

**UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE**

**ESCUELA POLITÉCNICA SUPERIOR DE ELCHE**

**GRADO EN INGENIERÍA ELECTRÓNICA Y  
AUTOMÁTICA INDUSTRIAL**



**“NAVEGACIÓN MEDIANTE ROS EN UN  
ENTORNO CONTROLADO SIENDO  
CONOCIDO EL MAPA”**

**TRABAJO FIN DE GRADO**

**Junio 2021**

**AUTOR: Francisco José Soler Mora**

**DIRECTOR/ES: Óscar Reinoso García**





*Escuela Politécnica Superior de Elche*

*Junio 2021*



# ÍNDICE GENERAL

<b>1. INTRODUCCIÓN</b> .....	<b>13</b>
1.1. MOTIVACIÓN DEL TRABAJO .....	13
1.2. OBJETIVOS DEL TRABAJO .....	13
1.3. MEMORIA DEL TRABAJO .....	14
<b>2. MATERIALES</b> .....	<b>15</b>
2.1. EQUIPO EMPLEADO .....	15
2.1.1. ¿QUÉ ES UN ROBOT MÓVIL?.....	15
2.1.2. PIONEER 3AT .....	16
2.1.3. LÁSER SICK LMS-200.....	18
2.2. SOFTWARE EMPLEADO .....	19
2.2.1. UBUNTU 16.04.....	19
2.2.2. ROS KINETIC.....	20
2.2.2.1. ESTRUCTURA DE ROS .....	21
2.2.2.2. TRABAJAR CON ROS.....	23
2.2.2.3. CONEXIÓN REMOTA ENTRE EQUIPOS CON ROS .....	35
2.2.2.4. TRASPASO DE ARVHICOS ENTRE EQUIPOS .....	38
2.2.3. ROSARIA.....	39
2.2.4. SICKTOOLBOX_WRAPPER.....	41
2.2.3. GAZEBO .....	42
2.2.4. RVIZ.....	47
<b>3. NAVEGACIÓN</b> .....	<b>50</b>
3.1. CONCEPTOS PREVIOS .....	50
3.1.1. ODOMETRÍA .....	50
3.1.2. TRANSFORMADAS .....	51
3.2. INTORUDCCIÓN .....	52
3.3. TAREAS A REALIZAR.....	54
3.2.1. ÁRBOL DE TRANSFORMADAS.....	55
3.2.2. OBTENCIÓN DEL MAPA .....	56
3.2.3. PUBLICACIÓN DEL MAPA .....	65
3.2.4. LOCALIZACIÓN .....	66
3.2.5. MOVIMIENTO DEL ROBOT.....	69
3.2.5.1. GESTIÓN DE MAPAS DE COSTES.....	70
3.2.5.2. ACCIONES DE NAVEGACIÓN.....	76

3.2.5.3. INTEGRACIÓN DE LAS ANTERIORES .....	84
3.3. NAVEGACIÓN EXTERIORES.....	87
<b>4. SIMULACIÓN .....</b>	<b>88</b>
4.1. IMPLEMENTACIÓN.....	88
4.1.1. CREACIÓN DE UN NUEVO PAQUETE .....	88
4.1.2. CREACIÓN DEL MODELO URDF .....	90
4.1.3. CREACIÓN DEL ENTORNO DE GAZEBO.....	94
4.1.4. PARÁMETROS DEL NAVIGATION STACK .....	95
4.1.5. <i>LAUNCH_FILES</i> .....	95
4.2. USO .....	96
4.3. RESULTADOS.....	99
<b>5. INTEGRACIÓN EN ROBOT REAL .....</b>	<b>103</b>
5.1. IMPLEMENTACIÓN.....	103
5.1.1. CREACIÓN DE UN NUEVO PAQUETE .....	103
5.1.2. PARÁMETROS DEL NAVIGATION STACK .....	104
5.1.3. <i>LAUNCH_FILES</i> .....	105
5.2. USO .....	106
5.3. RESULTADOS.....	107
<b>6. SEGUIDOR DE TRAYECTORIAS.....</b>	<b>108</b>
6.1. NODOS DISPONIBLES.....	109
6.2. USO .....	110
6.2.1. GUARDADO DE TRAYECTORIAS .....	110
6.2.2. REMUESTREO DE POSICIONES .....	113
6.2.4. SEGUIMIENTO DE LA TRAYECTORIA .....	114
6.3. RESULTADOS.....	115
<b>7. CONCLUSIONES Y TRABAJOS FUTUROS.....</b>	<b>116</b>
7.1. CONCLUSIONES .....	116
7.2. TRABAJOS FUTUROS.....	117
<b>8. ANEXOS.....</b>	<b>118</b>
Anexo I: <i>rosaria_params.yaml</i> .....	118
Anexo II: <i>rosaria_p3at.launch</i> .....	119
Anexo III: <i>sick_lms200_params.yaml</i> .....	119
Anexo IV: <i>sick_lms200_p3at.launch</i> .....	119
Anexo V: <i>robot_p3at_lab_world_simulated.launch</i> .....	120
Anexo VI: <i>tf_laser.launch</i> .....	121
Anexo VII: <i>p3at.launch</i> .....	121

Anexo VIII: *map\_server.launch* ..... 122

Anexo IX: *amcl\_p3at\_sicklms200\_params.yaml* ..... 122

Anexo X: *amcl\_p3at.launch* ..... 124

Anexo XI: *costmap\_common\_params.yaml* ..... 124

Anexo XII: *global\_costmap\_params.yaml* ..... 124

Anexo XIII: *local\_costmap\_params.yaml* ..... 125

Anexo XIV: *global\_planner\_params.yaml* ..... 126

Anexo XV: *base\_local\_planner\_params.yaml* ..... 127

Anexo XVI: *recovery\_behaviors\_params.yaml* ..... 128

Anexo XVII: *move\_base\_params.yaml* ..... 129

Anexo XVIII: *move\_base\_p3at.launch* ..... 130

Anexo XIX: *my\_p3at.urdf* ..... 130

Anexo XX: *robot\_p3at\_empty\_world\_simulated.launch* ..... 145

Anexo XXI: *navigation\_p3at.launch* ..... 146

Anexo XXII: *poses\_saver.cpp* ..... 146

Anexo XXIII: *poses\_resampling.cpp* ..... 147

Anexo XXIV: *path\_publisher.cpp* ..... 151

Anexo XXV: *point\_follower.cpp* ..... 152

Anexo XXVI: *hector\_params.yaml* ..... 157

Anexo XXVII: *hector\_mapping.launch* ..... 157

Anexo XXVIII: *test\_point\_follower.launch* ..... 157

Anexo XXIX: *robot\_p3at\_point\_follower\_world\_simulated.launch* ..... 158

**9. BIBLIOGRAFÍA ..... 160**

# ÍNDICE DE FIGURAS

Figura 2.1: Pioneer-3AT. Fuente: [2].....	16
Figura 2.2: Medidas Pioneer-3AT. Fuente: [2].....	16
Figura 2.3: Skid Steer Drive. Fuente: [53].....	17
Figura 2.4: Láser Sick LMS200. Fuente: [3].....	18
Figura 2.5: Funcionamiento Láser SICK LMS200. Fuente: [4].....	18
Figura 2.6: Terminal de Ubuntu.....	19
Figura 2.7: Diagrama básico de comunicación en ROS. Fuente: [10] .....	22
Figura 2.8: Ejemplo de últimas líneas del archivo ".bashrc" .....	24
Figura 2.9: Ejemplo de paquete en ROS.....	25
Figura 2.10: Ejemplo de catkin_create_pkg.....	25
Figura 2.11: Ejemplo de roscore.....	27
Figura 2.12: Ejemplo iniciación nodo RosAria.....	28
Figura 2.13: Ejemplo modificación de tópico en rosrun.....	28
Figura 2.14: Ejemplo modificación de parámetro al iniciar con rosrun. ...	29
Figura 2.15: Ejemplo de GUI generada por rqt_graph. Fuente: [17] .....	30
Figura 2.16: Ejemplo de launch_file.....	34
Figura 2.17: Ejemplo de roslaunch.....	34
Figura 2.18: Conexión por ssh con el propio equipo.....	36



<b>Figura 2.19: Últimas líneas del archivo ".bashrc" para el equipo de control.</b>	<b>37</b>
<b>Figura 2.20: Formato geometry_msgs/Twist</b>	<b>39</b>
<b>Figura 2.21: Pantalla principal de Gazebo</b>	<b>43</b>
<b>Figura 2.22: Ejemplo de la estructura de un archivo URDF</b>	<b>45</b>
<b>Figura 2.23: Ejemplo estructura &lt;joint&gt; URDF</b>	<b>45</b>
<b>Figura 2.24: Gazebo Building Editor</b>	<b>46</b>
<b>Figura 2.25: Pantalla principa del RViz</b>	<b>48</b>
<b>Figura 2.26: Función Add RViz</b>	<b>48</b>
<b>Figura 2.27: Modificar tópico asociado a mensaje en RViz</b>	<b>49</b>
<b>Figura 2.28: Configurar Fixed Frame en RViz</b>	<b>49</b>
<b>Figura 3.1: Ejemplo rqt_tf_tree</b>	<b>52</b>
<b>Figura 3.2: Diagrama de configuración del Navigation Stack. Fuente: [56]</b>	<b>53</b>
<b>Figura 3.3: Ejemplo de uso del nodo static_transform_publisher</b>	<b>55</b>
<b>Figura 3.4: Imagen del archivo tf_laser.launch</b>	<b>56</b>
<b>Figura 3.5: Ejemplo de archivo de imagen de un mapa</b>	<b>59</b>
<b>Figura 3.6: Ejemplo archivo YAML de un mapa</b>	<b>60</b>
<b>Figura 3.7: Ejemplo añadir mapa a RViz</b>	<b>64</b>
<b>Figura 3.8: Ejemplo visualizar tópico /map</b>	<b>64</b>
<b>Figura 3.9: Ejemplo de creación de mapa en simulación</b>	<b>65</b>
<b>Figura 3.10: Ejemplo de partículas de amcl en RViz</b>	<b>69</b>
<b>Figura 3.11: Ejemplo de mapa de costes global</b>	<b>70</b>

<b>Figura 3.12: Gráfico de posibles estados del robot según la posición de su centro. Fuente: [46]</b> .....	<b>71</b>
<b>Figura 3.13: A*. Fuente: [58]</b> .....	<b>78</b>
<b>Figura 3.14: Dijkstra. Fuente: [58]</b> .....	<b>78</b>
<b>Figura 3.15: Planificador por cuadrículas. Fuente: [58]</b> .....	<b>78</b>
<b>Figura 3.16: Comportamiento por defecto. Fuente: [58]</b> .....	<b>78</b>
<b>Figura 3.17: Ejemplo de simulación de trayectorias. Fuente: [59].</b> .....	<b>80</b>
<b>Figura 3.18: Diagrama de flujo recovery_behaviors por defecto. Fuente: [60]</b> .....	<b>83</b>
<b>Figura 4.1: Aspecto del paquete p3at_simulation.</b> .....	<b>89</b>
<b>Figura 4.2: Carpeta models del paquete p3at_simulation.</b> .....	<b>89</b>
<b>Figura 4.3:Primera parte archivo my_p3at.urdf.</b> .....	<b>90</b>
<b>Figura 4.4: Primera parte del archivo my_p3at.urdf</b> .....	<b>91</b>
<b>Figura 4.5: Segunda parte archivo my_p3at.urdf</b> .....	<b>91</b>
<b>Figura 4.6: Tercera parte del archivo my_p3at.urdf</b> .....	<b>91</b>
<b>Figura 4.7: Cuarta parte del archivo my_p3at.urdf</b> .....	<b>92</b>
<b>Figura 4.8: Quinta parte del archivo my_p3at.urdf</b> .....	<b>92</b>
<b>Figura 4.9: Sexta parte del archivo my_p3at.urdf</b> .....	<b>93</b>
<b>Figura 4.10: Séptima parte del archivo my_p3at.urdf</b> .....	<b>93</b>
<b>Figura 4.11: Aspecto del entorno descrito por el archivo real_lab.world.</b>	<b>94</b>
<b>Figura 4.12: Archivos de parámetros disponibles en el paquete p3at_simulation.</b> .....	<b>95</b>
<b>Figura 4.13: Contenido carpeta /launch del paquete p3at_simulation.</b> .....	<b>96</b>
<b>Figura 4.14: Inicio de simulación en Gazebo.</b> .....	<b>96</b>

<b>Figura 4.15: Ejemplo de uso del archivo navigation_p3at.launch. ....</b>	<b>97</b>
<b>Figura 4.16: RViz para navegación simulada.....</b>	<b>97</b>
<b>Figura 4.17: 2D POSE ESTIMATE RViz.....</b>	<b>98</b>
<b>Figura 4.18: 2D NAV GOAL RViz.....</b>	<b>99</b>
<b>Figura 5.1: Contenido del paquete p3at_real.....</b>	<b>104</b>
<b>Figura 5.2: Archivos de parámetros paquete p3at_real.....</b>	<b>104</b>
<b>Figura 5.3: Launch_files del paquete p3at_real.....</b>	<b>105</b>
<b>Figura 6.1: Líneas añadidas a CMakeLists de paquete p3at_simulation. ....</b>	<b>109</b>
<b>Figura 6.2: Nodo disponibles para seguidor de trayectorias. ....</b>	<b>110</b>
<b>Figura 6.3: Ejemplo de uso del archivo robot_p3at_point_follower_world_simulated.launch.....</b>	<b>111</b>
<b>Figura 6.4: Ejemplo de uso del archivo hector_mapping.launch.....</b>	<b>111</b>
<b>Figura 6.5: Ejemplo de captura de poses con el nodo poses_saver. ....</b>	<b>112</b>
<b>Figura 6.6: Ejemplo archivo full_poses.json.....</b>	<b>112</b>
<b>Figura 6.7: Ejemplo archivo resampled_poses.json.....</b>	<b>113</b>
<b>Figura 6.8: Ejemplo de uso del nodo poses_resampling.....</b>	<b>113</b>
<b>Figura 6.9: Ejemplo del archivo resampled_poses.json.....</b>	<b>114</b>
<b>Figura 6.10: Ejemplo de uso del nodo path_publisher. ....</b>	<b>114</b>
<b>Figura 6.11: Ejemplo de uso del archivo test_point_follower.launch.....</b>	<b>115</b>



# 1. INTRODUCCIÓN

La navegación de robots móviles puede ser una tarea ardua y compleja. En este Trabajo Fin de Grado se utilizan las facilidades que ofrece Robot Operating System (ROS) para realizar dicha navegación, utilizando los robots móviles disponibles en el laboratorio de Automática, Robótica y Visión por computador (ARVC) de la Universidad Miguel Hernández de Elche (UMH).

## 1.1. MOTIVACIÓN DEL TRABAJO

Este trabajo, se propone con la idea de ofrecer al personal investigador de la UMH, una herramienta que puedan usar para futuras investigaciones o proyectos con robots móviles, partiendo de una herramienta que les permita comandar el robot a través de un mapa de ocupación.

De esta manera se permite que el personal investigador se centre en el desarrollo de su proyecto agilizando así el progreso de este.

## 1.2. OBJETIVOS DEL TRABAJO

El objetivo del presente TFG es desarrollar la navegación en interiores de un robot móvil, partiendo de un mapa de ocupación obtenido mediante un sensor láser. Esta herramienta permitirá a cualquier usuario la capacidad de indicar puntos de destino a cierto robot móvil, y que éste se desplace a dichos destinos de forma autónoma, es decir, decidiendo el camino a seguir y evitando los obstáculos intermedios.

Dichos puntos, estarán dentro de un mapa de ocupación y se podrán proporcionar de forma gráfica. Asimismo, de manera adicional, se busca que el usuario sea capaz de almacenar varios puntos de destino para posteriormente proporcionárselos al robot y que éste siga la trayectoria descrita por dichos puntos.

### 1.3. MEMORIA DEL TRABAJO

En la memoria de este trabajo se redacta el procedimiento que se ha seguido para el desarrollo del sistema de navegación y todas las herramientas empleadas para su desarrollo, así como su uso.

En el primer capítulo, se redacta una pequeña introducción al trabajo, que se va a realizar y los objetivos de este.

En el segundo capítulo, se encuentra detallado el material de partida, que se ha empleado para desarrollar el objetivo de este trabajo.

El tercer capítulo se centra en la navegación. Se explica que es la navegación, como obtener un mapa de ocupación a través de un sensor láser, paquetes que se han empleado para el desarrollo de esta, y opciones para el desarrollo de una navegación en exteriores.

Cuarto y quinto capítulo son similares. Ambos muestran el procedimiento que se ha seguido para el desarrollo de la navegación, tanto en simulación, como en la realidad, y las configuraciones necesarias para el correcto funcionamiento de cada una de ellas, así como los resultados obtenidos en cada caso.

El sexto capítulo, plantea el desarrollo de una aplicación capaz de almacenar y reproducir trayectorias realizadas por el robot.

En el séptimo capítulo, se redactan las conclusiones a las que se ha llegado tras la realización del trabajo, y se proponen ideas para trabajos futuros y posibles mejoras.

En el capítulo de anexos se encuentran todos los archivos de código empleados, debidamente comentados.

Por último, está redactada según norma IEEE, la bibliografía empleada.

## 2. MATERIALES

En esta sección se detallan tanto el equipo, como el software empleado. Además, se realiza una pequeña introducción a la robótica móvil, para maximizar la comprensión del trabajo.

### 2.1. EQUIPO EMPLEADO

Como robot móvil, se ha utilizado uno de los diversos Pioneer 3AT disponibles en el laboratorio ARVC, el cual, incorpora un láser Sick LMS-200. Ambos se describen detalladamente a continuación.

#### 2.1.1. ¿QUÉ ES UN ROBOT MÓVIL?

¿Qué es un robot móvil? Cualquier usuario que no esté familiarizado con la robótica se hará esta pregunta. Por ello, dedicamos este apartado a resolverla.

Hasta hace relativamente poco tiempo la idea de robot se asociaba a la imagen de robot humanoide, es decir, un conjunto de elementos electromecánicos que se asemeja a la forma humana, y es capaz de realizar acciones y movimientos propios del ser humano. Gracias al desarrollo de la tecnología y su mayor implicación en la vida de las personas, esta idea está cambiando.

El hecho de que, a día de hoy, en nuestras casas dispongamos de robots de cocina, robots aspiradores, amplía el campo de visión a la hora de definir que es un robot. En concreto en este apartado vamos a definir que es un robot móvil, ya que el concepto de robot es muy amplio y no es objetivo de este trabajo definirlo.

Un robot móvil se podría definir como un conjunto de sistemas electromecánicos programables, capaces de moverse en un determinado entorno para realizar ciertas acciones programadas. Por otro lado, la definición de un robot móvil se puede concretar más, dependiendo del entorno en el que se realice el movimiento. Los tres tipos principales de robots móviles, según su

entorno, serían: terrestres, aéreos y acuáticos. Sus principales diferencias radican en los mecanismos que utilizan para realizar desplazamientos en su entorno. Para más información sobre la descripción de la robótica móvil [1].

El robot móvil empleado para este trabajo, Pioneer 3AT, se trata de un robot móvil terrestre.

### 2.1.2. PIONEER 3AT

El Pioneer 3AT, es un robot móvil terrestre, fabricado por Adept MobileRobots, para tareas de investigación y docencia. Como sistema de locomoción consta de 4 ruedas cuya configuración y transmisión del movimiento hace que se trate de un sistema “*skid-steer*”. Dicho sistema de locomoción se comenta en mayor profundidad en las páginas siguientes de este mismo capítulo. Si se desea más información sobre el Pioneer 3-AT se puede acceder a la hoja de características del fabricante [2].



Figura 2.1: Pioneer-3AT. Fuente: [2]

Para ubicar más al lector, se muestran a continuación las medidas del Pioneer-3AT.

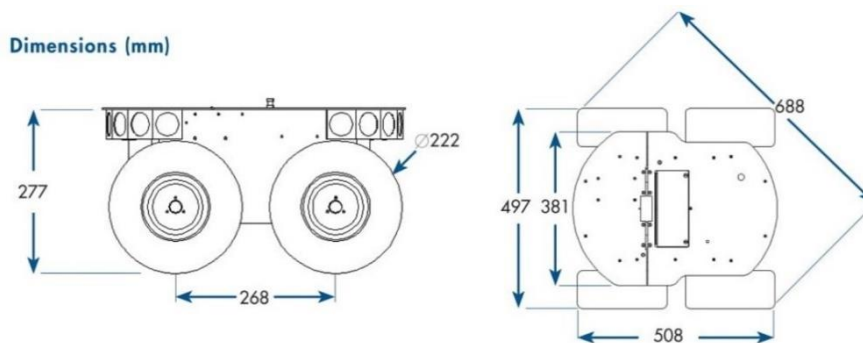


Figura 2.2: Medidas Pioneer-3AT. Fuente: [2]



Para una mejor comprensión de este TFG, se detalla el funcionamiento y las limitaciones del sistema de locomoción “*skid-steer*”, dado que es conveniente conocer dichas limitaciones a la hora de usar la navegación.

El sistema “*skid-steer*” se caracteriza por la transmisión del movimiento a las ruedas. Esta transmisión, está formada por 2 grupos de ruedas, cada grupo a un lado del robot. La transmisión de velocidad a cada grupo de ruedas está dispuesta de tal manera que todas las ruedas de un grupo, giran en el mismo sentido y la misma velocidad. Esto permite 3 configuraciones para el movimiento:

- **Mismo sentido y velocidad:** ambos grupos de ruedas giran en mismo sentido y velocidad, consiguiendo así un movimiento rectilíneo. Hacia adelante o hacia atrás dependiendo del sentido de giro.
- **Mismo sentido y distinta velocidad:** ambos grupos giran en el mismo sentido, pero con velocidades distintas. Esta diferencia de velocidad entre ambos lados del robot provoca un giro de cierto ángulo que viene definido por la diferencia de las velocidades.
- **Distinto sentido y misma velocidad:** cada grupo gira en un sentido, pero con la misma velocidad, provocando un giro sobre sí mismo al robot.

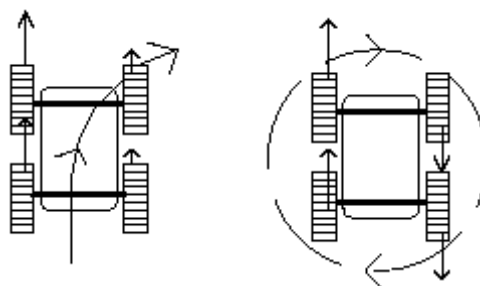


Figura 2.3: Skid Steer Drive. Fuente: [53]

Este sistema de locomoción se apoya en la suposición de que las ruedas del robot son capaces de deslizarse sobre la superficie en que se encuentra. Giros muy cerrados, provocan un mayor deslizamiento de las ruedas del robot, dicho deslizamiento no es tenido en cuenta por la odometría (estimación de la posición

del robot a corto plazo), esto provoca una peor localización, que a su vez, se traduce en una mala navegación.

### 2.1.3. LÁSER SICK LMS-200

Como herramienta principal para la navegación, concretamente la localización, detección de objetos y creación del mapa, el Pioneer-3AT lleva incorporado sobre su chasis un escáner láser 2D Sick LMS-200. Para acceder a sus características básicas [3] y para mayor profundidad [4].



Se trata de un dispositivo de medición sin contacto pensado para uso interior con una distancia máxima de detección de 80m y una apertura de 180°. Su principio de funcionamiento consiste en la medición del tiempo entre la emisión y recepción de rayos infrarrojos de clase 1 (no perjudicial para la vista (EN/IEC 60825-1, 21CFR 1040.10) en un plano radial paralelo a la superficie en la que se encuentre ubicado. Este tiempo es directamente proporcional a la distancia entre el dispositivo y el objeto detectado.

Dispone de 3 resoluciones posibles (0.25° | 0.5° | 1°), en nuestro caso usaremos la resolución de 1°.

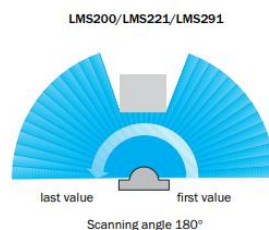


Figura 2.5: Funcionamiento Láser SICK LMS200. Fuente: [4]

## 2.2. SOFTWARE EMPLEADO

En este apartado, comentaremos el software utilizado para la realización de este trabajo, así como sus principales funciones.

### 2.2.1. UBUNTU 16.04

Ubuntu, es un sistema operativo de código abierto basado en Debian, una distribución de Linux. Su facilidad de uso, acceso a un gran número de software libre, y su compatibilidad con sistemas como ROS, OpenCV, PX4, hacen de Ubuntu una de las mejores opciones para desarrollo e investigación en robótica [5].

Para este trabajo se ha optado por el sistema operativo Ubuntu, en su versión 16.04.7 LTS (Xenial Xerus) [6], dado que ROS ofrece soporte total para este sistema.

A continuación, vamos a indicar la metodología de trabajo y citar los principales comandos de Ubuntu para poder emplear la navegación desarrollada [7]. La metodología de trabajo que se usa con Ubuntu es la siguiente:

Todas las operaciones se realizan a través de terminales. Para abrir un nuevo terminal hacemos clic derecho en el escritorio y seleccionamos la opción abrir terminal, y nos abrirá un terminal de Ubuntu.

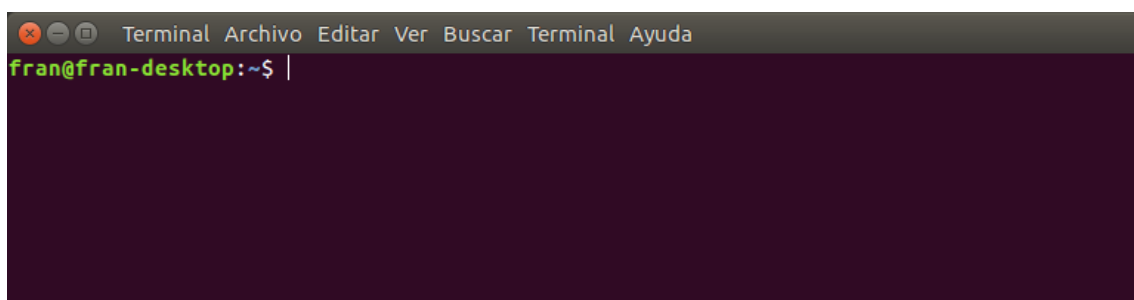


Figura 2.6: Terminal de Ubuntu.

A través de estos terminales, podemos indicarle acciones al sistema mediante una serie de comandos. A continuación, se citan los más importantes:

- **mkdir**: comando utilizado para creación de nuevos directorios.
- **cd**: comando utilizado para moverse entre directorios.

- **ls**: comando para listar los archivos dentro de la ubicación actual.
- **sudo**: comando para dar permisos de super usuario.
- **apt-get install**: comando para instalar paquetes.
- **apt-get update**: actualiza la lista de paquetes disponibles.
- **apt-get upgrade**: actualiza los paquetes instalados.
- **rm**: comando para borrar archivos.
- **echo**: comando para mostrar datos en el terminal.
- **ssh**: comando para realizar una conexión inalámbrica.
- **tar -czfv**: comando y opción para comprimir ficheros.
- **tar -xzvf**: comando y opción para descomprimir ficheros.
- **mv**: mueve ficheros al directorio deseado.
- **scp**: permite enviar archivos comprimidos de un equipo a otro.

Tras todos estos comandos se puede escribir la opción --help. Esta opción nos muestra el modo de empleo de dicho comando y las posibles opciones de las que dispone. Para más información sobre los comandos disponibles [8].

### 2.2.2. ROS KINETIC

En la introducción al trabajo, ya mencionamos ROS, Robot Operating System. Pero, ¿qué es ROS y porque usarlo?

ROS es un sistema operativo de código abierto orientado totalmente al uso en robótica. Tiene como objetivo la reutilización de código en investigación y desarrollo. Esto permite mayor abstracción de los códigos y un desarrollo e implementación más temprana.

Su estructura está distribuida en procesos, llamados **Nodos**, que trabajan de forma individual. Dichos nodos se encargan de publicar **Mensajes** en **Tópicos** y/o suscribirse a estos. Los mensajes tienen un tipo de dato asociado. El tipo de dato es independiente del lenguaje de programación que se use, es decir, tanto C++ como Python (los dos principales), pueden publicar los mismos tipos de datos. Esto se traduce en que sin importar el tipo de lenguaje en el que se haya desarrollado el nodo, los mensajes en los tópicos serán los mismos. Permitiendo que un nodo escrito en Python, sea capaz de suscribirse a un tópico publicado por un nodo escrito en C++ y viceversa.

Esto otorga una gran flexibilidad a la hora de usar ROS, puesto que el lenguaje de programación empleado para los nodos es indiferente. Los nodos se agrupan en **Paquetes** para facilitar su distribución y reutilización.

ROS consta de varias distribuciones, en este trabajo se usa la distribución Kinetic Kame, por tener mucho soporte y ser adecuada en combinación con Ubuntu 16.04.

Para cualquier duda sobre ROS de aquí en adelante se recomienda visitar su página oficial [9].

### 2.2.2.1. ESTRUCTURA DE ROS

Cómo nos indican en la página web oficial de ROS [10], éste, está dividido en 3 conceptos:

- **ROS Filesystem Level:** en este concepto se encuentra la estructura que han de tener las carpetas y los archivos mínimos necesarios para funcionar.
- **ROS Computation Graph Level:** aquí es donde se crea una red que permite que todos los procesos de ROS estén conectados. Cualquier nodo puede enviar o recibir datos de esta red.
- **ROS Community Level:** comprende las herramientas para compartir paquetes, documentación de estos, y dudas entre los desarrolladores.

Veamos ahora en mayor profundidad el *Filesystem Level* y *Computation Graph Level*.

#### **ROS Filesystem Level**

ROS, como su nombre indica, *Robot Operating System*, actúa como un sistema operativo. Los programas se organizan en carpetas, siempre con la misma estructura, y dentro de ellas están los archivos necesarios para el funcionamiento de dicho programa.

Dentro de este nivel se encuentran:

- **Paquetes:** carpeta que contiene la unidad mínima para realizar un programa en ROS.
- **Metapaquetes:** agrupación de paquetes.

- Tipos de mensajes: carpeta que contiene archivos `.msg` que definen la estructura de los mensajes definidos por el usuario.
- Tipos de servicios: carpeta que contiene archivos `.srv` que definen la estructura de los servicios definidos por el usuario.

### ROS Computation Graph Level:

Dentro del Computation Graph Level encontramos los siguientes conceptos:

- Nodo: proceso a ejecutar.
- Maestro: permite la comunicación entre nodos e inicia el Parameter Server.
- Parameter Server: permite almacenar parámetros que son tomados por los nodos durante su ejecución.
- Mensajes: estructura que contiene varios tipos simples de datos.
- Servicios: permite a un nodo, solicitar información de otro nodo.
- Tópicos: elemento donde se publica un determinado mensaje.
- Bags: herramienta que permite guardar y reproducir los datos generados durante un experimento en ROS.

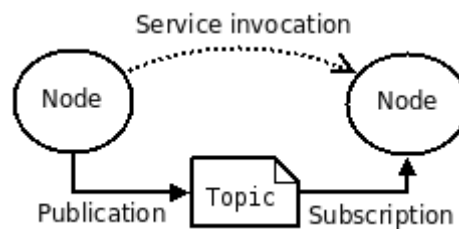


Figura 2.7: Diagrama básico de comunicación en ROS. Fuente: [10]

### 2.2.2.2. TRABAJAR CON ROS

En este apartado se explica cómo instalar ROS, y las herramientas que se han usado para desarrollar este trabajo.

#### Instalación

Se han seguido las instrucciones de instalación proporcionadas por la Wiki de ROS para la instalación de ROS Kinetic en Ubuntu [11].

#### Catkin workspace

ROS emplea como sistema de construcción *catkin*. Esto se traduce en que como directorio principal tendremos un *catkin\_workspace* [12]. Se trata de una carpeta que contendrá todos los paquetes que descarguemos directamente desde la fuente. En caso de descargar un paquete desde los repositorios, este se guardará en la ruta donde este instalado ROS (*/opt/ros/kinetic*).

El *catkin\_workspace* se identifica por ser una carpeta denominada */catkin\_ws* y que contiene:

- **/build:** carpeta usada por CMake para compilar los paquetes.
- **/devel:** contiene los programas compilados antes de ser instalados.
- **/src:** contiene el código fuente de los paquetes.

#### Crear un nuevo *catkin workspace*:

- Abrir terminal de Ubuntu.
- **\$ mkdir -p ~/catkin\_ws/src**
- **\$ cd ~/catkin\_ws/**
- **\$ catkin\_make**
- **\$ source devel/setup.bash**

En caso de querer trabajar siempre con el mismo *catkin\_workspace*, se recomienda editar el archivo “*.bashrc*” dentro de la Carpeta Personal de Ubuntu, y colocar al final la siguiente línea:

```
source ~/catkin_ws/src/devel/setup.bash
```

La adición de esta línea al final del archivo “.bashrc”, provoca que cada vez que se abra un nuevo terminal de Ubuntu, se ejecute de forma automática dicha línea. El comando *source*, permite al sistema tener acceso a nuestros paquetes de ROS.

```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

## CARPETAS DONDE ESTARÁN NUESTROS ELEMENTOS DE ROS ##

source /opt/ros/kinetic/setup.bash
source ~/programas_ros/catkin_ws/devel/setup.bash
```

Figura 2.8: Ejemplo de últimas líneas del archivo “.bashrc”.

## Paquetes

Son la unidad más elemental de ROS. Contiene los archivos y carpetas mínimas para crear un programa. Dichas carpetas y archivos son:

- **/bin**: programas compilados.
- **/include/nombre\_paquete**: librerías empleadas.
- **/msg**: mensajes no estándar.
- **/srv**: servicios no estándar.
- **/src**: código fuente de los programas.
- **/launch**: archivos “.launch” de ROS. Permiten iniciar varios nodos a la vez.
- **CMakeLists.txt**: archivo de instrucciones para el compilador.
- **package.xml**: archivo de especificaciones del paquete.



Las carpetas `/msg`, `/srv`, `/launch` no son creadas predeterminadamente por el comando de ROS `“catkin_make_pkg”`, el cual se detalla más adelante. En caso de necesitar mensajes o servicios propios, o `launch_files`, habrá que crear las carpetas manualmente con el nombre correspondiente.

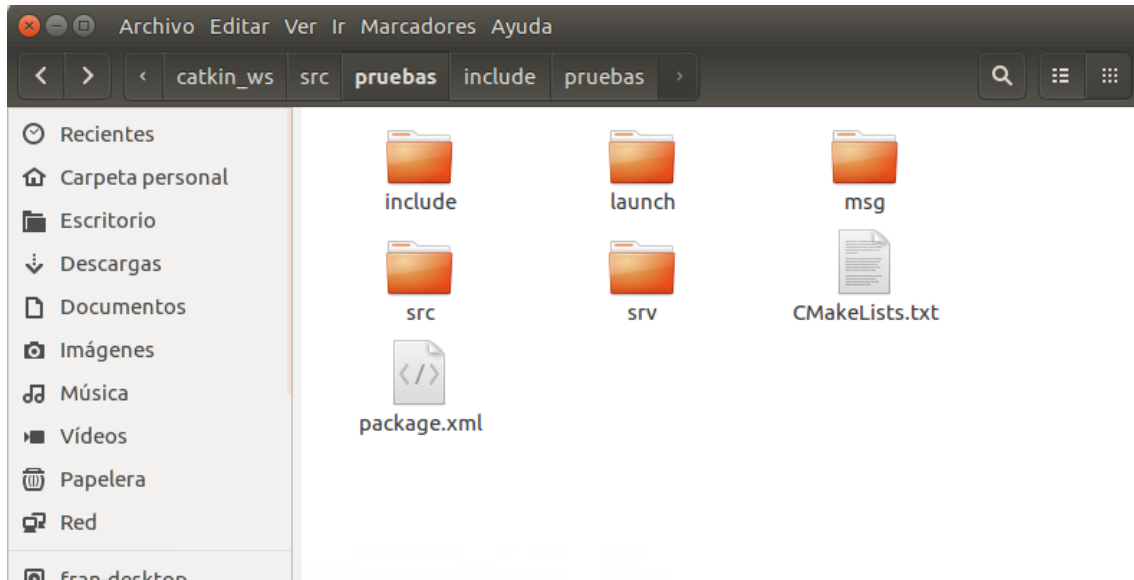


Figura 2.9: Ejemplo de paquete en ROS.

### Crear un nuevo paquete en ROS [13]:

- Abrir terminal de Ubuntu.
- `$ cd ~/catkin_ws/src`
- `$ catkin_create_pkg <package_name> [depend1] [depend2]`
- `$ cd ~/catkin_ws`
- `$ catkin_make`

```

Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ cd programas_ros/catkin_ws/src/
fran@fran-desktop:~/programas_ros/catkin_ws/src$ catkin_create_pkg tests_pkg roscpp
rospy geometry_msgs
Created file tests_pkg/package.xml
Created file tests_pkg/CMakeLists.txt
Created folder tests_pkg/include/tests_pkg
Created folder tests_pkg/src
Successfully created files in /home/fran/programas_ros/catkin_ws/src/tests_pkg. Plea
se adjust the values in package.xml.
fran@fran-desktop:~/programas_ros/catkin_ws/src$ |

```

Figura 2.10: Ejemplo de `catkin_create_pkg`.

### Descargar paquetes en ROS:

- Repositorios: se puede hacer desde cualquier directorio.

```
$ sudo apt-get install ros-"distribución_de_ros"- "paquete_deseado"
```

- GitHub: hay que estar en el directorio `/catkin_ws/src`.

```
$ sudo git clone "URL del paquete en GitHub"
```

### Compilar paquetes en ROS:

Desde el directorio `/catkin_ws`:

```
$ catkin_make
```

### Comandos para paquetes en ROS:

- `rospack`: encuentra y devuelve información sobre paquetes.
- `roscd`: permite moverse a la ubicación de paquetes de ROS.
- `catkin_create_pkg`: crea un nuevo paquete.
- `catkin_make`: compila el `workspace` donde se encuentre.

### Maestro

El maestro, también llamado **rosmaster**, es el componente de ROS que permite la comunicación de unos nodos con otros a través de sus tópicos, mensajes o servicios. Además, inicia el servidor de parámetros, el cual almacena valores de configuración necesarios para el funcionamiento de los nodos.

Para trabajar con ROS, el primer paso siempre ha de ser iniciar el **rosmaster**. Si el **rosmaster** no está activo, e intentamos iniciar un nodo en ROS, nos saltará un mensaje de error.

El arranque del **rosmaster** se hace mediante el comando **roscore**. Su uso es muy sencillo como se detalla a continuación.

- Abrir nuevo terminal de Ubuntu.
- **\$ roscore**

Este comando se puede ejecutar desde cualquier directorio.

```
Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ roscore
... logging to /home/fran/.ros/log/9b95894a-b152-11eb-92c3-bc307d93fccf/roslaunch-fr
an-desktop-4060.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://fran-desktop:34615/
ros_comm version 1.12.17

SUMMARY
=====

PARAMETERS
* /rostdistro: kinetic
* /rosversion: 1.12.17

NODES

auto-starting new master
process[rosmaster]: started with pid [4083]
ROS_MASTER_URI=http://fran-desktop:11311/

setting /run_id to 9b95894a-b152-11eb-92c3-bc307d93fccf
process[rosout-1]: started with pid [4096]
started core service [/rosout]
```

Figura 2.11: Ejemplo de roscore.

## Nodos

Contienen los distintos procesos a realizar en ROS. ROS está diseñado para que cada uno de estos, se encargue de realizar una pequeña acción y comunicar sus resultados mediante tópicos, servicios o a través del servidor de parámetros. Estas acciones, pueden ser la lectura de datos de una cámara usb, lectura de datos de un sensor láser, calcular la ubicación actual del robot, etc.

Gracias a esto, se puede desarrollar un sistema complejo, abordando pequeños procesos por separado.

Esto permite localizar posibles fallos por el sistema en nodos específicos, aislando cada uno de ellos y solucionando dichos fallos de forma individual.

A continuación, se nombran las principales herramientas de ROS para trabajar con nodos, pero no para desarrollarlos. En caso de querer profundizar en la creación de nodos desde cero, se recomienda seguir los tutoriales que se encuentran en la Wiki oficial de ROS [14].

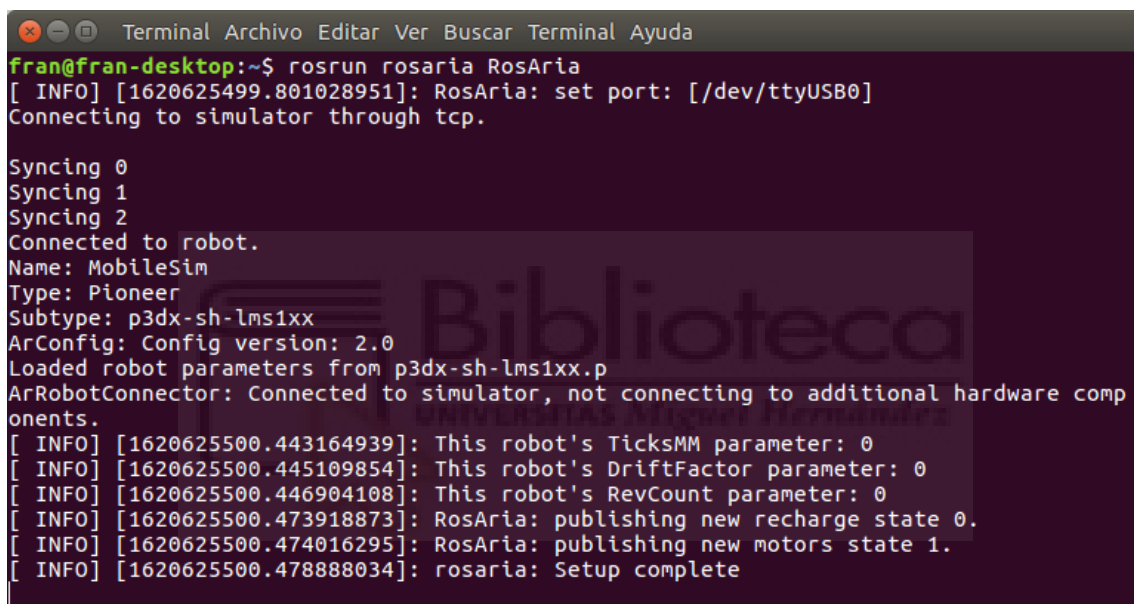
### Comandos para nodos en ROS:

En este apartado se van a nombrar tres herramientas de gran utilidad a la hora de desarrollar cualquier programa en ROS.

En primer lugar, tenemos el comando **roslaunch** [15]. Este comando permite ejecutar nodos individualmente. Además, ofrece la posibilidad de indicar parámetros o cambiar el nombre del tópico al que dicho nodo se debe suscribir o al que debe publicar. A continuación se detalla su uso:

- **\$ roslaunch <nombre\_paquete> <nombre\_nodo>**

Ejecuta un nodo, indicando este, y el paquete al que pertenece.



```

Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ roslaunch rosaria RosAria
[ INFO] [1620625499.801028951]: RosAria: set port: [/dev/ttyUSB0]
Connecting to simulator through tcp.

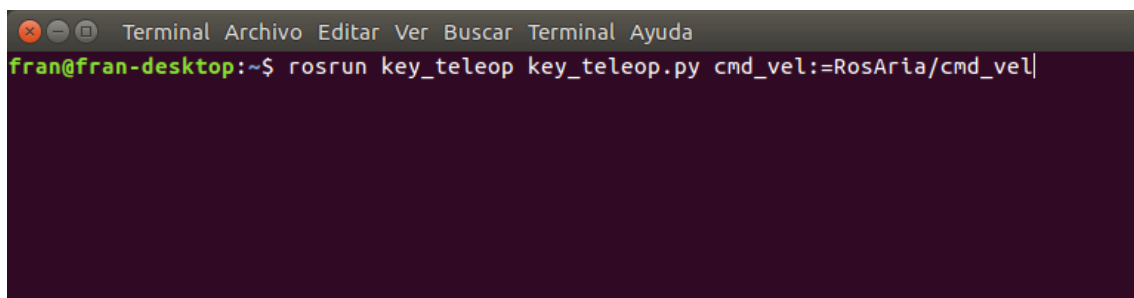
Syncing 0
Syncing 1
Syncing 2
Connected to robot.
Name: MobileSim
Type: Pioneer
Subtype: p3dx-sh-lms1xx
ArConfig: Config version: 2.0
Loaded robot parameters from p3dx-sh-lms1xx.p
ArRobotConnector: Connected to simulator, not connecting to additional hardware components.
[ INFO] [1620625500.443164939]: This robot's TicksMM parameter: 0
[ INFO] [1620625500.445109854]: This robot's DriftFactor parameter: 0
[ INFO] [1620625500.446904108]: This robot's RevCount parameter: 0
[ INFO] [1620625500.473918873]: RosAria: publishing new recharge state 0.
[ INFO] [1620625500.474016295]: RosAria: publishing new motors state 1.
[ INFO] [1620625500.478888034]: rosaria: Setup complete

```

Figura 2.12: Ejemplo iniciación nodo RosAria.

- **\$ roslaunch <nombre\_paquete> <nombre\_nodo> <tópico\_actual> := <tópico\_deseado>**

Permite modificar el tópico al que el nodo publica o se suscribe.



```

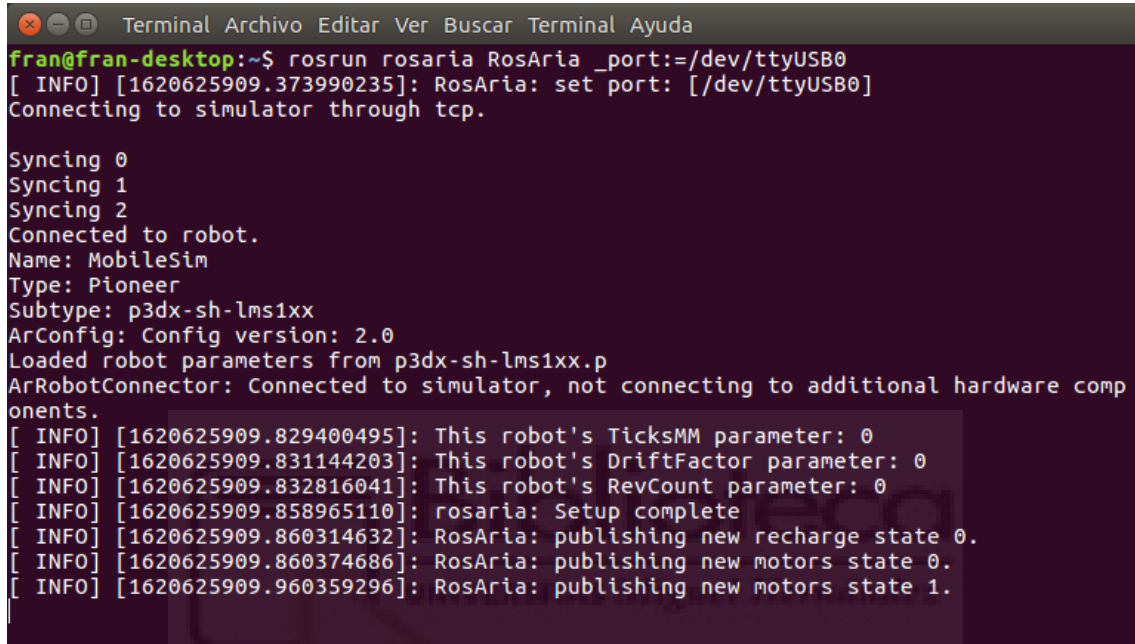
Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ roslaunch key_teleop key_teleop.py cmd_vel:=RosAria/cmd_vel

```

Figura 2.13: Ejemplo modificación de tópico en roslaunch.

- **\$ rosrun <nombre\_paquete> <nombre\_nodo> \_<parámetro>:= <valor>**

Permite iniciar el nodo con un valor determinado en cierto parámetro. Prestar especial atención a la barra baja previa al nombre del parámetro.



```

Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ rosrun rosaria RosAria _port:=/dev/ttyUSB0
[ INFO] [1620625909.373990235]: RosAria: set port: [/dev/ttyUSB0]
Connecting to simulator through tcp.

Syncing 0
Syncing 1
Syncing 2
Connected to robot.
Name: MobileSim
Type: Pioneer
Subtype: p3dx-sh-lms1xx
ArConfig: Config version: 2.0
Loaded robot parameters from p3dx-sh-lms1xx.p
ArRobotConnector: Connected to simulator, not connecting to additional hardware components.
[ INFO] [1620625909.829400495]: This robot's TicksMM parameter: 0
[ INFO] [1620625909.831144203]: This robot's DriftFactor parameter: 0
[ INFO] [1620625909.832816041]: This robot's RevCount parameter: 0
[ INFO] [1620625909.858965110]: rosaria: Setup complete
[ INFO] [1620625909.860314632]: RosAria: publishing new recharge state 0.
[ INFO] [1620625909.860374686]: RosAria: publishing new motors state 0.
[ INFO] [1620625909.960359296]: RosAria: publishing new motors state 1.

```

Figura 2.14: Ejemplo modificación de parámetro al iniciar con rosrun.

La segunda herramienta, es el comando de ROS **rostopic** [16], este comando dispone de las siguientes opciones:

- **rostopic info:** muestra información del nodo indicado, así como a qué tópicos está suscrito y en que tópicos publica.
- **rostopic kill:** termina el proceso del nodo indicado.
- **rostopic list:** muestra todos los nodos que se están ejecutando.
- **rostopic machine:** muestra todos los nodos activos en un determinado equipo.
- **rostopic ping:** comprueba la conexión de un nodo.

- **roscleanup**: elimina los registros de aquellos nodos con los que no puede contactar. Útil cuando hemos terminado el proceso de un nodo, pero ROS sigue manteniendo sus tópicos o parámetros.

Por último, otra herramienta que se considera importante a la hora de trabajar con nodos en ROS es **rqt\_graph** [17].

Esta herramienta, es un paquete de ROS, que contiene un nodo llamado **rqt\_graph**. Dicho nodo muestra al usuario los distintos nodos activos y las relaciones entre sus tópicos mediante una interfaz gráfica. Esto permite verificar de forma rápida y eficaz que la comunicación entre los distintos nodos se está llevando a cabo de forma correcta. El uso de esta herramienta es el siguiente:

- Abrir un nuevo terminal de Ubuntu.
- **\$ rosrun rqt\_graph rqt\_graph**

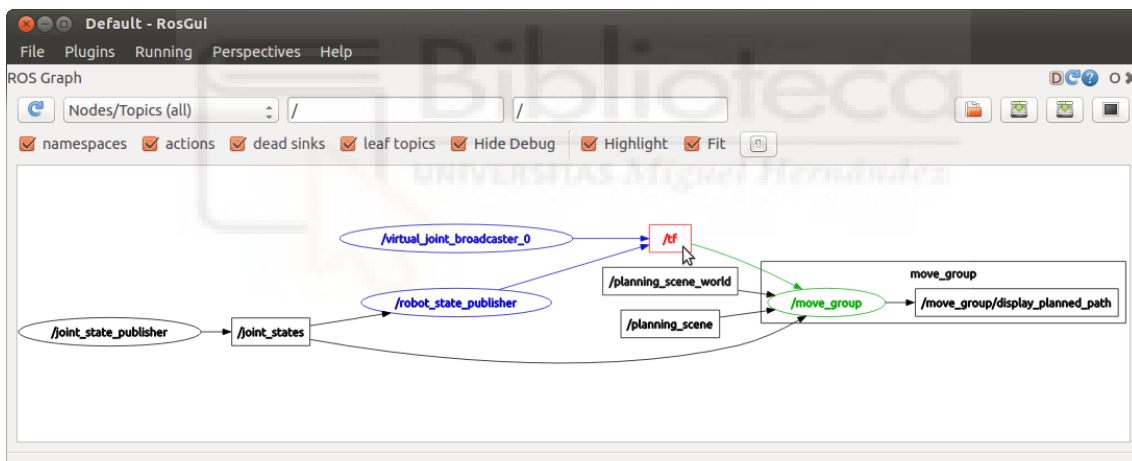


Figura 2.15: Ejemplo de GUI generada por rqt\_graph. Fuente: [17]

## Mensajes

Cuando se habla de **mensajes** en ROS, se está haciendo referencia a una estructura de datos, la cual, está compuesta por varios tipos de datos primitivos (enteros, *floats*, booleanos, etc.) o por un vector de estos (*array*). Dicha estructura, o mensaje, es la encargada de llevar la información a los tópicos.

ROS proporciona una gran cantidad de tipos de mensajes con los que trabajar, pero además, tiene la opción de crear nuevos tipos de mensajes. El método para crear nuevos mensajes se explica en [18].

### Comandos para mensajes en ROS:

Para obtener información sobre los tipos mensajes en ROS tenemos el comando **rosmmsg**. Toda la información sobre este comando se puede consultar en [19].

### Tópicos

Los **tópicos** se consideran los canales de comunicación a través de los cuales los nodos envían y reciben **mensajes**. El envío y recepción de mensajes es independiente. Los nodos no se refieren entre sí para obtener mensajes, se refieren a tópicos. Un nodo se suscribe o publica los tópicos que necesite ignorando si van a ser usados o no, o quién los va a utilizar. Esto permite que exista independencia entre la generación de datos y el consumo de estos. Varios nodos pueden suscribirse o publicar en el mismo tópico. No se recomienda que distintos nodos publiquen en el mismo tópico, pudiendo sobrescribir los datos de este.

Los tópicos están fuertemente tipados, es decir, cada tópico tiene un tipo de mensaje asociado. Para suscribirse a un tópico, los nodos han de indicar el nombre de dicho tópico, el tipo de mensaje que tiene asociado. Lo mismo se aplica a la hora de publicar tópicos.

### Comandos para tópicos en ROS:

Para trabajar con tópicos ROS dispone del comando **rostopic**. Este comando permite mostrar el mensaje que está actualmente en dicho tópico, publicar un mensaje en un determinado tópico, ver a la frecuencia a la que se actualiza, la lista de los tópicos activos y más opciones que se nombran a continuación.

- **rostopic bw**: muestra el ancho de banda usado por el tópico indicado.
- **rostopic delay**: muestra el retraso que tiene determinado tópico.
- **rostopic echo**: muestra el mensaje asociado al tópico por pantalla.

- **rostopic find:** muestra los tópicos con un determinado tipo de mensaje.
- **rostopic hz:** muestra la frecuencia a la que un tópico está siendo publicado.
- **rostopic info:** muestra información general sobre el tópico.
- **rostopic list:** muestra los tópicos activos.
- **rostopic pub:** permite publicar manualmente en un tópico.
- **rostopic type:** muestra el tipo de mensaje asociado a dicho tópico.

En la Wiki oficial de ROS encontramos toda la información sobre este comando y su método de uso [20].

### Servicios

Un **servicio** es una herramienta de ROS, que permite realizar acciones de pregunta y respuesta entre nodos. En caso de necesitar una respuesta inmediata a una determinada pregunta, se deben usar servicios en lugar de usar tópicos. Su funcionamiento consiste en disponer de dos nodos, uno que realiza la función de servidor y otro que realiza la función de cliente.

Como ejemplo de servicio, se propone la realización de una suma. Por un lado, tendremos un nodo que ejercerá la función de servidor, donde se realizará la operación de suma. Por el otro lado, dispondremos de un nodo cliente, que será el encargado de llamar al servicio ofrecido por el servidor y proporcionarle los valores a sumar. Este ejemplo se desarrolla en [21].

Al igual que los tópicos tienen un tipo de mensaje asociado, los servicios tienen un tipo de servicio asociado, el cual está formado por dos tipos de mensajes, uno para la pregunta (tipo de mensaje de entrada) y uno para la respuesta (tipo de mensaje de salida).

Para profundizar en este tema y desarrollar servicios se facilita la referencia [21]. Además, existe la posibilidad de crear servicios definidos por el usuario como se explica en [18].

### Comandos para servicios en ROS:

ROS dispone de dos comandos en relación con los servicios. El primero de ellos es **rossrv**, el cual se utiliza para obtener información sobre el servicio.



El segundo, se trata de **rosservice**, cuya función más importante es realizar llamadas a dichos servicios. Las opciones de cada uno de ellos y su uso se pueden consultar en [19], para *rossrv*, y en [22] para *rosservice*.

### **Parámetros**

Los parámetros son valores normalmente estáticos que se guardan en el *parameter\_server*. Dichos valores, se usan para establecer la configuración de los nodos y permiten a estos últimos tomar valores durante su ejecución.

Hay varias maneras de indicar a un nodo los valores de sus parámetros. La primera de ellas es indicarle el valor manualmente a la hora de iniciar el nodo mediante *rosvrun*. La segunda es indicar estos valores dentro de un *launch\_file* que inicie el nodo. Por último, y la que se ha seguido en este trabajo es la creación de un archivo de texto en formato YAML que se cargará junto con el nodo, en un *launch\_file*. Prestar especial atención a la redacción de los archivos YAML, dado que no aceptan tabulaciones, sólo espacios.

### **Comandos para parámetros en ROS:**

ROS proporciona el comando ***rosparam*** para trabajar con parámetros. Este comando ofrece la posibilidad de obtener valores de parámetros, colocar valores en parámetros, cargarlos desde un archivo, guardarlos en un archivo, borrar parámetros y mostrar una lista de todos ellos.

Toda la información sobre este comando y su uso se encuentra en la siguiente referencia [23].

### **Launch Files**

Los *launch\_files* son una herramienta que proporciona ROS para lanzar varios nodos a la vez. Dispone de la posibilidad de indicar los parámetros de cada nodo, cargar un archivo de parámetros referentes a determinado nodo, cambiar el tópico al que dichos nodos se suscriben/publican o incluso, lanzar otros *launch\_files*.

```
<launch>
  <!--INICIAR hector_mapping-->
  <node name="hector_mapping" pkg="hector_mapping" type="hector_mapping"> <!--INICIA EL NODO INDICADO-->
    <remap from="/scan" to="/sim/scan"/> <!--CAMBIA EL TÓPICO AL QUE SE SUSCRIBE EL NODO-->
    <roscparam file="$(find p3at_simulation)/params/hector_params.yaml" command="load" /> <!--CARGA ARCHIVO DE PARÁMETROS-->
  </node>
</launch>
```

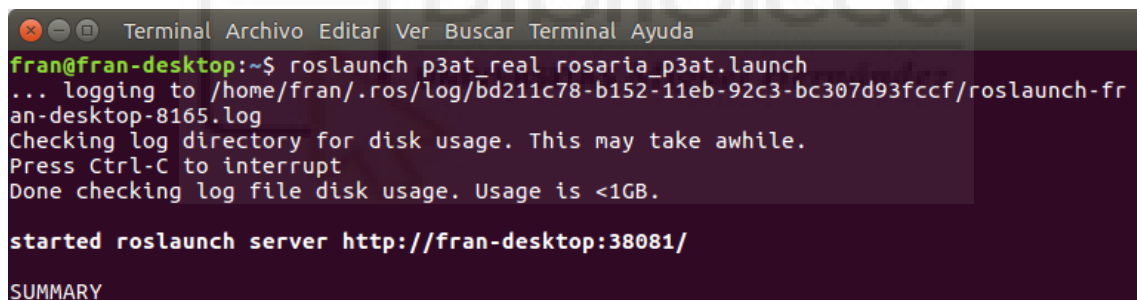
Figura 2.16: Ejemplo de *launch\_file*.

Si se quiere aprender a escribir un *launch\_file* desde cero disponemos de la siguiente referencia [24].

### **Comandos para *launch files* en ROS:**

ROS dispone del comando ***roslaunch***, el cual permite lanzar desde el terminal de Ubuntu cualquier *launch\_file* de forma sencilla. Este comando se encarga automáticamente de iniciar *roscore* en caso de que este no esté activo. Su uso es el siguiente:

- ***\$ roslaunch <nombre\_paquete> <nombre\_launch\_file>***



```
Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ roslaunch p3at_real rosaria_p3at.launch
... logging to /home/fran/.ros/log/bd211c78-b152-11eb-92c3-bc307d93fccf/roslaunch-fran-desktop-8165.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://fran-desktop:38081/
SUMMARY
```

Figura 2.17: Ejemplo de *roslaunch*.

En la referencia [25], se detalla el uso de este comando.

### **Bagfiles**

Es una herramienta que permite almacenar todos los mensajes que se ejecutan en un programa de ROS y posteriormente reproducirlos. Esta herramienta, está destinada a realizar experimentos, guardar datos de estos, y después reproducirlos con distintas configuraciones para desarrollar y depurar nuestro programa sin necesidad de repetir el experimento cada vez.

### Comandos para bagfiles en ROS:

El comando **roscap**, permite guardar y reproducir mensajes. Además, permite seleccionar si queremos guardar los mensajes de todos los tópicos, de una serie de tópicos en concreto, de todos los tópicos suscritos o publicados por un determinado nodo, y más opciones que podemos encontrar en [26].

### 2.2.2.3. CONEXIÓN REMOTA ENTRE EQUIPOS CON ROS

Haciendo uso del comando de Ubuntu antes mencionado, **ssh**, podemos realizar una conexión remota entre dos equipos conectados a la misma red. Esto, permite ejecutar ROS en varios equipos, compartiendo una misma red de comunicación donde se publican los tópicos, parámetros, etc. Para que la red funcione adecuadamente ha de existir un sólo *rosmaster* y se ha de indicar a ambos equipos donde se encuentra dicho maestro. A continuación, se detalla el procedimiento para realizar la conexión remota entre dos supuestos equipos.

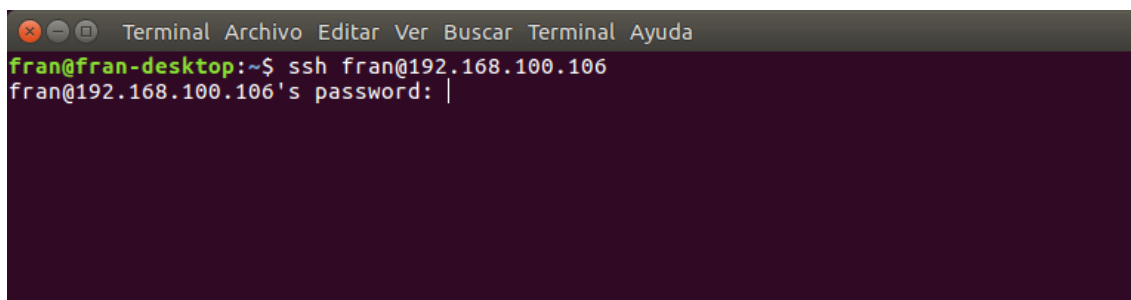
#### Equipos:

- Equipo de **control** con nombre *fran* y dirección IP:192.168.120.126
- Equipo **remoto** con nombre *arvc* y dirección IP: 192.168.120.125

#### Procedimiento:

- Abrir nuevo terminal de Ubuntu desde el equipo de control.
- **\$ ssh arvc@192.168.120.125**

Primero se detalla el nombre de usuario del equipo al que deseamos conectarnos (*arvc*), y después se indica su dirección IP (*192.168.120.125*). Una vez ejecutado el comando anterior, en caso de que el equipo al que queremos conectarnos tenga contraseña, el sistema nos pedirá que la indiquemos. El sistema no muestra los dígitos escritos al poner la contraseña, así que debemos introducirla y pulsar intro.



```
Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ ssh fran@192.168.100.106
fran@192.168.100.106's password: |
```

Figura 2.18: Conexión por ssh con el propio equipo.

Si la contraseña es correcta, el terminal se reiniciará y veremos cómo el nombre de usuario y equipo cambia. En el ejemplo estamos realizando una conexión por ssh con el propio equipo, por lo tanto el nombre de usuario no variará. Dicho terminal es un terminal del equipo remoto, y todo lo que ejecutemos en ese terminal, será como si lo estuviésemos ejecutando directamente en el equipo remoto.

Una vez seguidos estos pasos, ya tenemos la posibilidad de ejecutar comandos por terminal en un equipo remoto. Una vez hecho esto, debemos indicar a ambos equipos, su propia dirección IP y dónde se va a estar ejecutando el *rosmaster*. Esto se consigue mediante los siguientes pasos:

#### Terminal de equipo remoto:

- **`$ export ROS_IP=192.168.120.125`**

Indica a ROS cuál es la IP del equipo actual.

- **`$ export ROS_MASTER_URI=https://192.168.120.125:11311`**

Indica a ROS la dirección IP del equipo en el que se ejecutará el maestro.

#### Terminal de equipo de control:

- **`$ export ROS_IP=192.168.120.126`**

Indica a ROS cuál es la IP del equipo actual.

- **`$ export ROS_MASTER_URI=https://192.168.120.125:11311`**

Indica a ROS la dirección IP del equipo en el que se ejecutará el maestro.

Cómo se puede observar, el valor *ROS\_MASTER\_URI* ha de ser el mismo en ambos equipos, indicando así, que el *rosmaster* sólo ha de estar presente en el equipo al que corresponde la dirección IP de *ROS\_MASTER\_URI*. Una vez hecho esto, ya está establecida totalmente la conexión entre los equipos y configurado su funcionamiento con ROS.

Para mayor comodidad a la hora de trabajar, se recomienda añadir al archivo *“.bashrc”* de cada equipo, los comandos *export* arriba indicados. De esta manera cada vez que se abra un nuevo terminal, ya quedará indicado donde se va a ejecutar el *rosmaster* y la IP del equipo correspondiente.

```
## CARPETAS DONDE ESTARÁN NUESTROS ELEMENTOS DE ROS ##  
  
source /opt/ros/kinetic/setup.bash  
source ~/programas_ros/catkin_ws/devel/setup.bash  
  
## CONFIGURACION PARA CONEXIÓN CON ROBOT ##  
  
export ROS_MASTER_URI='http://192.168.120.125:11311'  
export ROS_IP='192.168.120.126'
```

Figura 2.19: Últimas líneas del archivo *“.bashrc”* para el equipo de control.

Atención, esto provoca que sólo se pueda ejecutar el *rosmaster* en el equipo al que corresponde la IP en *ROS\_MASTER\_URI*. En caso de querer ejecutar nodos solamente en un equipo, sin la existencia de conexión remota, se recomienda comentar las líneas añadidas al archivo *“.bashrc”* para que no existan problemas a la hora de ejecutar el *rosmaster* en dicho equipo.

#### 2.2.2.4. TRASPASO DE ARVHICOS ENTRE EQUIPOS

Normalmente, los paquetes se desarrollan en el equipo de control, y después se pasan al equipo remoto, ubicado en el robot. Para hacer esta tarea más sencilla y sin necesidad de cables, disponemos del comando de Ubuntu **scp**, mediante el cual podemos pasar archivos desde un equipo, a otro conectado en la misma red. En caso de querer enviar un paquete completo,

habrá que comprimirlo antes de ser enviado. Para comprimir desde terminal se emplea el comando **tar**.

### Procedimiento:

#### Terminal del equipo donde se encuentra el paquete:

- Abrir nuevo terminal de Ubuntu.
- Moverse hasta la carpeta `/src` que contiene el paquete en cuestión.
- **`$ tar -czfv p3at.tar.gz p3at_real`**  
El comando anterior comprime el fichero `p3at_real` y genera un archivo llamado `p3at.tar.gz`.
- **`$ scp p3at.tar.gz arvc@192.168.120.125:/home/arvc/catkin_ws/src`**  
El primer argumento del comando `scp` (`p3at.tar.gz`), corresponde al archivo que se desea enviar. El segundo argumento, corresponde al nombre de usuario, IP del equipo y ruta del directorio donde se desea enviar el archivo.

Una vez enviado el archivo, sólo queda conectarnos al equipo donde se ha enviado y descomprimirlo.

- Abrir nuevo terminal de Ubuntu.
- **`$ ssh arvc@192.168.120.125`**
- Introducir contraseña del usuario `arvc` y pulsar intro.
- **`$ cd /home/arvc/catkin_ws/src`**  
Nos movemos la carpeta donde se ha enviado el archivo.
- **`$ tar -xzf p3at.tar.gz`**  
Descomprime el archivo `p3at.tar.gz` en el directorio actual.

Terminado este paso, ya estaría listo para usar el paquete enviado al segundo equipo.

### 2.2.3. ROSARIA

**RosAria** es un paquete que actúa como interfaz entre ROS y la mayoría de robots fabricados por Adept MobileRobots, entre ellos el Pioneer-3AT. Dicho paquete obtiene datos como la velocidad, odometría, aceleración del controlador interno del robot y los traduce a mensajes y tópicos con los que ROS pueda

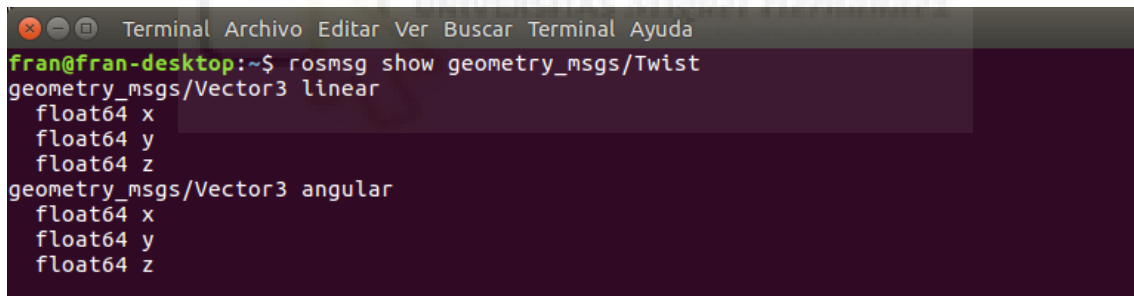
trabajar. Para su correcto funcionamiento, dicho paquete ha de encontrarse ubicado en el equipo montado en el robot.

### Nodo RosAria

El paquete RosAria dispone de un nodo que es el encargado de realizar todas sus funciones. Dicho nodo se denomina igual que el paquete, **RosAria**. A continuación, se detallan los tópicos a los que se suscribe, los tópicos en los que publica y los parámetros de que dispone para su configuración.

### Suscripción a tópicos

El nodo RosAria sólo se suscribe a un tópico. El nombre de este tópico por defecto es `/RosAria/cmd_vel` y contiene mensajes del tipo `geometry_msgs/Twist` [27]. La estructura de este tipo de mensaje está compuesta por un vector de tres posiciones que almacena la velocidad lineal en cada una de las direcciones de los ejes (x, y, z) y un segundo vector que hace lo mismo que el anterior, pero registrando las velocidades angulares. Para el Pioneer-3AT, dada su construcción y movimiento, sólo nos interesa la componente x del vector lineal, y la componente z del vector angular.



```
Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ rosmmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z
```

Figura 2.20: Formato `geometry_msgs/Twist`

Realizando una simple publicación en el tópico `/RosAria/cmd_vel`, el usuario es capaz de indicar al Pioneer-3AT las velocidades de avance y rotación deseadas. El controlador del Pioneer-3AT mantendrá la velocidad indicada mientras que no se publique ningún nuevo mensaje en `/RosAria/cmd_vel`.

### Publicación de tópicos


RosAria publica una serie de tópicos, pero no todos ellos contienen información dado que este paquete da soporte a distintas configuraciones de robots. En nuestro caso, nos vamos a centrar en un solo tópico, `/RosAria/pose`.

`/RosAria/pose` publica la información relativa a la odometría del robot. `/RosAria/pose` contiene mensajes del tipo `/nav_msgs/Odometry` [28]. Este tipo de mensajes contienen información sobre la posición y orientación del robot respecto de su inicio y de la velocidad del robot con un cierto error (covarianza). La odometría es una estimación de la posición del robot. Se obtiene mediante la lectura de los encoders de las ruedas, y representa la distancia y el giro recorrido en cada uno de los ejes respecto de donde se ha iniciado el robot. Más información sobre la odometría disponible en [29].

Cabe comentar, que RosAria publica la transformada entre los *frames* `base_link` (base del robot) y `odom` (odometría del robot). El concepto de *frame*, transformada y odometría se trata en el tercer capítulo de este trabajo.

### Parámetros

Para mayor comodidad a la hora de iniciar los sistemas del robot, se ha creado un *launch\_file* ("[rosaria\\_p3at.launch](#)") junto con un archivo de parámetros ("[rosaria\\_params.yaml](#)") para iniciar el nodo RosAria.

Para la configuración de los parámetros se recomienda prestar especial atención en el parámetro `/RosAria/port` (puerto serie del ordenador al que esté conectado el robot) dado que puede provocar errores a la hora de intentar lanzar el nodo RosAria. El resto de parámetros se recomienda dejarlos de serie. Puede consultar los archivos en el apartado de anexos. Adicionalmente se adjunta a este documento un archivo comprimido que contiene los paquetes desarrollados para este trabajo. 

El paquete RosAria no está disponible en los repositorios oficiales de ROS, por lo que habrá que instalarlo desde GitHub. El proceso de instalación se muestra a continuación:

- ***Abrir terminal de Ubuntu.***
- ***\$ cd ~/catkin\_ws/src***  
Accede a la carpeta `/src` dentro del *workspace*.
- ***\$ sudo git clone https://github.com/amor-ros-pkg/rosaria.git***  
Descarga el paquete desde GitHub.
- ***\$ sudo dpkg -i libaria\_2.9.1+ubuntu16\_i368.deb***  
Instala librerías necesarias para RosAria.



- **\$ cd ~/catkin\_ws**  
Accede al directorio principal del *workspace*.
- **\$ catkin\_make**  
Compilamos dicho *workspace*.

En caso de necesitar más información sobre el uso, o la instalación del paquete RosAria dirigirse a [30].

#### 2.2.4. SICKTOOLBOX\_WRAPPER

**Sicktoolbox\_wrapper** se trata de un paquete desarrollado por Morgan Quigley, bajo licencia BSD, para interactuar en ROS con los láseres Sick LMS2XX mediante la librería *sicktoolbox*.

Dicho paquete contiene dos nodos, el primero **sicklms**, esá orientado a trabajar con los modelos LMS de la marca. Por otro lado, tenemos el nodo **sickl**, que del mismo modo que el anterior, está orientado a los modelos LD de la marca. Cómo se ha comentado en un apartado anterior el láser utilizado en el desarrollo de este trabajo es un Sick LMS200, por lo tanto, nos centraremos solamente en el nodo *sicklms*. Al igual que *RosAria*, este paquete ha de estar ubicado en el equipo montado en el robot.

##### Nodo sicklms

Dicho nodo en cuestión, permite acceder a los datos generados por el láser en un formato comprensible por ROS y configurar el funcionamiento de este a través de parámetros.

##### Tópicos publicados

Este nodo tan sólo publica un tópico, sin realizar ninguna suscripción. El tópico publicado se designa con el identificador **/scan** y tiene asociado el formato de mensaje *sensor\_msgs/LaserScan*.

##### Parámetros

Los parámetros utilizados en este nodo, así como la función de cada uno de ellos, se encuentra detallada en el archivo "[sick\\_lms200\\_params.yaml](#)" ubicado en el apartado de anexos. Además, se anexa el archivo

“[sick\\_lms200\\_p3at.launch](#)” para iniciar dicho nodo con los parámetros que se le han especificado en “[sick\\_lms200\\_params.yaml](#)”.

### 2.2.3. GAZEBO

Gazebo es una de las mejores aplicaciones de software libre que se utiliza para realizar simulaciones dentro del campo de la robótica. Dispone de características como múltiples motores de físicas (ODE, Bullet, Simbody, DART), visualización mediante un potente motor de renderizado, modelos 3D de robots comerciales, extensa gama de sensores y cámaras y posibilidad de desarrollar tus propios sensores mediante plugins.

Gazebo tiene la posibilidad de integrarse con ROS, lo que lo hace perfecto para este trabajo. Además, el código para un determinado programa en ROS es el mismo tanto para simulación como para un entorno real, permitiendo el desarrollo casi completo en simulación.

#### Instalación

En primer lugar, hay que encontrar cual es la mejor combinación de versiones de Gazebo y ROS, para ello disponemos de [31]. Para nuestro caso, ROS Kinetic, la mejor versión de Gazebo es la 7.

Primero, instalar **Gazebo 7** en el sistema:

- Abrir terminal de Ubuntu.
- `$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'`  
Da permiso a Ubuntu para instalar software de la URL indicada.
- `$ wget https://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -`  
Configura las claves de acceso a la URL indicada.
- `$ sudo apt-get update`  
Actualiza la lista de paquetes disponibles para Ubuntu.
- `$ sudo apt-get install gazebo7`  
Descarga e instala Gazebo en su versión 7.

Segundo, instalar el paquete **`gazebo_ros_pkgs`**, encargado de comunicar ROS y Gazebo.

- Abrir terminal de Ubuntu.
- **\$ sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control**  
Instala `gazebo_ros_pkgs` y `gazebo_ros_control` desde los repositorios.

### Iniciar Gazebo

Una vez instalado, podemos ejecutar Gazebo en relación con ROS de la siguiente manera:

- Abrir nuevo terminal de Ubuntu.
- **\$ roscore**  
Inicia el maestro.
- Abrir nuevo terminal de Ubuntu.
- **\$ rosrun gazebo\_ros\_pkgs gazebo**  
Ejecuta Gazebo en relación con ROS.

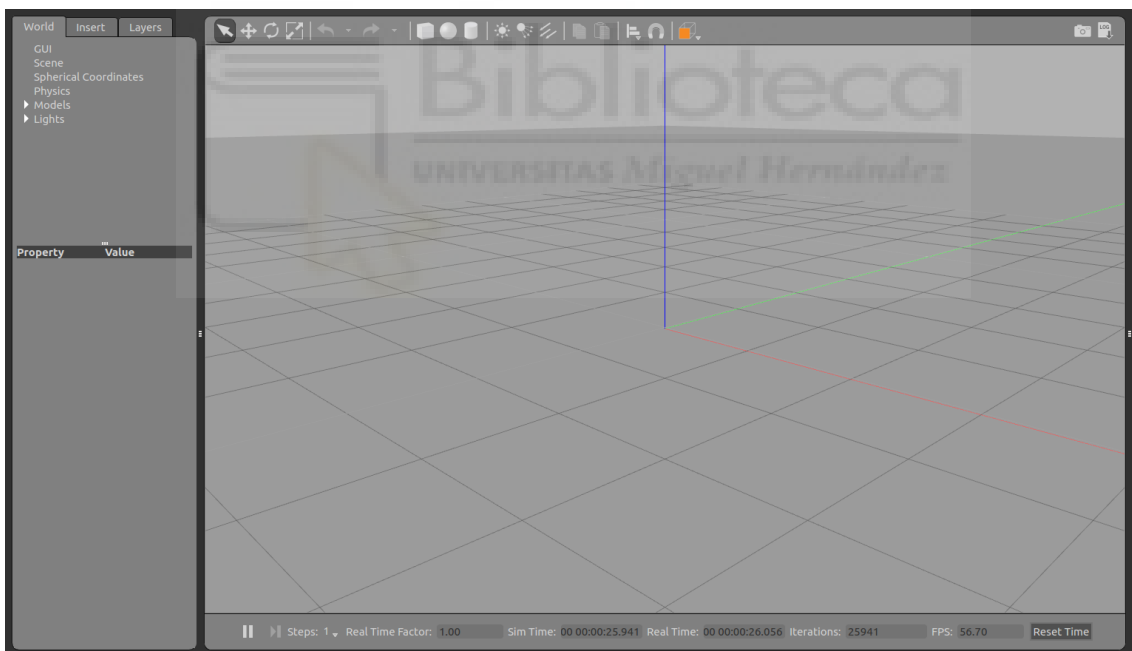


Figura 2.21: Pantalla principal de Gazebo

En la *Figura 2.22: Pantalla principal de Gazebo*, se muestra la interfaz de Gazebo, desde aquí, entre otras cosas, se pueden insertar modelos de robots disponibles, crear nuevos modelos y crear nuevos entornos de trabajo para nuestra simulación, a partir de ahora los llamaremos `gazebo_worlds`.

Para realizar una simulación en gazebo necesitaremos 2 elementos como mínimo, un archivo que describa el robot a emplear (*gazebo\_model*) y otro que describa el entorno (*gazebo\_world*).

### **URDF (*Unified Robotic Description Format*)**

URDF es un formato de lenguaje en XML, empleado en ROS para describir todos los elementos de un robot. Para un correcto funcionamiento del archivo URDF en Gazebo, hay que seguir las instrucciones indicadas en la web oficial de Gazebo [32]. Gazebo dispone de su propio formato de descripción de modelos, denominado SDF, y cuya documentación podemos encontrar en [33]. Al insertar un modelo URDF en Gazebo, internamente se realizará una conversión de formato URDF a SDF. No es necesaria la comprensión del formato SDF, pero sí recomendable, aunque su formato es bastante similar al URDF.

Podemos encontrar la documentación perteneciente al formato URDF en [34] y la especificación de sus elementos en XML [35]. Los principales elementos que podemos encontrar dentro de un URDF son los siguientes:

- **<robot>**: Engloba todas las propiedades del robot.
- **<link>**: Describe propiedades de cada uno de los elementos individuales del robot como pueden ser la inercia, aspecto visual, geometría, y propiedades de colisión de dicho elemento.
- **<joint>**: Describe la relación que tiene cada uno de los links entre sí. Cada <joint>, parte de un <link> principal y describe la relación con uno secundario. A su vez, este secundario, será el <link> principal de otra <joint>.
- **<sensor>**: Especifica sensores como cámaras o láseres y sus principales características.

- **<gazebo>**: Permite especificar propiedades necesarias para realizar una correcta simulación en Gazebo.

```

<?xml version="1.0" ?>
<robot name="NOMBRE_DEL_ROBOT">
  <!--DESCRIPCIÓN DE TODOS LOS ELEMENTOS DEL ROBOT-->
  <link name="LINK_1">
    <.../>
  </link>

  <link name="LINK_N">
    <.../>
  </link>

  <!--DESCRIPCIÓN DE LAS UNIONES ENTRE LOS LINKS DEL ROBOT-->
  <joint name="NOMBRE_PARA_LA_UNIÓN" type="TIPO_DE_UNIÓN">
    <.../>
  </joint>

  <!--DESCRIPCIÓN DE LAS REFERENCIAS A GAZEBO-->
  <gazebo reference="LINK_1">
    <.../>
  </gazebo>

  <!--DESCRIPCIÓN DE PLUGINS DE GAZEBO-->
  <gazebo>
    <plugin filename="CODIGO_FUENTE_PLUGIN" name="NOMBRE_PLUGIN">
      <.../>
    </plugin>
  </gazebo>
</robot>

```

Figura 2.22: Ejemplo de la estructura de un archivo URDF

Las uniones entre un link y otro se realizan de la siguiente forma:

```

<joint name="1_2_JOINT" type="TIPO_DE_UNIÓN">
  <origin rpy="0 0 0" xyz="0.0 0.0 0.0"/>
  <parent link="LINK_1"/>
  <child link="LINK_2"/>
</joint>

```

Figura 2.23: Ejemplo estructura <joint> URDF

En este caso, *<origin>* *<rpy>*, representa el giro en radianes para cada uno de los ejes (x, y, z). *<origin>* *<xyz>* representa el desplazamiento del origen para los ejes (x, y, z). Estos valores están referenciados a los ejes del *<parent\_link>*. A esto se le conoce como transformada y se estudia en mayor profundidad en el tercer capítulo de este libro.

### Gazebo world

Además del modelo del robot, necesitamos un entorno en el que realizar nuestra simulación, este entorno será nuestro *gazebo\_world*. En esta sección, se explica cómo crear un *gazebo\_world* para realizar la simulación deseada.

Cuando lanzamos Gazebo en su forma predeterminada el entorno de simulación aparece vacío. Para crear un entorno que se asemeje a nuestro entorno real, debemos añadir paredes y objetos al entorno básico de Gazebo.

- Iniciamos Gazebo como se ha descrito con anterioridad.
- Accedemos a la pestaña *Edit -> Building Editor*.

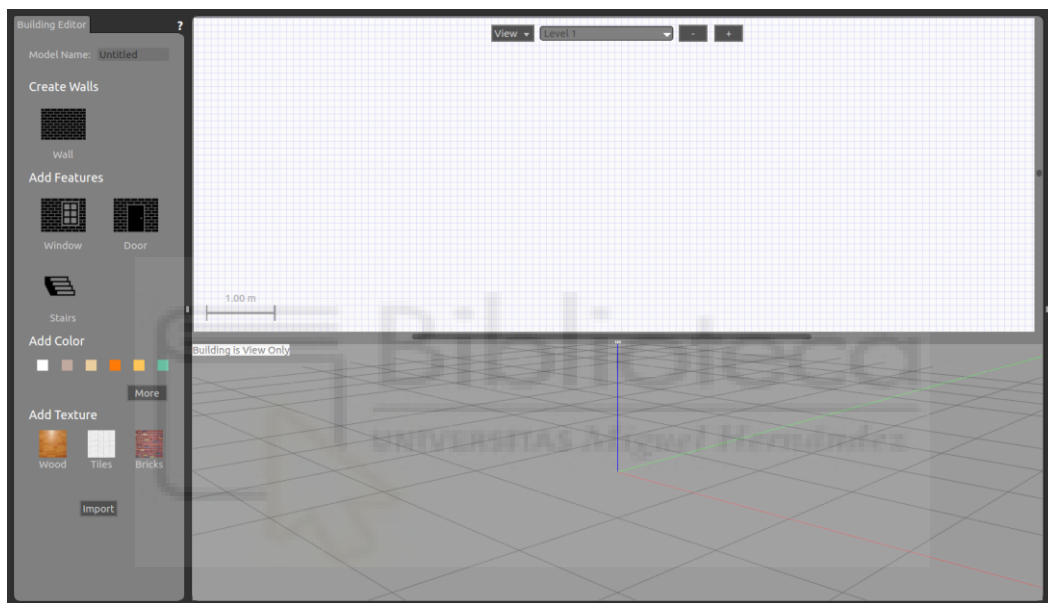


Figura 2.24: Gazebo Building Editor

- Con la opción *Wall* realizamos el diseño de las paredes de nuestro entorno.
- Accedemos a la pestaña *File -> Save As*.
- Accedemos a la pestaña *File -> Exit Building Editor*.
- Añadimos formas deseadas al entorno desde el panel principal de Gazebo.
- Accedemos a la pestaña *File -> Save World As*.

Una vez hecho esto, dispondremos de un archivo con extensión “.world”, que se usará como *gazebo\_world* en la simulación.

### Lanzar simulación

Ahora que ya disponemos de un archivo URDF que describe nuestro robot, y un archivo “.world”, que describe el entorno de nuestra simulación, podemos pasar a lanzar dicha simulación. El inicio de la simulación se realiza mediante un *launch\_file*. Como ejemplo, se proporciona el archivo [“robot\\_p3at\\_lab\\_world\\_simulated.launch”](#), el cual está comentado para su entendimiento, en el apartado de anexos.

#### 2.2.4. RVIZ

RViz se trata de una herramienta que proporciona ROS para visualización de datos de tópicos en un entorno 3D. Como ejemplo, a través de él, se pueden obtener de forma visual, y mucho más intuitiva, datos obtenidos por el láser para contrastar su veracidad. Ofrece una gran variedad de tipos de mensajes que puede mostrar, que van desde la odometría, hasta los datos del láser y el modelo 3D del robot si así lo deseamos.

Su utilización es muy sencilla, a continuación, explicamos un poco su funcionamiento. Para profundizar en el uso del programa se facilita la referencia [36].

#### Iniciar RViz

- Abrir nuevo terminal de Ubuntu.
- **\$ roscore**  
Inicia el maestro.
- Abrir nuevo terminal de Ubuntu.
- **\$ rosrn rviz rviz**  
Ejecuta RViz.

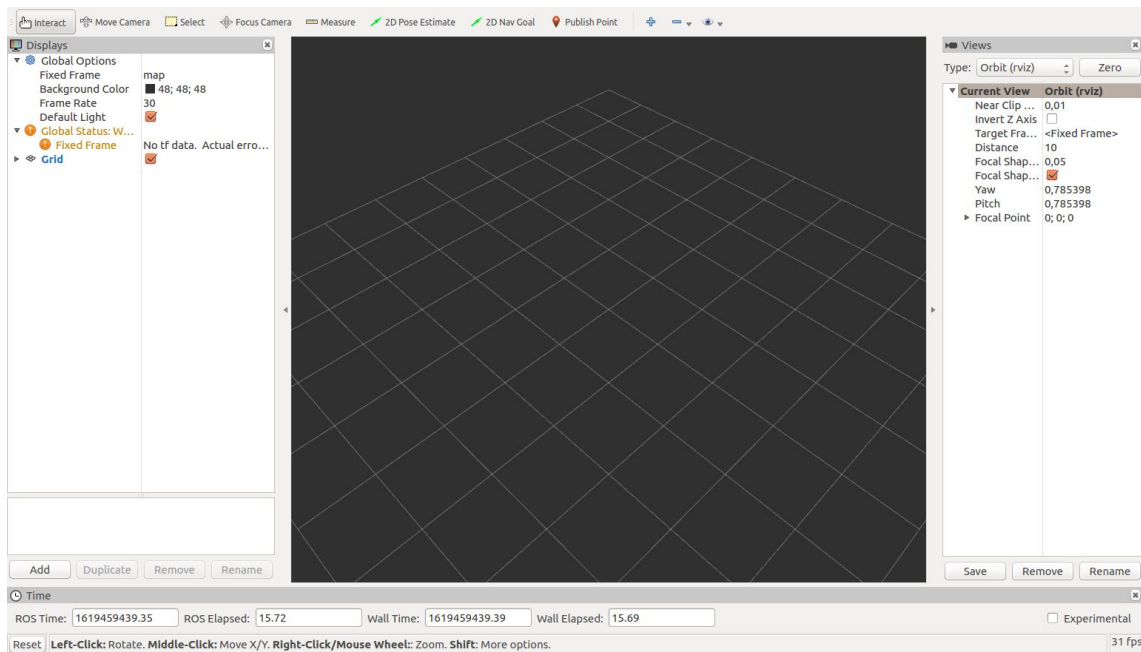


Figura 2.25: Pantalla principal del RViz

Para poder visualizar nuevos tópicos, observamos la parte inferior izquierda, donde se encuentra el botón Add. Una vez lo pulsamos nos aparece lo siguiente:

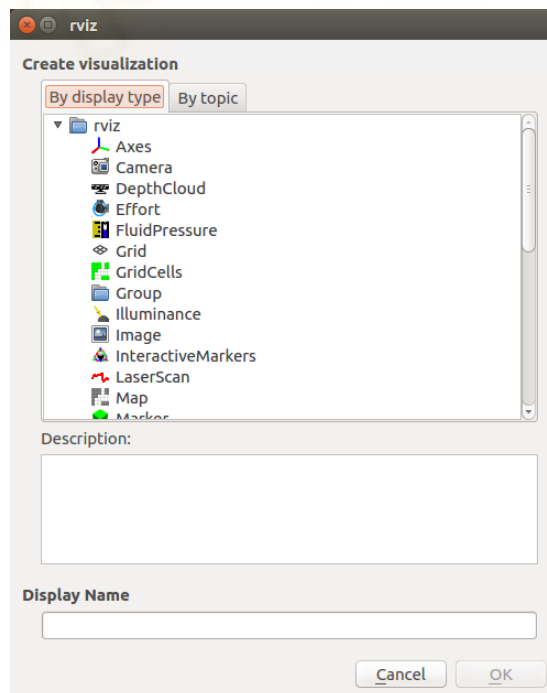


Figura 2.26: Función Add RViz



Desde aquí podemos seleccionar el tipo de mensajes que queremos que se muestren, o podemos seleccionar los tópicos que estén activos en ese momento. Si seleccionamos el tipo de mensaje que queremos mostrar, tendremos que indicar el tópico al que está asociado ese mensaje. Para ello desplegamos el tipo de mensaje añadido y en el apartado tópico, colocamos el tópico deseado.

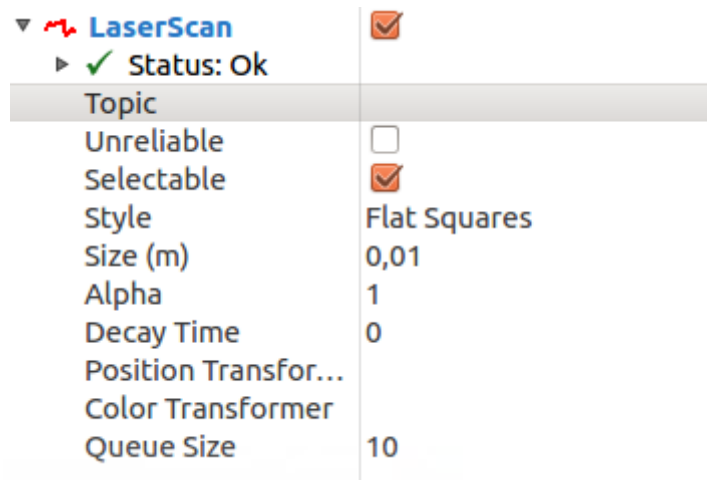


Figura 2.27: Modificar tópico asociado a mensaje en RViz

Una vez que tenemos todos los mensajes/tópicos deseados podemos guardar esa configuración, y cargarla cada vez que la necesitemos para no tener que configurar cada vez todos los mensajes/tópicos que queremos que se muestren. Si al guardar la configuración, se sobrescribe el archivo default, no será necesario cargar ésta, al abrir RViz.

Cuando se simule un entorno con un mapa, en el apartado de *Global Options*, se debe indicar como *fixed frame* el *frame map*.

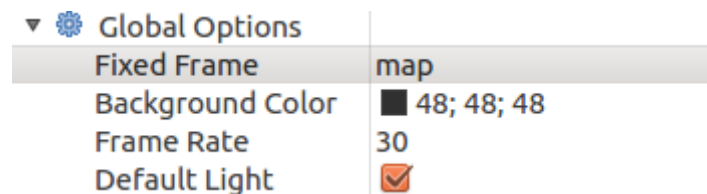


Figura 2.28: Configurar Fixed Frame en RViz

## 3. NAVEGACIÓN

En los capítulos anteriores, se ha hablado del objetivo de este trabajo, el equipamiento con el que se va a realizar y las principales herramientas que ROS pone a nuestro alcance para desarrollar un proyecto.

En este capítulo, se explica qué es la navegación, los requisitos mínimos que se necesitan para desarrollarla, cada una de las funciones a realizar, así como los respectivos paquetes empleados para realiza dichas funciones. Adicionalmente, se comenta una posible solución para realizar una navegación en exteriores.

### 3.1. CONCEPTOS PREVIOS

Antes de adentrarnos en las tareas a realizar para llevar a cabo la navegación de un robot móvil, hemos de tener presentes una serie de conceptos previos.

#### 3.1.1. ODOMETRÍA

La odometría es la estimación de la posición del robot. Se obtiene mediante la lectura de los encoders de las ruedas y representa la distancia y el giro recorrido en cada uno de los ejes respecto de donde se ha iniciado el robot. Cada vez que el robot se enciende y se apaga la odometría se reinicia.

Para trabajar con este concepto en ROS, se utilizan mensajes del tipo `/nav_msgs/Odometry` [28]. Este tipo de mensajes contienen información sobre la posición y orientación del robot respecto de su inicio y de la velocidad del robot con un cierto error (covarianza). Más información sobre la odometría disponible en [29].

En este trabajo los datos de la odometría son publicados por el nodo *RosAria* tal y como está descrito en el capítulo segundo.

### 3.1.2. TRANSFORMADAS

Cuando hablamos de navegación en robótica móvil, se asume que se dispone de una serie de sensores y actuadores que interactúan con el entorno. Dentro del campo de la robótica móvil, a cada elemento de un robot, se le denomina **frame**. Cada *frame* tiene asociado unos ejes locales (x, y, z) asociados. Para que la interacción con el entorno resulte certera, se ha de conocer la posición en todo momento de cada *frame* y las relaciones de traslación y rotación entre cada uno de ellos.

A la unión de dicha traslación y rotación de un *frame* respecto de otro, se le conoce como **transformada**. Una transformada es una herramienta que refleja la traslación y rotación que existe entre los ejes locales de dos *frames* dentro de un mismo sistema robótico.

Cada transformada toma un *frame* como principal y otro como secundario (*child\_frame*).

La **traslación** (x, y, z) de una transformada se define como la distancia de los ejes del *child\_frame* con respecto del *frame* principal.

- x = desplazamiento en metros para el eje x.
- y = desplazamiento en metros para el eje y.
- z = desplazamiento en metros para el eje z.

La **rotación**, (r, p, y) de una transformada se define como el giro en radianes, para cada uno de los ejes del *child\_frame* con respecto del *frame* principal.

- r (*roll*) = giro en radianes sobre el eje x.
- p (*pitch*) = giro en radianes sobre el eje y.
- y (*yaw*) = giro en radianes sobre el eje z.

En la referencia [37], se encuentra un ejemplo de transformada, y como trabajar con ellas.

Un sistema robótico complejo está formado por muchos *frames* y muchas transformadas. Para poder obtener la posición en cada momento de cada *frame* respecto de unos únicos ejes (sistema de coordenadas), se necesita de un árbol

de transformadas (***tf\_tree***). Un *tf\_tree* es la agrupación ordenada de todas las transformadas de un sistema robótico.

Para simplificar el trabajo con transformadas y crear un *tf\_tree*, ROS cuenta con el paquete ***tf*** [38]. Dicho paquete se encarga de generar un árbol de transformadas, además de ofrecer una serie de nodos para trabajar con estas.

Dicho árbol de transformadas se publica bajo el tópico ***/tf***, pero se recomienda hacer uso del paquete ***rqt\_tf\_tree*** para obtener el árbol de transformadas de forma gráfica. Esta última herramienta, permite comprobar de forma sencilla que todas las relaciones entre *frames* existen y son correctas.

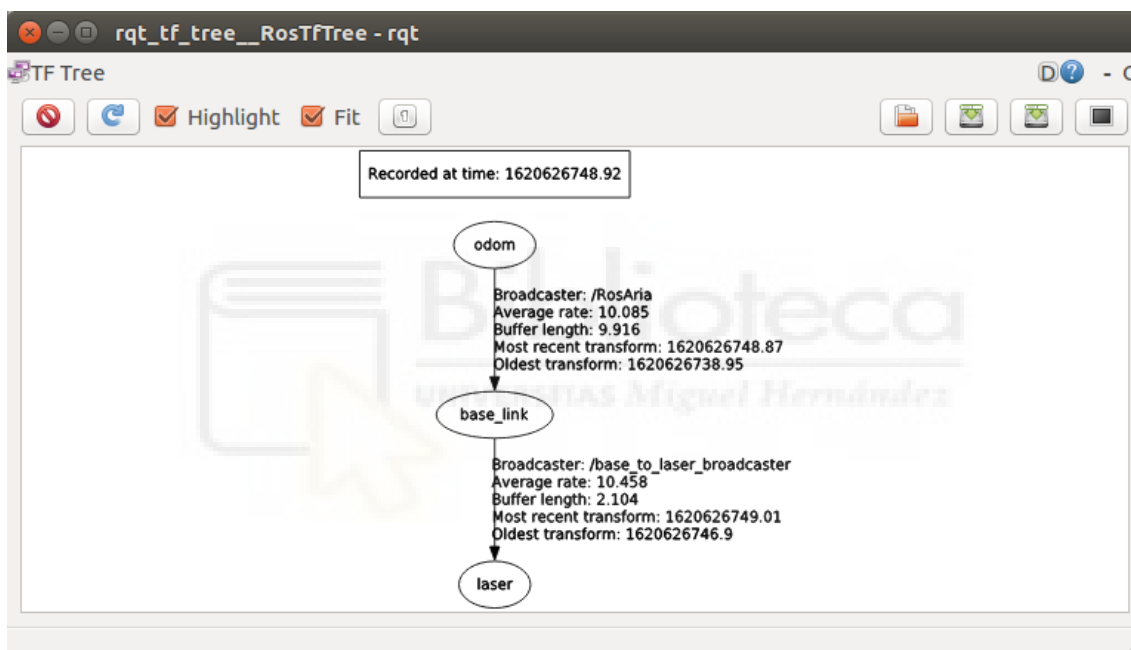


Figura 3.1: Ejemplo *rqt\_tf\_tree*.

## 3.2. INTORUDCCIÓN

La navegación según [39], se define como la metodología que permite guiar el curso de un robot móvil de forma segura, a través de un entorno con obstáculos.

Para este trabajo se emplea el ***Navigation Stack*** que ofrece ROS [40]. Esta herramienta contiene una serie de paquetes, que en su conjunto, permiten

navegar un robot móvil. Para poder emplear dicha herramienta, el robot necesita cumplir una serie de requisitos físicos:

- El sistema de locomoción del robot móvil ha de ser diferencial (ej. "skid-steer") o holonómico. Además, el control de velocidad del robot, ha de realizarse de la forma: velocidad lineal en el eje x, velocidad lineal en el eje y, velocidad angular alrededor del eje z.
- Requiere tener instalado un sensor láser fijado a la base del robot.
- A pesar de que esta herramienta puede usarse en cualquier robot que cumpla los requisitos anteriores, se recomienda que la forma del robot sea cuadrada o circular, dado que ha sido desarrollado en este tipo de robots y su funcionamiento en otro tipo puede ser errático.

La estructura de funcionamiento del *Navigation Stack* es la siguiente:

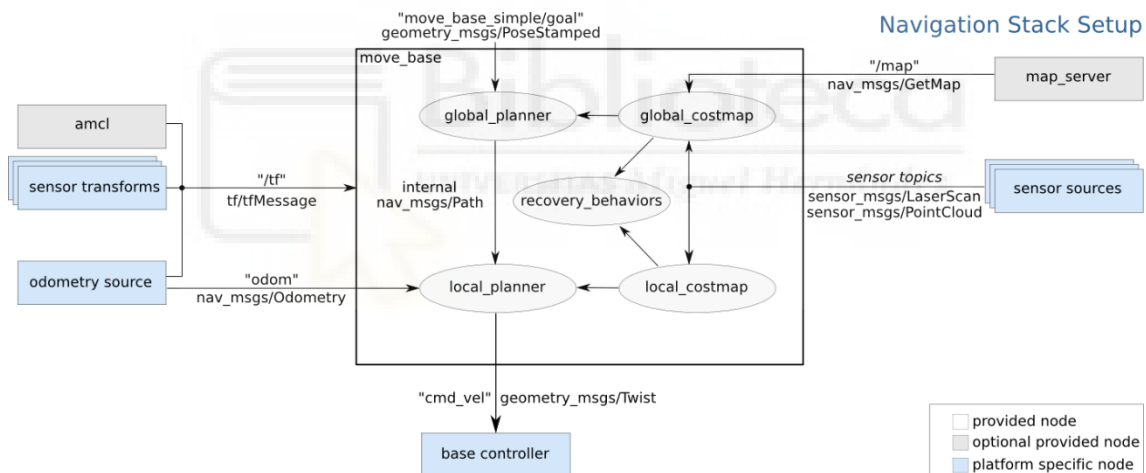


Figura 3.2: Diagrama de configuración del *Navigation Stack*. Fuente: [56]

Como podemos observar en este diagrama, los nodos empleados se clasifican en tres tipos: nodos dados por el *Navigation Stack*, nodos que pueden funcionar junto con el *Navigation Stack*, y nodos que debe dar el usuario y dependen del equipo empleado.

#### Nodos que dependen del equipo

En primer lugar, el usuario debe proveer al *Navigation Stack* la odometría del robot, los datos de los sensores acoplados, las transformadas de estos con respecto a la base del robot y un nodo que controle el movimiento de dicho robot.

Para la publicación de la odometría y el control de la base del robot disponemos del nodo **RosAria**, del cual hemos hablado en el segundo capítulo de este trabajo. En dicho capítulo, también se indica, que el nodo empleado para obtener los datos del láser, es el **sicklms** del paquete *sicktoolbox\_wrapper*.

Además, el sistema requiere de las transformadas de los sensores utilizados, respecto de la base del robot. Para ello, se ha empleado el nodo **static\_transform\_publisher** del paquete *tf*, tal y como se ha explicado en la primera sección de este capítulo.

#### Nodos que trabajan con *Navigation Stack*

Para el funcionamiento de la navegación, se necesita un mapa de ocupación a partir del cual poder navegar y ser capaz localizarse. De ello, se encargan los nodos **map\_server** y **amcl** respectivamente. Estos son los nodos clasificados por su capacidad de funcionar junto con el *Navigation Stack* (*move\_base*).

#### Nodos provistos por el *Navigation Stack*

Por último, y con los datos proporcionados por los nodos mencionados en los párrafos anteriores, el *Navigation Stack* genera comandos de velocidad para navegar a través de un entorno con obstáculos. La generación de los comandos de velocidad se realiza mediante una serie de nodos que están integrados en el paquete **move\_base**. Los nodos que forman parte de *move\_base*, se detallan en la sección que se redacta a continuación.

### 3.3. TAREAS A REALIZAR

Para llevar a cabo este trabajo, se ha dividido la navegación en una serie de tareas a realizar. Dichas tareas son: publicar el árbol de transformadas de nuestro sistema, generar un mapa de ocupación a partir de un sensor láser, publicar dicho mapa, localizarse dentro de este, y publicar comandos de velocidad a la base del robot para realizar una navegación segura.

A continuación, se detalla cada una de estas tareas:

### 3.2.1. ÁRBOL DE TRANSFORMADAS

Para publicar el árbol de transformadas, en primer lugar, necesitamos conocer los *frames* de los que está compuesto nuestro sistema. En este trabajo, el sistema robótico está compuesto por:

- ***/map***: *frame* referente al mapa.
- ***/odom***: *frame* referente a la odometría.
- ***/base\_link***: *frame* referente a la base del robot.
- ***/laser***: *frame* referente al láser.

Para una mayor reutilización de los códigos, los nombres que reciben los *frames* siguen la convención establecida en [41].

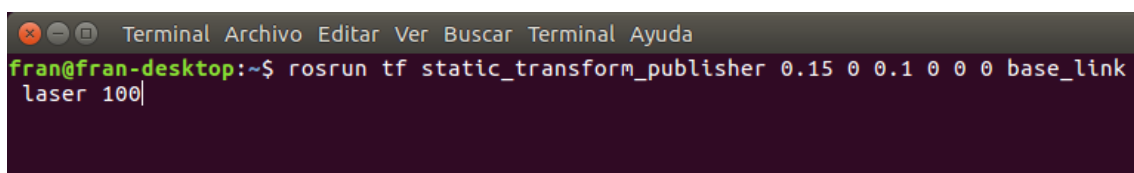
Todas las transformadas excepto la que relaciona */base\_link* -> */laser*, son publicadas automáticamente por los nodos del sistema de navegación. En las siguientes secciones, donde se detallan cada uno de los nodos que componen el sistema de navegación, se indica que transformada publica cada uno de estos.

Dado que la transformada */base\_link* -> */laser* no es publicada automáticamente por los nodos implementados para la navegación, se debe realizar la publicación de esta, de forma manual. Puesto que en nuestro trabajo se dispone de un sólo sensor, el láser Sick LMS200, y que dicho sensor, está fijo a la base del robot, se ha empleado el nodo ***static\_transform\_publisher***.

Este nodo pertenece al paquete *tf* mencionado anteriormente. Permite publicar bajo el tópico */tf* la información relativa a la transformada estática entre dos *frames*.

Su uso es el siguiente:

- Abrir terminal de Ubuntu.
- **`$ rosrn tf static_transform_publisher <x y z> <yaw pitch roll> <frame_principal> <child_frame> <frecuencia_de_publicación>`**



```
Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ rosrn tf static_transform_publisher 0.15 0 0.1 0 0 0 base_link
laser 100
```

Figura 3.3: Ejemplo de uso del nodo *static\_transform\_publisher*.

Para publicar la transformada entre la base del Pioneer 3-AT y el láser se ha creado el archivo “[tf\\_laser.launch](#)”. En este archivo se hace uso del nodo `static_transform_publisher`, al cual se le indican como argumento sus respectivos valores de traslación, rotación, `frame` principal, `child_frame` y frecuencia de publicación.

```
<launch>
  <!-- PUBLISH TRANSFORM BETWEEN laser AND base_link-->
  <node pkg="tf" type="static_transform_publisher" name="base_to_laser_broadcaster" args=" 0.15 0 0.1 0 0 base_link laser 100" />
</launch>
```

Figura 3.4: Imagen del archivo `tf_laser.launch`.

### 3.2.2. OBTENCIÓN DEL MAPA

Para obtener un mapa de ocupación en el que basar la navegación, se ha hecho uso de los paquetes `gmapping` y `map_server`. En esta sección, describiremos cada uno de dichos paquetes por separado, y por último, describiremos el proceso para realizar un mapa de ocupación desde cero.

#### Gmapping

El paquete `gmapping` está destinado a la creación de un mapa de ocupación cuadrangular en dos dimensiones, a partir de un sensor láser colocado en la base del robot. Cuando hablamos de un mapa cuadrangular, quiere decir que éste, está dividido en un número determinado de celdas. Para cada una de esas celdas, se comprueba cuál es su estado (libre, ocupado o desconocido) y en función de éste, se construye dicho mapa de ocupación.

El proceso de creación del mapa es llevado a cabo por el nodo `slam_gmapping`, que realiza las operaciones de localización y creación del mapa de forma simultánea. Al proceso de realizar las operaciones de localización y mapeado simultáneamente, se le conoce como *SLAM* (“*Simultaneous Localization and Mapping*”), de ahí el nombre del nodo. A medida que la base del robot se va moviendo dentro de un determinado entorno, el nodo `slam_gmapping` va creando y publicando el mapa de ocupación bajo el tópico `/map` y la transformada entre el mapa y la odometría en el tópico `/tf`. Toda la



información referente al paquete se ha obtenido de [42]. A continuación, se detallan los requerimientos del nodo, los tópicos asociados a este, y sus parámetros más significativos.

### Nodo *slam\_gmapping*

#### Requerimientos

Los requisitos necesarios para la utilización del nodo *slam\_gmapping* son:

- Robot móvil que proporcione datos sobre su odometría.
- Sensor láser fijo a la base del robot.
- Transformada que permita relacionar los datos obtenidos por el láser con la odometría.

En nuestro caso, el primer requisito, se cumple gracias al nodo *RosAria*, que se comunica con el robot y publica los datos sobre su odometría. En segundo lugar, disponemos del sensor láser Sick LMS-200 montado sobre la base del robot y gracias al nodo *sicklms* podemos obtener sus datos. Por último, la transformada que relaciona el láser con la odometría, se consigue publicando la transformada entre el láser y la base del robot. Puesto que *RosAria* publica la transformada entre la base del robot, y la odometría, tenemos un pequeño árbol de transformadas que permite relacionar el láser con la odometría. En la [Figura 3.1](#), se puede observar el árbol de transformadas que relaciona la odometría con el láser.

#### Uso

- Abrir terminal de Ubuntu.
- **\$ roscore**
- **\$ rosrun gmapping slam\_gmapping**

El comando anterior inicia el nodo *slam\_gmapping* del paquete *gmapping*.

En caso de que el tópico donde se publiquen los datos del láser no sea `/scan`, se debe hacer un remap del tópico, quedando el comando anterior de la siguiente forma:

- \$ **roslaunch gmapping slam\_gmapping scan:= <nombre del tópico donde se publican los datos del láser>**

### Tópicos

#### Suscripciones:

- **/tf**: suscripción al tópico **/tf** para poder relacionar el láser con la odometría.
- **/scan**: suscripción al tópico **/scan**, donde se publican (por el nodo *sicklms*) los datos del láser. Prestar especial atención, que el tópico en el que se publican los datos y el tópico al que se suscribe el nodo, se denominen igual.

#### Publicaciones:

- **/map\_metadata**: publica información sobre las características principales del mapa (resolución, anchura, altura, origen).
- **/map**: publica los datos que conforman el mapa de ocupación cuadrangular en dos dimensiones.

### Parámetros

En este apartado sólo se detallan los parámetros más importantes, aunque se recomienda dejar todos ellos en su forma predeterminada.

- **base\_frame**: indica el nombre del *frame* asociado a la base del robot.
- **map\_frame**: indica el nombre del *frame* asociado al mapa.
- **odom\_frame**: indica el nombre del *frame* asociado a la odometría.
- **map\_update\_interval**: cada cuanto tiempo actualiza el mapa mientras éste está siendo creado.
- **maxUrange**: máximo rango del sensor láser.
- **delta**: resolución del mapa en mm/celda. Para robótica el valor más usado es 0.05 m/celda.

## Map\_server

El paquete *map\_server*, ofrece dos principales funciones. La primera de ellas es la de publicar un mapa de ocupación almacenado previamente, y la segunda, permite almacenar un mapa que esté siendo publicado. De estas dos funciones se encargan los nodos *map\_server* y *map\_saver* respectivamente. Antes de describir cada uno de estos nodos en profundidad, se detalla el formato de mapa con el que trabaja el paquete *map\_server*.

### Formato de mapa

Cada mapa está descrito por un par de archivos. Un archivo YAML que describe las características principales del mapa, y un archivo de imagen, que contiene los datos de ocupación.

#### Archivo de imagen

Los datos de ocupación se almacenan en un archivo de imagen con extensión “.*pgm*”. Dicha imagen está formada de una serie de celdas (cuyo número viene definido por la resolución del mapa y su tamaño), y a cada una de ellas, se le asocia un color en función del estado en que se encuentren.

Posibles estados de las celdas:

- **Ocupado:** celda en color negro.
- **Libre:** celda en color blanco.
- **Desconocido:** celda en color gris.



Figura 3.5: Ejemplo de archivo de imagen de un mapa.

