Zh várható anyaga:

package, node készítés, publisher, subscriber, timer használata

várható üzenettípusok:

- std_msgs:

1. Header
2. Float64
3. Bool
4. Int16

- sensor_msgs:

1. LaserScan (gyak6)
2. PointCloud2 (gyak6)

- visuaization_msgs:

1. Marker (gyak5, 6)
2. MarkerArray (gyak5, 6)

- nav_msgs:

1. Odometry (gyak3-sub gyak7-publish)
2. Path (gyak3)

- geometry_msgs:

1. Point, Pose, PoseStamped (gyak3, 6)
2. Quaternion (gyak7)
3. Twist (gyak7)

tf listener, buffer(gyak11), broadcaster(gyak7), static_broadcaster(gyak10), TransformStamped(gyak7)

launch fájl készítés, paraméter beadás/bekérés(gyak3, 7)

rosbag visszajátszás

rviz használata

Package létrehozás:

1. Terminalban a **workspace/src-n** belül kell létehozni a pekidzset:

```
ros2 pkg create --build-type ament_python package name
```

2. A package nevével megyegyző mappába kell írni a nodeokat. illetve kell csinálni a packagen belül egy launch és egy rviz mappát is. Rviz kód az Rvzi mappába másolható a gyak11-ből. Launch mappa másolható bárhonnét igazából.

3. Launch file nevét: package_name.launch.xml-re változtatni. Az rviz args részében is beírni a dolgokat. Rviz file neve is legyen a package neve

4. setup.py file, data files részébe beírni a szokásos két sort:

```
data_files=[
    ('share/ament_index/resource_index/packages',
        ['resource/' + package_name]),
    ('share/' + package_name, ['package.xml']),
    ('share/' + package_name + '/launch', ['launch/zh_gyak_minta.launch.xml']),
    ('share/' + package_name + '/rviz', ['rviz/zh_gyak_minta.rviz'])
]
```

itt is át kell írni a launch file és rviz flie neveket.

Kész a package: 1 pont xd

Node készítés alap:

```
import rclpy
from rclpy.node import Node


class MyCustomNode(Node):  # MODIFY NAME

    def __init__(self):
        super().__init__("node_name")  # MODIFY NAME


def main(args=None):
    rclpy.init(args=args)
    node = MyCustomNode()  # MODIFY NAME
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == "__main__":
    main()
```

a setup.py mappa enty pointjába feljegyezni a node belépési pontját így:

```
entry_points={
    'console_scripts': [
        "node_1=zh_gyak_minta.node_1:main" #node exec név = mappa amiben a .py file van (pkg name).node .py file neve:main
    ]
```

launch mappába létrehozzuk a node-ot:

```
<node pkg="zh_gyak_minta" exec="node_1" name="node_1" output="screen">

</node>
```

pkg: pkg neve
exec: ami a setup.py console scripts egyenlet bal oldalán van
name: kurva mindegy ez lesz a node neve és kész
output: mindig minden körülmények között screen

Tesztelés h jó e a node:

```
class Node1(Node):  # MODIFY NAME

    def __init__(self):
        super().__init__("node_1")  # MODIFY NAME
        self.get_logger().info("mukodik a node")
```

self.get_logger().info(“valami”)

most kell buildelni (illetve akkor amikor módosítod a launch filét és vagy setup.py-t)! Ctrl shift B, colcon build (csak ha vscode a workspaceben van megnyitva)

ÚJ TERMINÁLT KELL NYITNI!
És ezt beírni:
ros2 launch package_név launch_file_név

feliratkozás:

```
    #subscriber
    self.sub = self.create_subscription(Odometry, "/odom", self.odom_callback, 1)

def odom_callback(self, msg:Odometry):
    pass
```

sub változó lesz a subscription. create_subscription(message type, topic név, callback függvény, 1)

callback function mindig meg kell hívni, a classon belül kell definiálni, de nem az initen belül.
Egyik argumentum self, másik msg: message type
Callback function: mindig amikor megkapod az üzenetet a függvény meghívódik és a kapott üzenetet az msg változóba elmenti

hogy néz ki az adott message típus?

ros2 interface show (pl.:) nav_msgs/msg/Odometry (ilyenkor import úgy néz ki h: from nav_msgs.msg import Odometry)

hogyan kell publisholni:

mikor kell a use simtime: ha az rviz valamiért rákot kap

```
<node pkg="zh_gyak_minta" exec="node_1" name="node_1" output="screen">
    <param name="use_sim_time"  value="true"/>
</node>
```

timer létrehozása: (gyak2 publisher.py)

```
class Publisher_Node(Node):
    def __init__(self):
        super().__init__('publisher_node')

        self.publisher = self.create_publisher(String, "chatter", 10)

        self.timer = self.create_timer(1.0, self.timer_callback)

    def timer_callback(self):
        message = "Hello, ROS2!"
        self.get_logger().info(message)
        self.get_logger().warn(message)
        self.get_logger().error(message)

        msg = String()
        msg.data = message
        self.publisher.publish(msg)
```

create timer argumentumai: idő, hogy milyen gyakran legyen meghívva, callback függvény
callback függvénybe lehet írni mi történjen ilyenkor amikor meghíbódik a cuci

