

Tutorial

Stack de navegación ROS2

Fundamentos de Robótica Autónoma



MINA - Facultad de Ingeniería - Udelar

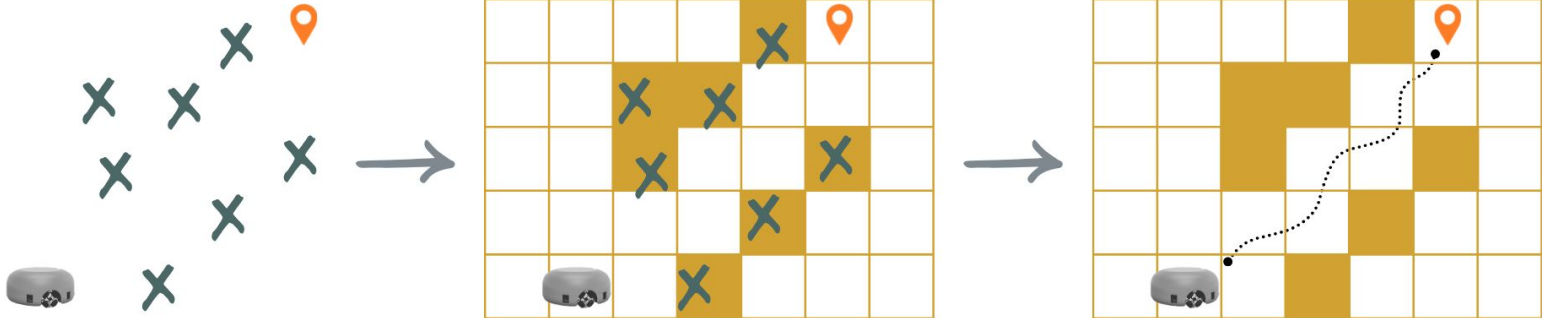
Introducción

- ¿A dónde debo ir?
- ¿Cómo llegar a determinado lugar?
- ¿Dónde he estado?
- ¿Dónde estoy?



ROS2 - Nav2

Dada una pose actual, un mapa y un objetivo, el sistema de navegación genera un plan para alcanzarlo y emite comandos para dirigir al robot de forma autónoma, respetando las restricciones de seguridad y evitando los obstáculos que encuentre en el camino.



ROS2 - Nav2



ROS2 - Nav2

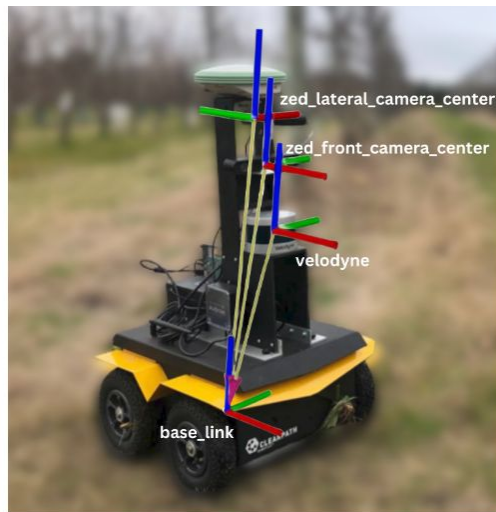


ROS2 - Nav2

TFs (TransForms)

Marcos de
referencia

- Permite mantener una estructura de coordenadas entre distintos marcos de referencia y calcular transformaciones entre ellos en tiempo real.
- <https://www.ros.org/reps/rep-0105.html>

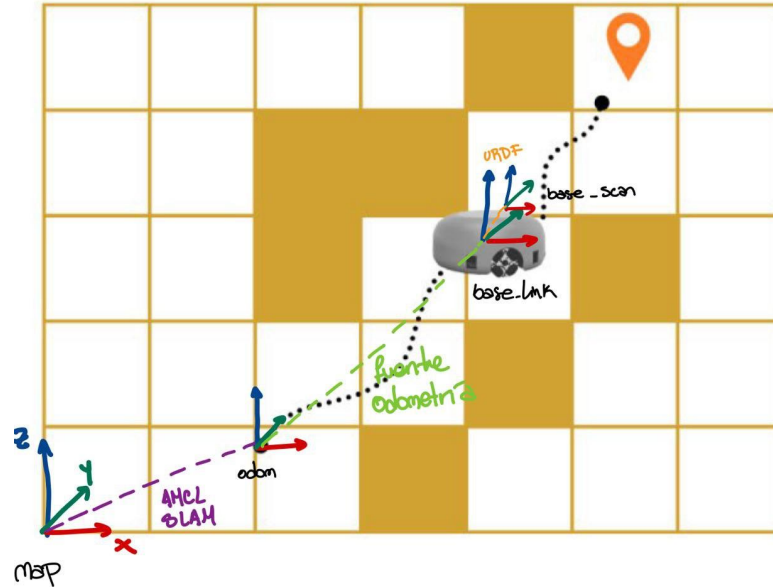


ROS2 - Nav2

TFs

- ¿Qué transformaciones necesita Nav2?

Marcos de
referencia



ROS2 - Nav2

Tutorial:

Marcos de referencia

1. Levantar el mundo simulado

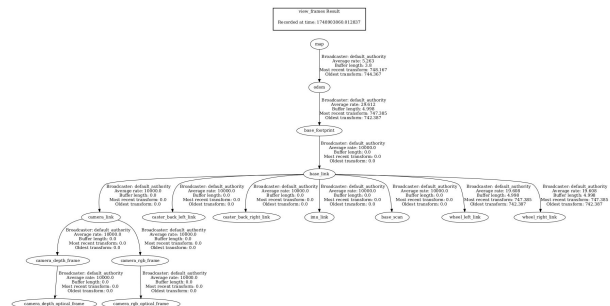
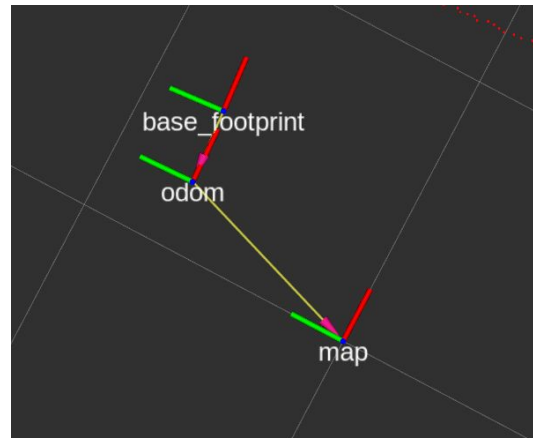
```
$ ros2 launch turtlebot3_gazebo  
turtlebot3_world.launch.py
```

2. Levantar rviz y visualizar tf's:

```
$ rviz2
```

3. Visualizar el árbol de tf's

```
$ ros2 run tf2_tools view_frames
```



Para todos los tutoriales se necesita especificar el modelo de turtlebot a utilizar. En el caso de utilizar el git de fra_nav2 ya está agregado en el .bashrc, si no se debe configurar con el siguiente comando:

```
$ export TURTLEBOT3_MODEL=waaffle
```


ROS2 - Nav2



ROS2 - Nav2

Se publican usualmente como mensajes

Tienen asociado un timestamp

Sensado

Están asociados a un marco de referencia

Ejemplos:

- LaserScan
- Image
- PointCloud2

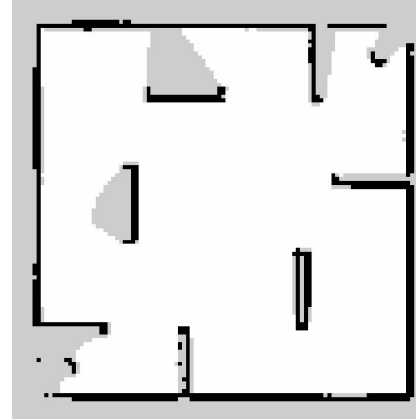
ROS2 - Nav2



ROS2 - Nav2

- Puede recibir un mapa previamente creado (SLAM o a mano)
- Mapa 2D ocupación (OccupancyGrid):
 - Celdas libres
 - Celdas ocupadas
 - Celdas desconocidas

Mapa



ROS2 - Nav2

SLAM:

- Crea mapa de ocupación
- Calcula la pose del robot en el mapa
- Realiza cierre de ciclos: detectar un lugar antes visitado y ajustar el mapa de acuerdo a esta nueva información
- p.e. cartographer, rtabmap, gmapping

Mapa

ROS2 - Nav2

Tutorial:

1. Levantar el mundo simulado

```
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

2. Correr SLAM - Cartographer

```
$ ros2 launch turtlebot3_cartographer cartographer.launch.py  
use_sim_time:=True
```

Mapa

3. Mover el robot:

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

4. Salvar el mapa

```
$ ros2 run nav2_map_server map_saver_cli -f ./mapcambié
```

ROS2 - Nav2

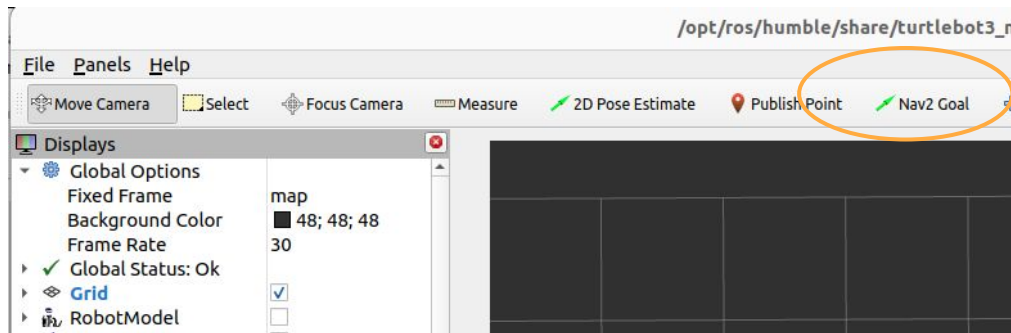


ROS2 - Nav2

- Posición y orientación del mapa a la que el robot debe llegar
- Publicable desde RVIZ o desde código.

