9.1 Experiencing ROS2

This is the end of our journey!. And last but not least... we're going to experience the amazing [Robot Operating System 2 (ROS2) ecosystem (https://docs.ros.org/en/humble/index.html)](https://docs.ros.org/en/humble/index.html).

For that, we are going to use the [MVSim (https://mvsimulator.readthedocs.io/en/latest/index.html)](https://mvsimulator.readthedocs.io/en/latest/index.html), a lightweight MultiVehicle Simulator:

> Lightweight, realistic dynamical simulator for 2D (“2.5D”) vehicles and robots. It is tailored to analysis of vehicle dynamics, wheel-ground contact forces, and accurate simulation of typical robot sensors. This project includes C++ and Python libraries, the standalone CLI application mvsim, and ROS 1 and ROS 2 nodes, and it is licensed under the permissive 3-clause BSD License.

0:00 / 0:14

9.1.1 Preparing the path

For working with ROS2 and MVSim you will need:

- Ubuntu 22.04 (you can natively install it, or through a Virtual Machine (e.g., VMWare or WSL).
- ROS2 ([https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html (https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html)](https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html)). You will need both the *Desktop installation* and *Development tools*.
- MVSim ([https://mvsimulator.readthedocs.io/en/latest/install.html (https://mvsimulator.readthedocs.io/en/latest/install.html)](https://mvsimulator.readthedocs.io/en/latest/install.html)).

Important! ROS2 relies on a number of environment variables to work properly, so we need to load them in our terminal. This can be done through the `source` command, which executes the commands from a file in the current shell:

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

However, executing the command in that manner requires you to load the variables each time you open a new terminal. To solve this, you can add the `source` command to your `.bashrc` file, which is automatically executed every time a new terminal session is started, ensuring that the necessary variables are consistently loaded:

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

Later on you will repeat this but with the environment variables of your workspace.

You can also install terminator for managing terminals:

```
sudo apt install terminator
```

9.1.2 Insight into MVSim

MVSim simulates a world as defined in a `.world` file. It could be illustrative to take a look at any of them, hey are placed at: `/opt/ros/humble/share/mvsim/mvsim_tutorial` . This file tells MVSim everything about the world, from obstacles, to robots, sensors, and other objects in the virtual environment. Depending on the world you are running, the available topics can change.

0:00 / 0:10

The following is an example for the world shown above:

```xml 0
```

```
    <!-- GUI options -->
    <gui>
            <ortho>false</ortho>
            <show_forces>true</show_forces>  <force_scale>0.01
</force_scale>
            <cam_distance>35</cam_distance>
            <fov_deg>60</fov_deg>
            <refresh_fps>20</refresh_fps>
            <!-- <follow_vehicle>r1</follow_vehicle> -->
    </gui>

    <!-- Light parameters -->
    <lights>
    </lights>

    <!-- =======================
               Scenario definition
         ======================= -->
    <element class="occupancy_grid">
            <!-- File can be an image or an MRPT .gridmap file
-->
            <file>uma_campus.gridmap.gz</file>

            <!--<show_collisions>true</show_collisions>-->
    </element>

    <!-- ground grid (for visual reference) -->
    <element class="ground_grid">
            <floating>true</floating>
    </element>

    <!-- =============================
               Vehicle classes definition
         ============================= -->
    <include file="../definitions/ackermann.vehicle.xml" />

    <!-- =======================
               Vehicle(s) definition
         ======================= -->
    <vehicle name="r1" class="car_ackermann">
            <init_pose>0 0 0</init_pose>  <!-- In global coord
s: x,y, yaw(deg) -->
            <init_vel>0 0 0</init_vel>  <!-- In local coords:
```

MVSim also comes with a number of launch files for working with these demo worlds. For example, `demo_1robot.launch` :

```xml
<?xml version="1.0"?>
<!-- ROS1 launch file. See *.launch.py files for ROS2 launch files -->
<launch>
        <arg name="world_file" default="$(find mvsim)/mvsim_tutorial/demo_1robot.world.xml" />
        <arg name="mvsim_do_fake_localization" default="true"/>

        <node pkg="mvsim" type="mvsim_node" name="mvsim_simulator" output="screen">
                <param name="world_file" value="$(arg world_file)"/>
                <param name="do_fake_localization" value="$(arg mvsim_do_fake_localization)"/>
        </node>

        <node pkg="rviz" type="rviz" name="rviz" args="-d $(find mvsim)/mvsim_tutorial/demo_1robot.rviz"/>
</launch>
```

With its python-based counterpart:

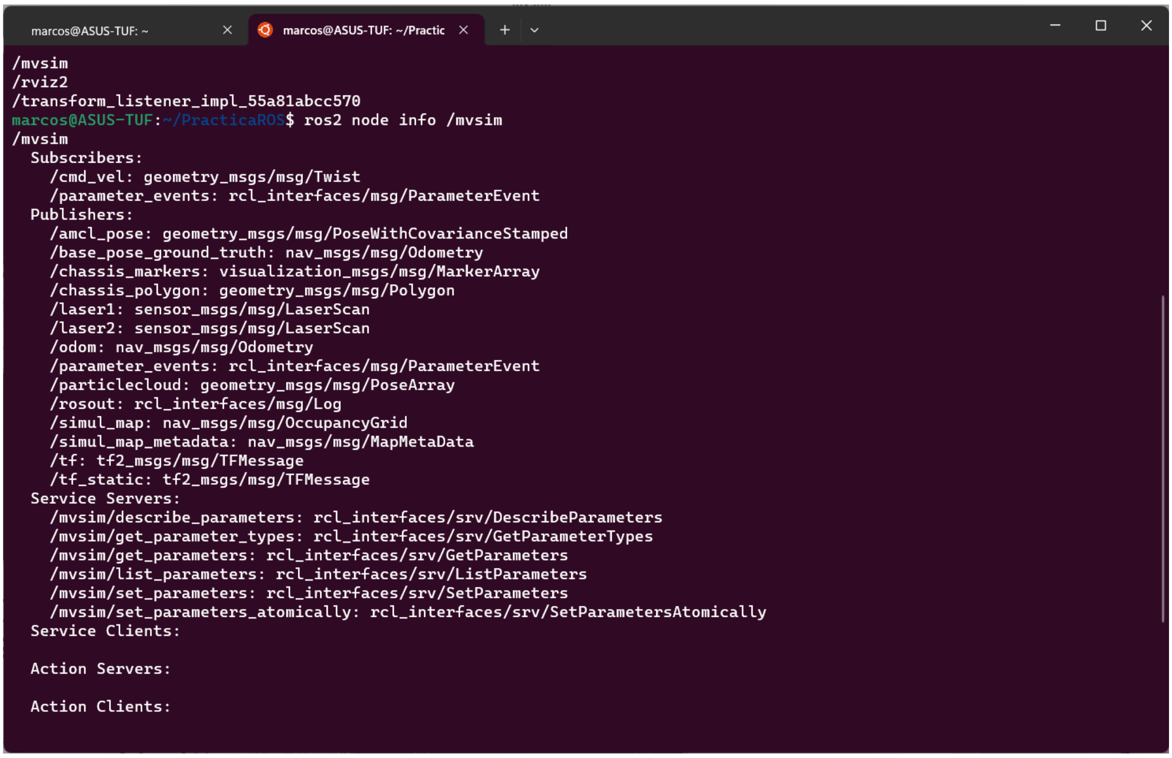
```python
# ROS2 launch file

from launch import LaunchDescription
from launch.substitutions import TextSubstitution
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument
from ament_index_python import get_package_share_directory
import os


def generate_launch_description():
    mvsimDir = get_package_share_directory("mvsim")
    #print('mvsimDir: ' + mvsimDir)

    # args that can be set from the command line or a default
```

ASSIGNMENT 1: Analyzing topics

What to do?

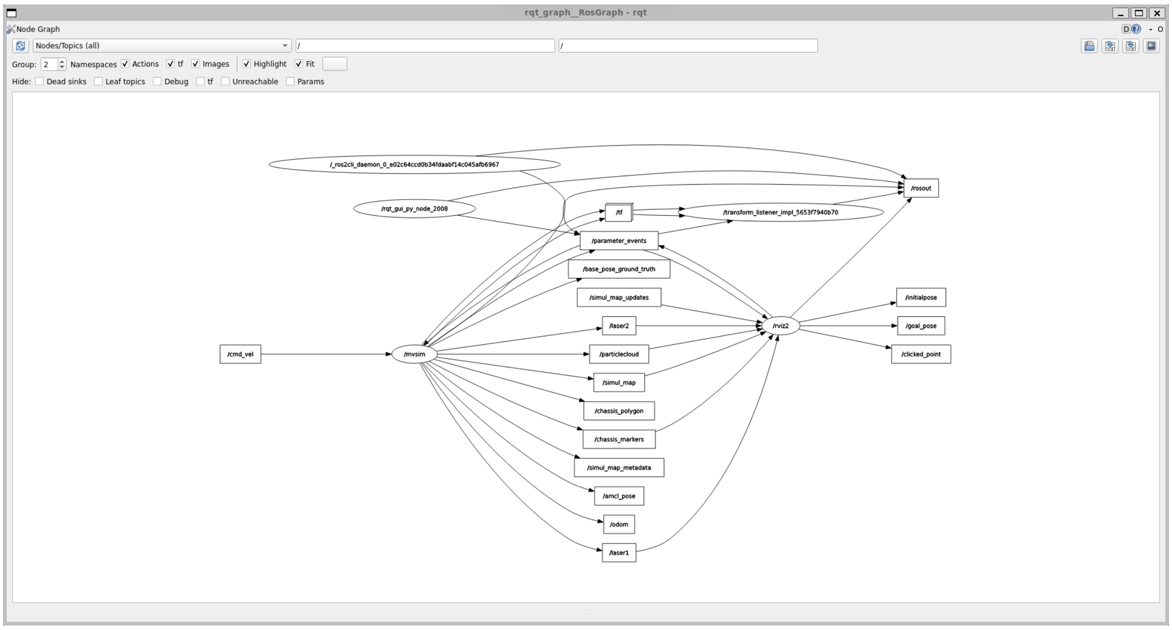
- Run the demo world with a robot with `ros2 launch mvsim demo_1robot.launch.py`
- Check the available topics and include them in a list below, also stating which nodes publish/subscribes to them (rqt_graph can help here).

Los topics pueden verse con el comando "ros2 node info /mvsim"

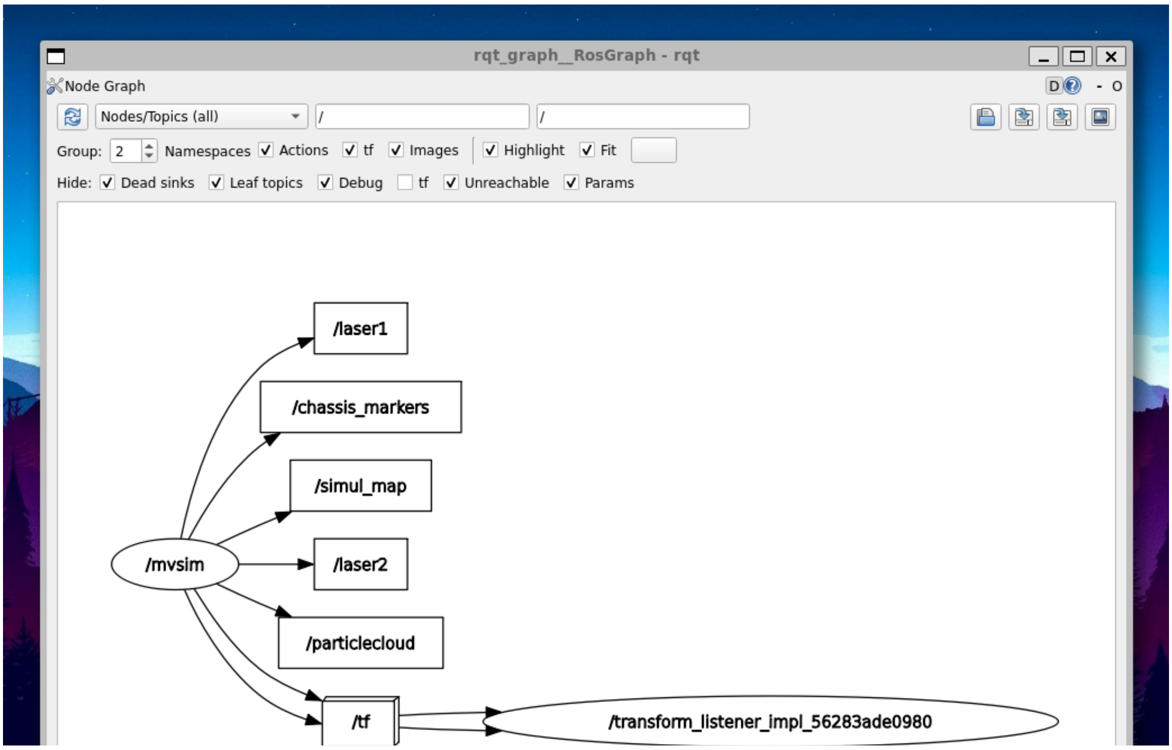


```
/mvsim
/rviz2
/transform_listener_impl_55a81abcc570
marcos@ASUS-TUF:~/PracticaROS$ ros2 node info /mvsim
/mvsim
  Subscribers:
    /cmd_vel: geometry_msgs/msg/Twist
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /amcl_pose: geometry_msgs/msg/PoseWithCovarianceStamped
    /base_pose_ground_truth: nav_msgs/msg/Odometry
    /chassis_markers: visualization_msgs/msg/MarkerArray
    /chassis_polygon: geometry_msgs/msg/Polygon
    /laser1: sensor_msgs/msg/LaserScan
    /laser2: sensor_msgs/msg/LaserScan
    /odom: nav_msgs/msg/Odometry
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /particlecloud: geometry_msgs/msg/PoseArray
    /rosout: rcl_interfaces/msg/Log
    /simul_map: nav_msgs/msg/OccupancyGrid
    /simul_map_metadata: nav_msgs/msg/MapMetaData
    /tf: tf2_msgs/msg/TFMessage
    /tf_static: tf2_msgs/msg/TFMessage
  Service Servers:
    /mvsim/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /mvsim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /mvsim/get_parameters: rcl_interfaces/srv/GetParameters
    /mvsim/list_parameters: rcl_interfaces/srv/ListParameters
    /mvsim/set_parameters: rcl_interfaces/srv/SetParameters
    /mvsim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

Los cuales generan el siguiente árbol:



En vista simplificada:



ASSIGNMENT 2: Manually controlling the robot

One of the topics MVSim is suscribed to is `/cmd_vel`, so we can send motion commands to the robot:

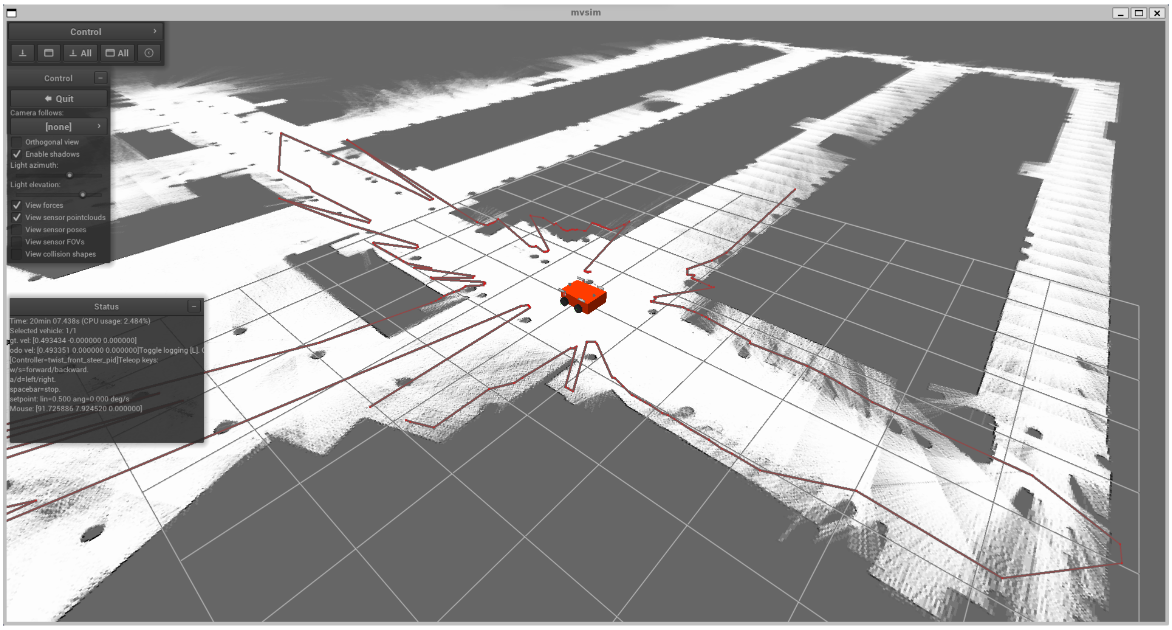
```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.5, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

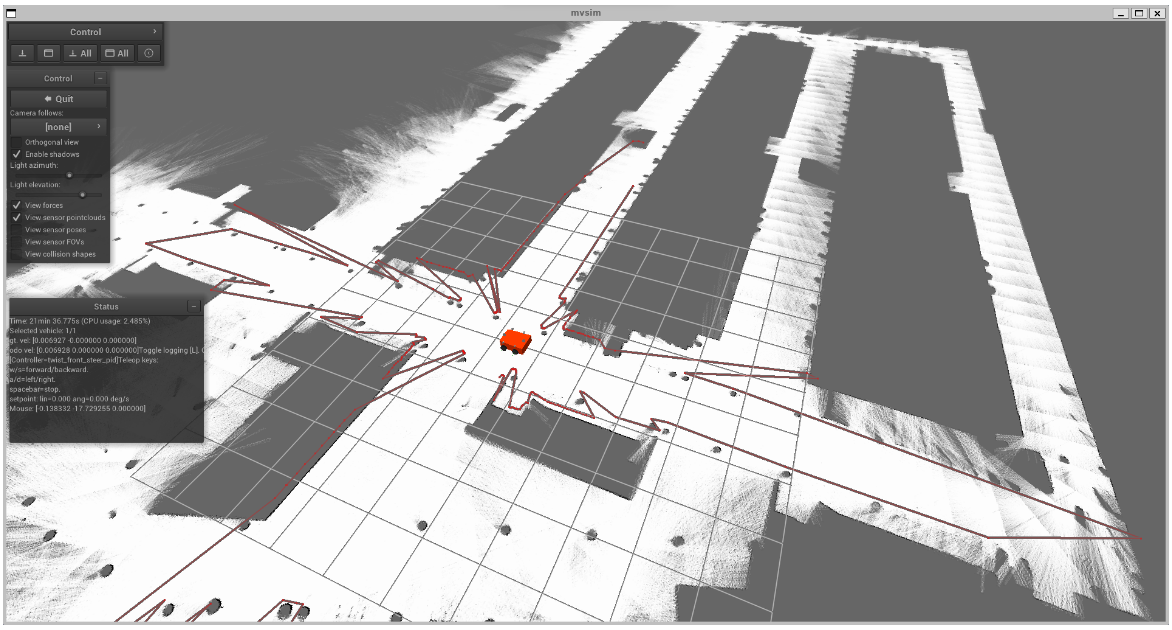
What to do?

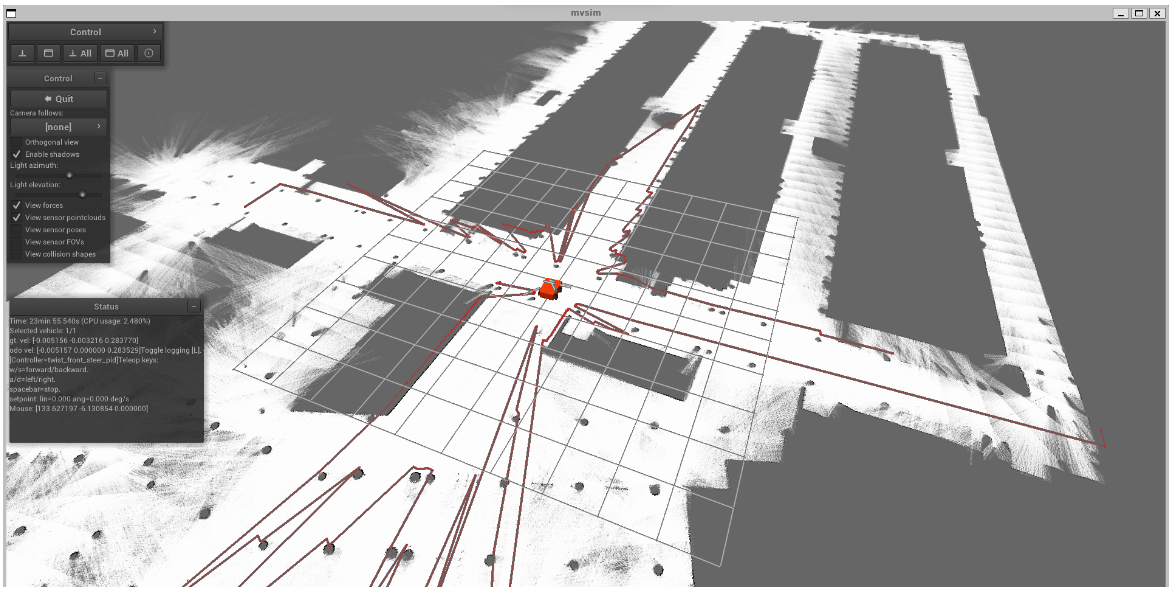
- Send some motion commands, and take some screenshots of them and the robot. You can send a motion command full of zeros to stop it!

Take care commanding the robot, it can crash otherwise!

Comenzamos haciendo que el robot avance por el pasillo. Después girará a la izquierda y como el giro no es del todo preciso (no es de 90º) cuando retome su camino en línea recta se chocará con la pared. También hay que destacar que en su camino a chocarse detectará un pilar que ligeramente modificará su trayectoria, puesto que lo evitará en cierta medida.







ASSIGNMENT 3: Implementing reactive navigation

Finally, we are going to implement a reactive navigation technique so our robot can safely move through the environment. Recall this pipeline when developing ROS 2 software:

image.png

Some handy ROS 2 links:

- [Workspace creation tutorial (https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html)](https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html)
- [Package creation tutorial (https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html)](https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html)
- [Example of a simple publisher and subscriber (https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html)](https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html)
- [List of ROS 2 tutorials (https://docs.ros.org/en/humble/Tutorials.html)](https://docs.ros.org/en/humble/Tutorials.html)

What to do?

1. Create a **ROS 2 workspace**.
   - Once created, add its `setup.bash` file to your shell configuration: `echo "source ~/humble_ws/install/setup.bash" >> ~/.bashrc`
2. Create a **package** (it can be python or c++ based). The name is of your choice!
   - CAUTION! this has to be done in the `src` directory of your workspace.
   - Customize the `package.xml` file.
3. Craete a **node**, and implement in it the Potential Fields technique we played with in the last practical session. For example, the next line creates a package called `reactive_nav` with a node named `potential_fields_nav`, using `ament_python` to specify that it will be a Python package:

   ```
   ros2 pkg create --build-type ament_python --node-name potential_fields_nav reactive_nav
   ```

During the node implementation notice that:

- You can perceive the environment through the information provided by the laser. This is a **subscription**.
- You can also send motion commands. Here the node plays the role of a **publisher**.
- The robot true pose is also provided to you in a topic (`/base_pose_ground_truth`). Second **subscription**.
- You can use RViz to listen to new goal destinations through the `/goal_pose` topic. Third and last **subscription**.
- You have to include the dependencies of your package into the `package.xml` .
- Check that your node appears in `setup.py` as an entry point.

Explain all you did and the obtained results. You can include short videos or gifs to illustrate the technique's behaviour.

Tras crear el workspace, el package y el nodo, he realizado las subscripciones y publicaciones a los topics que se indican. Posteriormente, creo las funciones de callback necesarias para calcular la fuerza repulsiva, atractiva y total que se ejercerán sobre el robot.

Sin embargo, tras mucho intentar mejorarlo no he sido capaz de hacer que puedas establecer la meta en una posición alejada (que no sea en linea recta o tras un giro). Como se muestra en el video, puedo hacer que vaya paso a paso siguiendo una serie de poses

0:00 / 0:31

Contenido del fichero principal: Potential Fields Node

In [ ]:
```
######################################
####### Potential Fields Node #######
#### Autor: Marcos Hidalgo Baños ####
######################################

import numpy as np
import rclpy
from geometry_msgs.msg import Pose, Twist
from geometry_msgs.msg import PoseStamped
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry


################################
#### Definiciones globales ####
################################

KGOAL = 2.0
KOBSTACLE = 12.0
RADIUS_OF_INFLUENCE = 5
MAX_DIST = 1.0

goal_pose = Pose()
true_pose = Pose()
motion_command_msg = Twist()


##############################
#### Funciones auxiliares ####
##############################

def repulsive_force(true_pose, true_angle, angle_inc, angle_min, dist_

    # Distinguimos aquellos landmarks que son detectables...
    seen = [obj for obj in dist_observ if obj <= RADIUS_OF_INFLUENCE]

    if seen:
        # ... obtenemos sus angulos a partir de las mediciones del las
        angles = [angle_min + i * angle_inc for i in range(len(seen))]
        landmark = []

        # Iteramos por cada observacion
        for idx in range(len(seen)):

            # Reconstruimos la observacion
            z = (seen[idx] * np.cos(angles[idx]),
                 seen[idx] * np.sin(angles[idx]))

            # Reconstruimos el landmark a partir de la pose del robot y
            landmark.append((true_pose.position.x + z[0] * np.cos(true_
                             true_pose.position.y + z[0] * np.sin(true_

        # Calculamos la fuerza repulsiva
        FRep = (KOBSTACLE * sum((1/d - 1/RADIUS_OF_INFLUENCE)/(d**2) *
                KOBSTACLE * sum((1/d - 1/RADIUS_OF_INFLUENCE)/(d**2) *
    else:
        FRep = (0,0) # No hay repulsividad si no hay obstaculos

    return FRep

def attractive_force(goal_error, dist_goal):
```

```
    FAtt = ((-KGOAL * goal_error[0]) / dist_goal,
            (-KGOAL * goal_error[1]) / dist_goal)

    return FAtt

###############################
#### Funciones de callback ####
###############################

def goal_pose_callback(msg):
    global goal_pose
    goal_pose = msg.pose


def base_pose_callback(msg):
    global true_pose, true_twist
    true_pose = msg.pose.pose
    true_twist = msg.twist.twist


def laser_callback(msg, motion_command_publisher):
    global goal_pose, motion_command_msg, true_pose, true_twist
    dist_goal = np.sqrt((true_pose.position.x - goal_pose.position.x)*

    # Comprobamos que no hemos llegado al destino...
    if dist_goal > MAX_DIST:

        dist_observ = msg.ranges
        angle_min = msg.angle_min
        angle_inc = msg.angle_increment
        true_angle = np.arctan2(true_twist.linear.y, true_twist.linear

        # Calculamos la fuerza de repulsion...
        FRep = repulsive_force(true_pose, true_angle, angle_inc, angle_

        # Calculamos la fuerza atractiva...
        goal_error = (true_pose.position.x - goal_pose.position.x, tru
        FAtt = attractive_force(goal_error, dist_goal)

        # ... para obtener la total
        FTotal = FAtt + FRep

        # Obtenemos la velocidad y angulo
        v = ((FTotal[0] * np.cos(true_angle) + FTotal[1] * np.sin(true_
             (-FTotal[0] * np.sin(true_angle) + FTotal[1] * np.cos(tru
        Theta = np.arctan2(v[1], v[0])

        # Le mandamos al robot los comandos de movimiento

        motion_command_msg.linear.x = np.sqrt(v[0]**2 + v[1]**2)
        motion_command_msg.angular.z = Theta

    else:

        # Mandamos un movimiento nulo (quedarse)
        motion_command_msg.linear.x = 0.0
        motion_command_msg.angular.z = 0.0

    motion_command_publisher.publish(motion_command_msg)
```

```
#######################
#### Programa Main ####
#######################

def main():

    # Paso 1: Establecer el nuevo nodo
    print("Creando e inicializando nodo...")
    rclpy.init()
    nodo = rclpy.create_node('motion_command_node')

    # Paso 2: Añadir las conexiones a topics necesarias
    print("Estableciendo subscripciones y publicaciones...")
    motion_command_publisher = nodo.create_publisher(Twist, 'cmd_vel',

    nodo.create_subscription(LaserScan, 'laser1', lambda msg: laser_ca
    nodo.create_subscription(Odometry, 'base_pose_ground_truth', base_
    nodo.create_subscription(PoseStamped, 'goal_pose', goal_pose_callb

    # Paso 3: Mantener el nodo
    print("Escuchando...")
    rclpy.spin(nodo)

    # Paso 4: Cierre del nodo
    print("Apagando nodo...")
    nodo.destroy_node()
    rclpy.shutdown()

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

OPTIONAL: Improve the technique

The Potential Fields navigation technique admits different improvements upon this baseline version. Implement any of your choice, and include here a discussion about it and some resources illustrating how the robot behaviour improves.

Your answer here!

OPTIONAL: Try other worlds

MVSim comes with different demo world. Try your navigation technique with others and discuss the results.

Your answer here!