paraméter létrehozás:

a) launch file:

```
<node pkg="gyak3" exec="path.py" name="path_node" output="screen">
    <param name="use_sim_time" value="true"/>
    <param name="path_topic_name" value="/path"/>
    <param name="odom_topic_name" value="/odom"/>
    <param name="max_size" value="500"/>
</node>
```

ilyenkor maga a nodeban is kell a paramétereket deklarálni:

```python
class PathCreator(Node):
    def __init__(self):
        super().__init__('odom_path')

        # Declare parameters with default values
        self.declare_parameter('path_topic_name', '/path')
        self.declare_parameter('odom_topic_name', '/odom')
        self.declare_parameter('max_size', 1500)

        path_topic_name = self.get_parameter('path_topic_name').value
        odom_topic_name = self.get_parameter('odom_topic_name').value
        self.max_size = self.get_parameter('max_size').value
```

b) eset: yaml file:

path_param.yaml

src > mgm_gyak > gyak4 > config > path_param.yaml

```yaml
/**:
  ros__parameters:
    path_topic_name: /path
    max_size: 500
    use_sim_time: true
```

ilyenkor a gyászkeret faszom launchfile:

```xml
<!-- Start path node -->
<node pkg="gyak3" exec="path.py" name="path_node" output="screen">
    <param from="$(find-pkg-share gyak4)/config/path_param.yaml"/>
</node>
```

Transzformációk:

1.: tfBuffer, tfListener használata (gyak11 save path, balass mintazh megold)

```python
import tf2_ros
from tf2_geometry_msgs.tf2_geometry_msgs import do_transform_pose
from tf_transformations import euler_from_quaternion
```

Akkor használjuk ha pl: van egy globális koordináta rsz. Meg van egy lokális koord. Rsz. És tudjuk köztük a kapcsolatot. Ha megvannak pl. a lokális rsz-ben a sebességek akkor a tfBuffer segítségével. Meglévő pontok transzformációja egyik frameből a másikba.

```python
self.tfBuffer = tf2_ros.Buffer()
self.listener = tf2_ros.TransformListener(self.tfBuffer, self)
```

Listenernek meg kell adni amire hallgat. Egyébként nem kell meghivni sehol de ide kell írni xd

Meg kell vizsgálni hogy lehet e transzormálni:

```python
if self.tfBuffer.can_transform("map", msg.header.frame_id, rclpy.time.Time(), rclpy.duration.Duration(seconds=0.1)):
```

argumentumok: melyik framebe, melyik frameből, másik kettő mindig ugyanaz csak seconds változik max

ha a can transform igaz akkor utána:

```
trans_2map = self.tfBuffer.lookup_transform("map", msg.header.frame_id, rclpy.time.Time())
```

majd pedig egy változóba elrakhatjuk, a régi koordináta rendszerben lévő poziciót, az új szerintibe az ha trans_2map a transzformációs mátrix

```
actual_pose = do_transform_pose(msg.pose.pose, trans_2map)
```

2.:broadcaster: (gyak7 diff robot)

Minden időpillanatban más a kapcsolat a két frame között. Lehet x y z irányú eltolás és elforgatás is

```
self.broadcaster = tf2_ros.TransformBroadcaster(self)
```

```
odom = Odometry()
odom.header.stamp = self.get_clock().now().to_msg()
odom.header.frame_id = "odom"
odom.child_frame_id = "base_link"
odom.pose.pose.position.x = self.x
odom.pose.pose.position.y = self.y

# Orientation from yaw

q = quaternion_from_euler(0, 0, self.yaw)
odom.pose.pose.orientation.x = q[0]
odom.pose.pose.orientation.y = q[1]
odom.pose.pose.orientation.z = q[2]
odom.pose.pose.orientation.w = q[3]
self.pub_odom.publish(odom)

tf_stamped = TransformStamped()
tf_stamped.header = odom.header
tf_stamped.child_frame_id = odom.child_frame_id

tf_stamped.transform.translation.x = odom.pose.pose.position.x
tf_stamped.transform.translation.y = odom.pose.pose.position.y
tf_stamped.transform.translation.z = odom.pose.pose.position.z

tf_stamped.transform.rotation = odom.pose.pose.orientation

self.broadcaster.sendTransform(tf_stamped)
```

tf_stamped TransfromStamped típusú message ez teremt kapcsolatot az odom és base link framek között.
Headerjébe: amiből
child: amibe

utána eltolások és elmozdulások

utána mint egy publisher kiküldjük a broadcasterrel a TransfromStamped type messaget

3.: staticBoradcaster:

Ha a különbség a kettő frame között állandó. Ha nincs rotation a két frame között akkor is w értékét egyre kell rakni mert különben hibát ír ki

```
self.static_broadcaster = tf2_ros.StaticTransformBroadcaster(self)

front_wheel = TransformStamped()
front_wheel.header.frame_id = "base_link"
front_wheel.child_frame_id = "front_wheel"

front_wheel.transform.translation.x = self.wheel_base
front_wheel.transform.rotation.w = 1.0

self.static_broadcaster.sendTransform(front_wheel)
```

Message type shitek: