCL összekapcsoláshoz
- pcl_conversions: konverzió PCL adat típusok és ROS üzenet típusok között
- pcl_msgs: PCL-lel kapcsolatos üzenetek
- ROS2-ben is: perception_pcl