```
ros2 topic pub /goal_pose geometry_msgs/PoseStamped "{header: {frame_id: 'map'},  
pose: {position: {x: 2.0, y: 1.0, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.0, w:  
1.0}}}"
```

Objetivo



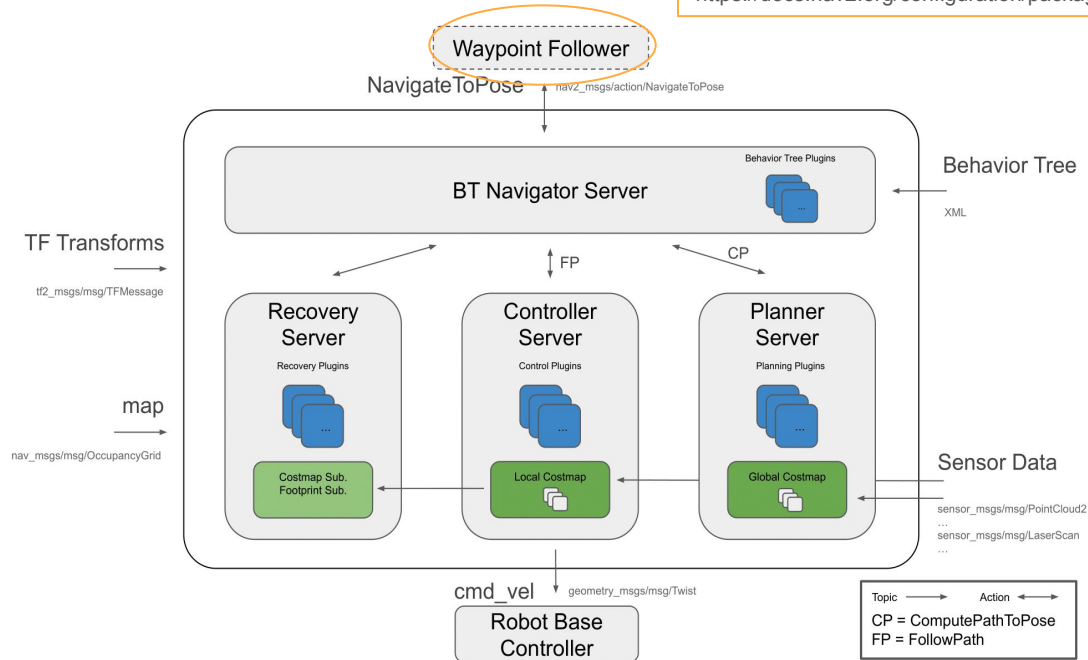
ROS2 - Nav2



ROS2 - Nav2

- Lista de poses
- Envía una por una
- Permite:
 - Esperar entre puntos
 - Hacer tareas específicas como esperar en un punto, tomar una foto o esperar un input
 - Cancelar en caso de error

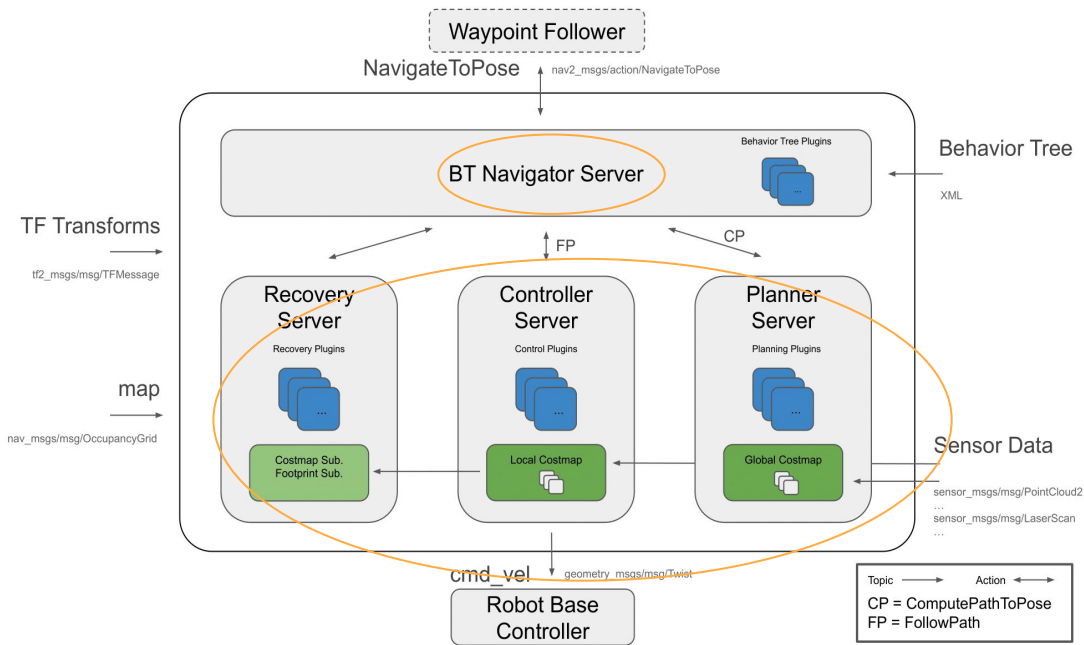
<https://docs.nav2.org/configuration/packages/configuring-waypoint-follower.html>



ROS2 - Nav2

Ejecuta los árboles de comportamiento

<https://docs.nav2.org/configuration/packages/configuring-bt-navigator.html>



Ejecuta comportamientos cuando el BT Navigator server lo indica.

Planner server: generar una trayectoria (global path) desde la posición actual del robot hasta el destino (`goal_pose`), evitando obstáculos conocidos en el `global_costmap`.

Control server: Convierte la trayectoria a comandos de velocidad. Tiene en cuenta los obstáculos del `local_costmap`.

Recovery server: ejecuta los plugins (`nav2_params.yaml`), uno por uno, hasta que alguno tiene éxito o se agota el número de reintentos.

- `ClearCostmap`: Limpia el mapa local o global
- `Spin`: Gira en el lugar para mejorar visibilidad
- `BackUp`: Retrocede en línea recta
- `Wait`: Espera unos segundos

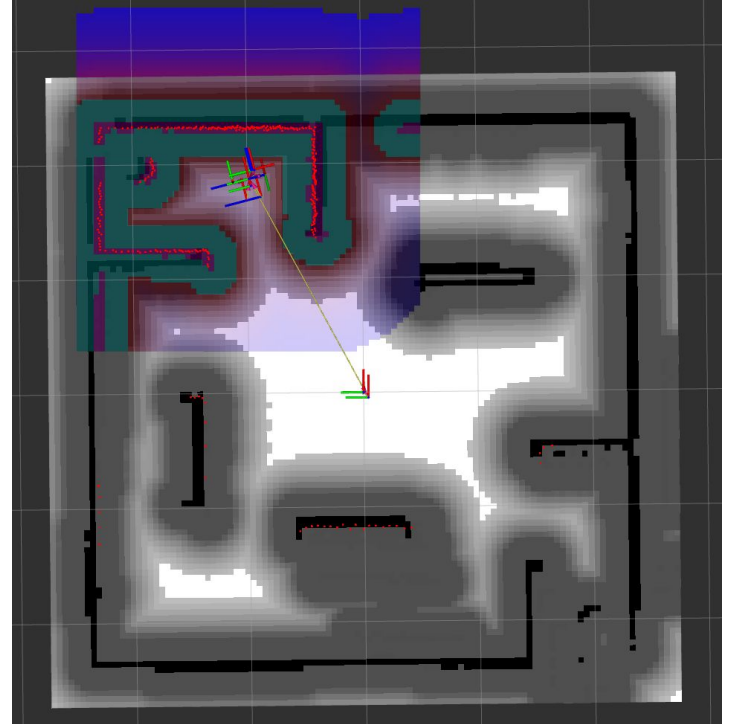
ROS2 - Nav2

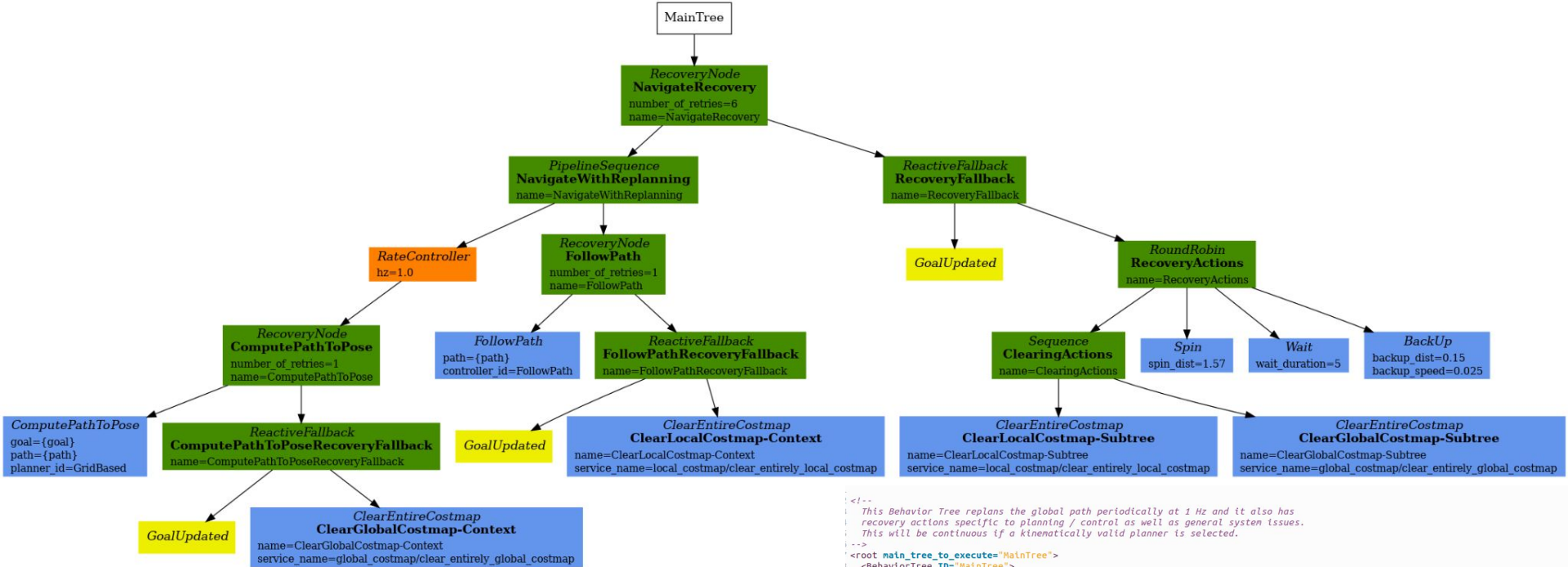
Global costmap:

- Mapa global (frame map)
- Se usa para planificar las rutas desde la pose actual al objetivo
- Estático (map.pgm) aunque puede incluir obstáculos actualizados si se fusionan sensores en tiempo real

Local costmap:

- Mapa más chico, centrado en el robot (frame odom)
- Se mueve con el robot como una "ventana móvil"
- Sirve para evitar obstáculos cercanos y dinámicos
- Lo utiliza el planificador local





```

<!--
This Behavior Tree replans the global path periodically at 1 Hz and it also has
recovery actions specific to planning / control as well as general system issues.
This will be continuous if a kinematically valid planner is selected.
-->
<root main_tree_to_execute="MainTree">
  <BehaviorTree ID="MainTree">
    <RecoveryNode number_of_retries="6" name="NavigateRecovery">
      <PipelineSequence name="NavigateWithReplanning">
        <RateController hz="1.0">
          <RecoveryNode number_of_retries="1" name="ComputePathToPose">
            <ComputePathToPose goal="{goal}" path="{path}" planner_id="GridBased"/>
            <ClearEntireCostmap name="ClearGlobalCostmap-Context" service_name="global_costmap/clear_entirely_global_costmap"/>
          </RecoveryNode>
        </RateController>
        <RecoveryNode number_of_retries="1" name="FollowPath">
          <FollowPath path="{path}" controller_id="FollowPath"/>
          <ClearEntireCostmap name="ClearLocalCostmap-Context" service_name="local_costmap/clear_entirely_local_costmap"/>
        </RecoveryNode>
      </PipelineSequence>
    </RecoveryNode>
    <ReactiveFallback name="RecoveryFallback">
      <GoalUpdated/>
      <RoundRobin name="RecoveryActions">
        <Sequence name="ClearingActions">
          <ClearEntireCostmap name="ClearLocalCostmap-Subtree" service_name="local_costmap/clear_entirely_local_costmap"/>
          <ClearEntireCostmap name="ClearGlobalCostmap-Subtree" service_name="global_costmap/clear_entirely_global_costmap"/>
        </Sequence>
        <Spin spin_dist="1.57"/>
        <Wait wait_duration="5"/>
        <BackUp backup_dist="0.15" backup_speed="0.05"/>
      </RoundRobin>
    </ReactiveFallback>
  </BehaviorTree>
</root>

```

ROS2 - Nav2

Tutorial:

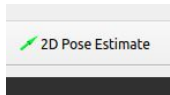
1. Levantar el mundo simulado

```
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

2. Levantar el nodo de navegación:

```
$ ros2 launch turtlebot3_navigation2 navigation2.launch.py use_sim_time:=True  
map:=map.yaml
```

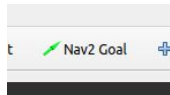
3. Marcar la pose inicial



4. Mover el robot para que se ubique en el mapa

```
$ ros2 run turtlebot3_teleop teleop_keyboard
```

5. Setear una pose objetivo



Nav2 en el robot real

- Verificar que se publiquen las tf's (odom - base_link - laser)
- Verificar nombre del tópicos lidar (/scan)
- Configuraciones de nav2:

Basados en turtlebot3_ws/src/turtlebot3/turtlebot3_navigation2/param/waffle.yaml cambiar los parámetros necesarios (p.e `robot_radius`, `inflation_radius`, `footprint`).

- Cambiar el bringup_launch que levanta el nav2 adaptado a la plataforma local.
- Revisar y ajustar el comando de cmd_vel
- Usar `use_sim_time: false` en robot real
- Se pueden modificar los árboles de comportamiento.

Fin!

Laboratorio 3

SLAM + Stack de navegación

Problema a resolver

Resolver un problema de navegación:

Primero en entorno simulado con un robot comercial

Luego desplegar la solución en el robot del curso

Pasos a seguir:

1. Generar mapas con SLAM en Turtlebot3
2. Navegar en un mapa siguiendo puntos indicados por un usuario
3. Generar marcas en un mapa
4. Desplegar la solución en Francesca