### Archivo YAML

El archivo YAML tiene el siguiente aspecto.

```
image: laboratorio.pgm
resolution: 0.050000
origin: [-100.000000, -100.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Figura 3.6: Ejemplo archivo YAML de un mapa

- **image**: ruta al archivo que contiene los datos de ocupación. En caso de que el archivo de imagen este ubicado en el mismo directorio que el archivo YAML, bastará con indicar el nombre del archivo de imagen.
- **resolution**: resolución del mapa en metros/celda.
- **origin**: posición de la esquina inferior izquierda del mapa indicada de la forma desplazamiento en x, desplazamiento en y, giro respecto de z.
- **occupied\_thresh**: valor de probabilidad a partir del cual una celda se considera ocupada.
- **free\_thresh**: valor de probabilidad a partir del cual una celda se considera libre.
- **negate**: invierte el color asociado al estado de cada celda (ocupado, libre, desconocido).

Una vez descrito el formato de mapa empleado por el paquete *map\_server*, pasemos a describir cada uno de sus nodos.

#### **Nodo *map\_server***

Este nodo se encarga de leer un mapa guardado en el equipo, y publicar dicho mapa, en el formato definido por ROS *nav\_msgs/OccupancyGrid*. Para ello convierte los estados asociados a cada celda de un mapa en valores numéricos, 0 para celdas libres, 100 para celdas ocupadas y -1 para celdas cuyo estado se desconoce.

### Uso

- Abrir terminal de Ubuntu.
- Moverse hasta la carpeta donde están los archivos del mapa.

➤ **\$ roscore**

➤ **\$ rosrun map\_server map\_server laboratorio.yaml**

El comando anterior inicia el nodo *map\_server* del paquete *map\_server* con el argumento *laboratorio.yaml*, que es el archivo YAML que describe las características del mapa. También se puede introducir como argumento la ruta hasta el archivo YAML.

### Tópicos

#### Suscripciones:

Este nodo no necesita estar suscrito a ningún tópico.

#### Publicaciones:

- **/map\_metadata:** contiene las características principales del mapa.
- **/map:** contiene el mapa en formato *nav\_msgs/OcupancyGrid*.

### Parámetros

- **frame\_id:** nombre que se le da al *frame* asociado al mapa. Se recomienda dejarlo predeterminado.

### Nodo map\_saver

Este nodo toma los datos publicados en el tópico */map*, para crear el par de archivos necesarios que describen un mapa (YAML e imagen).

### Uso

- Abrir terminal de Ubuntu.
- Moverse a la carpeta donde se desea que se guarden los archivos que definen el mapa.
- **\$ roscore**
- **\$ rosrun map\_server map\_saver -f laboratorio**

El comando anterior inicia el nodo *map\_saver* del paquete *map\_server* y con la opción *-f* especifica el nombre que se le quiere dar al mapa a guardar.

### Tópicos

#### Suscripciones:

- ***/map***: contiene el mapa en formato *nav\_msgs/OcupancyGrid*.

#### Publicaciones:

Este nodo no publica ningún tópico.

Estos paquetes nos proporcionan las herramientas necesarias para generar un mapa de ocupación a partir de un sensor láser fijado a la base de un robot móvil, y la capacidad de almacenar dicho mapa para su uso a posteriori. A continuación, se describe el proceso para crear dicho mapa desde cero.

### Procedimiento para la obtención del mapa de ocupación

A medida que la base del robot se mueve dentro de un determinado entorno, el nodo *slam\_gmapping* va publicando el mapa de ocupación generado bajo el tópico */map*. Una vez el robot ha recorrido todo el entorno del que se desea crear el mapa, se hace uso del nodo *map\_saver* para almacenar en el equipo el par de archivos que describen dicho mapa.

#### Terminal de control:

- Abrir terminal de Ubuntu.
- **`$ export ROS_IP=192.168.120.126`**
- **`$ export ROS_MASTER_URI=https://192.168.120.125:11311`**
- **`$ ssh arvc@192.168.120.125`**

Tras esto, obtendremos acceso a un terminal del equipo montado en el Pioneer-3AT. El proceso para la conexión remota queda detallado en el punto [2.2.2.2](#) de este trabajo.

### Terminal remoto:

- **\$ export ROS\_IP=192.168.120.125**
- **\$ export ROS\_MASTER\_URI=https://192.168.120.125:11311**
- **\$ roslaunch p3at\_real p3at.launch**

El archivo [p3at.launch](#) se encuentra detallado en el apartado de anexos y se encarga de iniciar el *rosmaster*, lanzar *RosAria*, *sicklms*, y publicar la transformada entre */base\_link* -> */laser*.

- **\$ rosrun gmapping slam\_gmapping**

Iniciamos el nodo encargado de crear el mapa y publicar este en el tópico */map*.

Llegados a este punto, necesitamos un nodo que nos permita enviar comandos de velocidad a la base del robot. Para esta tarea se emplea el nodo ***key\_teleop*** contenido en el paquete ***key\_teleop***. Dicho paquete se puede descargar desde [43].

Este nodo publica comandos de velocidad lineal en el eje X, y de velocidad angular en el eje z, tomando como entrada las flechas del teclado. El tópico en el que publica *key\_teleop* es */cmd\_vel*, por tanto, para que *RosAria* (suscrito a */RosAria/cmd\_vel*) obtenga los comandos de velocidad publicados por *key\_teleop*, hay que renombrar el tópico en el que publica *key\_teleop*, para que coincida con el tópico al que se suscribe *RosAria*.

### Terminal de control:

- Abrir nuevo terminal de Ubuntu en el equipo de control.
- **\$ rosrun key\_teleop key\_teleop cmd\_vel:=RosAria/cmd\_vel**

Se inicia el nodo *key\_teleop* y se modifica el tópico al que publica..

Cabe destacar, que para poder comandar el robot con este nodo, debemos estar en el terminal que está ejecutando el nodo *key\_teleop*.

Opcionalmente, se puede iniciar RViz para visualizar cómo se va creando el mapa a medida que se mueve el robot en el entorno. Esto no es obligatorio, pero sí muy recomendable porque permite asegurarnos de que el mapa del entorno está completo.

### Terminal de control:

- Abrir nuevo terminal de Ubuntu en el equipo de control.
- **\$ rosrun rviz rviz**
- Seguimos el proceso indicado en el punto [2.2.4](#) para visualizar nuevos tópicos en RViz.
- Pulsamos el botón *add* y seleccionamos que queremos un mapa.

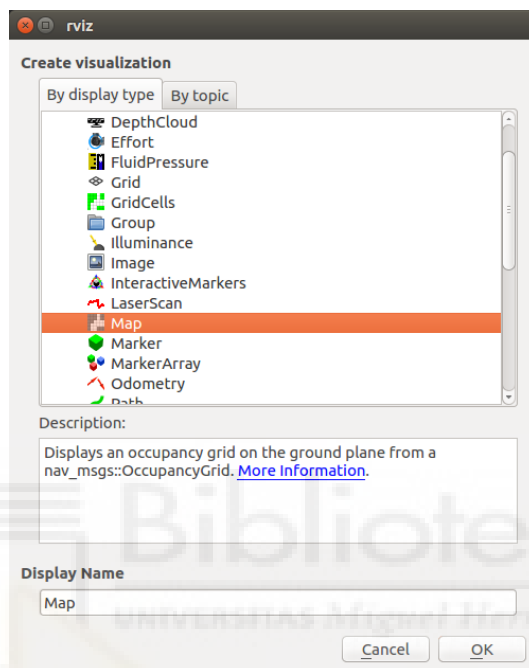


Figura 3.7: Ejemplo añadir mapa a RViz.

- Indicamos el tópicos en el que se está publicando el mapa.

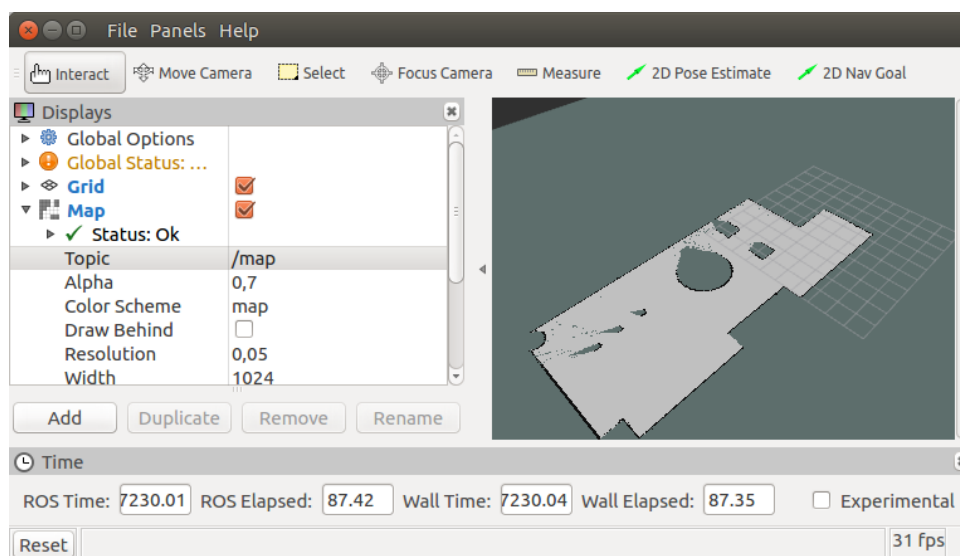


Figura 3.8: Ejemplo visualizar tópicos /map.



Ahora ya podemos mover la base del robot mediante el nodo `key_teleop` y ver cómo se va creando el mapa.

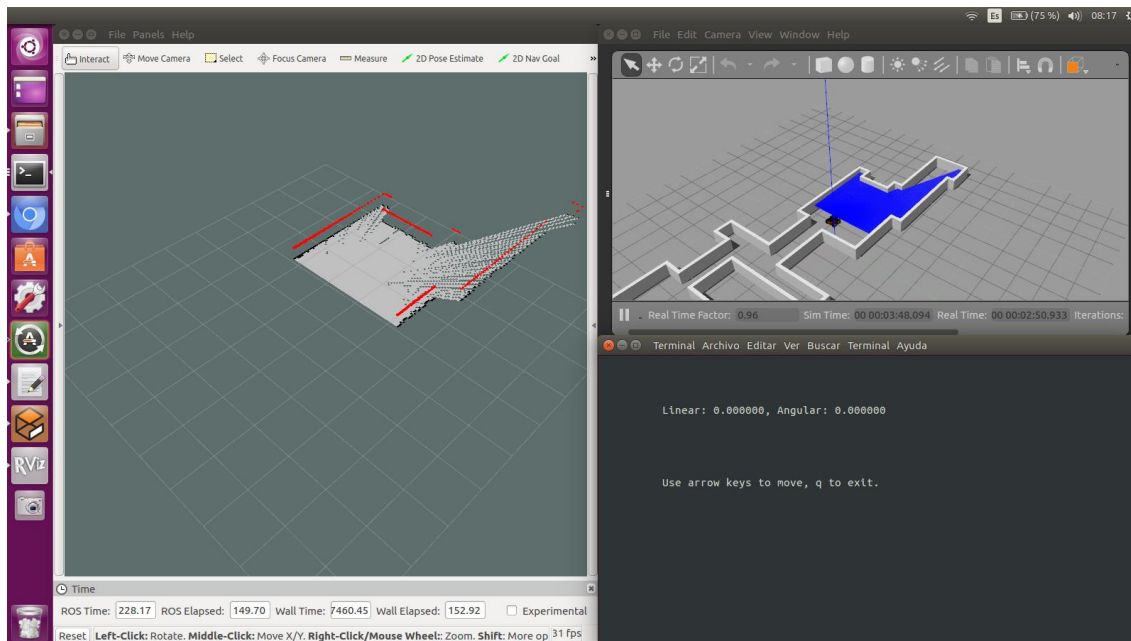


Figura 3.9: Ejemplo de creación de mapa en simulación.

Una vez hemos terminado de mover el robot por el entorno, tenemos que guardar el mapa.

#### Terminal de control:

- Abrir nuevo terminal de Ubuntu en el equipo de control.
- **`$ roscd p3at_real/maps`**  
Nos movemos al directorio donde deseamos guardar el mapa, en este caso `/maps` que está dentro del paquete `p3at_real`.
- **`$ rosrn map_server map_saver -f laboratorio`**  
Guarda el par de archivos que describen el mapa que está siendo publicado en el tópico `/map` con el nombre `laboratorio`.

### 3.2.3. PUBLICACIÓN DEL MAPA

Para emplear el *Navigation Stack* necesitamos que el mapa de ocupación del entorno sea publicado. Para esta tarea empleamos el nodo `map_server` contenido en el paquete `map_server`. El uso de este nodo ya se ha explicado en el punto anterior, por eso en este punto pasaremos directamente al procedimiento para publicar el mapa.

### Procedimiento para publicar el mapa

#### Terminal de control:

- Abrir nuevo terminal de Ubuntu en el equipo de control.
- **`$ roscd p3at_real/maps`**  
Nos ubicamos en el directorio donde esté el mapa que queremos publicar.
- **`$ rosrn map_server map_server laboratorio.yaml`**  
Iniciamos el nodo `map_server` pasando como argumento el archivo YAML que describe las características del mapa.

Para mayor comodidad, se ha creado el archivo [map\\_server.launch](#) con el que también se puede realizar la tarea de publicar el mapa. Se puede consultar dicho archivo en el capítulo de anexos.

#### 3.2.4. LOCALIZACIÓN

Llegado este punto, en el cual tenemos disponibles y configurados correctamente todos los sistemas del robot, y tenemos un mapa de ocupación, podemos realizar la localización en ese mapa. Para ello, se emplea el nodo ***amcl*** contenido en el paquete ***amcl***, el cual, mediante los datos del láser, el mapa, y el *tf\_tree* publica la posición estimada del robot dentro del mapa proporcionado.

Dicho nodo implementa un muestreo de partículas adaptativo (*KLD Sampling*) junto con el algoritmo de *Monte Carlo Localization (MCL)*, el cual, calcula la probabilidad de la posición del robot, dentro de un mapa dado, mediante un filtro de partículas. El nombre *amcl*, no es más que las siglas de *Adaptative Monte Carlo Localization*.

El algoritmo de *Monte Carlo Localization*, comienza con una cantidad de partículas (especificada como parámetro) ubicadas de forma aleatoria dentro del mapa dado. A medida que el robot se mueve por dicho mapa, se actualiza el filtro de partículas y estas se van agrupando alrededor de un punto, el punto central de la agrupación de dichas partículas, indicará la ubicación del robot.

Para más información sobre los algoritmos empleados para el desarrollo del nodo se dispone del libro [44]. Además, toda la información sobre el paquete *amcl* se encuentra detallada en [45].

A continuación, se detalla el uso del nodo *amcl*, tópicos con los que trabaja, y sus principales parámetros.

### Uso

#### Terminal de control:

- Abrir nuevo terminal de Ubuntu en el equipo de control.
- **\$ rosrun amcl amcl**  
Ejecuta la localización con los parámetros por defecto.

### Tópicos

#### Suscripciones:

- **/scan**: contiene los datos publicados por el láser.
- **/tf**: contiene las transformadas entre todos los frames del robot.
- **/initialpose**: especifica la posición inicial estimada del robot.
- **/map**: contiene los datos del mapa de ocupación.

#### Publicaciones:

- **/amcl\_pose**: publica la posición estimada del robot con un determinado error (covarianza).
- **/particlecloud**: publica la nube de partículas usada por el algoritmo para estimar la posición del robot.
- **/tf**: publica la transformada entre el mapa y la odometría.

### Parámetros

Los parámetros de este nodo se pueden dividir en tres ámbitos: parámetros del filtro de partículas, parámetros del sensor láser, parámetros referentes a la odometría. Dado que el número de parámetros que emplea es

muy elevado, se van a mencionar los más importantes, dejando el resto de parámetros con sus valores por defecto.

#### Parámetros del filtro de partículas

- ***min\_particles***: número mínimo de partículas para el filtro.
- ***max\_particles***: número máximo de partículas para el filtro.
- ***kld\_err***: máximo error entre la distribución de las partículas estimada y la real.
- ***update\_min\_d***: distancia (en metros) lineal a recorrer antes de actualizar el filtro de partículas.
- ***update\_min\_a***: giro (en radianes) a realizar antes de actualizar el filtro de partículas.
- ***transform\_tolerance***: tiempo en que la transformada entre el mapa y la odometría se mantiene publicada.

#### Parámetros del sensor láser

- ***laser\_min\_range***: mínima distancia a la que tomar los valores del láser como válidos.
- ***laser\_max\_range***: máxima distancia a la que tomar los valores del láser como válidos.

#### Parámetros de la odometría

- ***odom\_model\_type***: puede tener varios valores de una lista, para robots diferenciales usar “diff” y para robots con ruedas omnidireccionales usar “omni”. En nuestro caso, usaremos “diff”.
- ***odom\_frame\_id***: nombre del *frame* asociado a la odometría.
- ***base\_frame\_id***: nombre del *frame* asociado a la base del robot.
- ***global\_frame\_id***: nombre del *frame* asociado al mapa.
- ***tf\_broadcast***: indica mediante un valor booleano si *amcl* ha de publicar o no la transformada entre el mapa y la odometría.

Para iniciar la localización, se ha creado el archivo “[amcl\\_p3at.launch](#)”. Este *launch\_file* inicia el nodo *amcl* con los parámetros especificados en el archivo “[amcl\\_p3at\\_sicklms200\\_params.yaml](#)”. Dichos archivos se pueden consultar en el apartado de anexos de esta memoria.

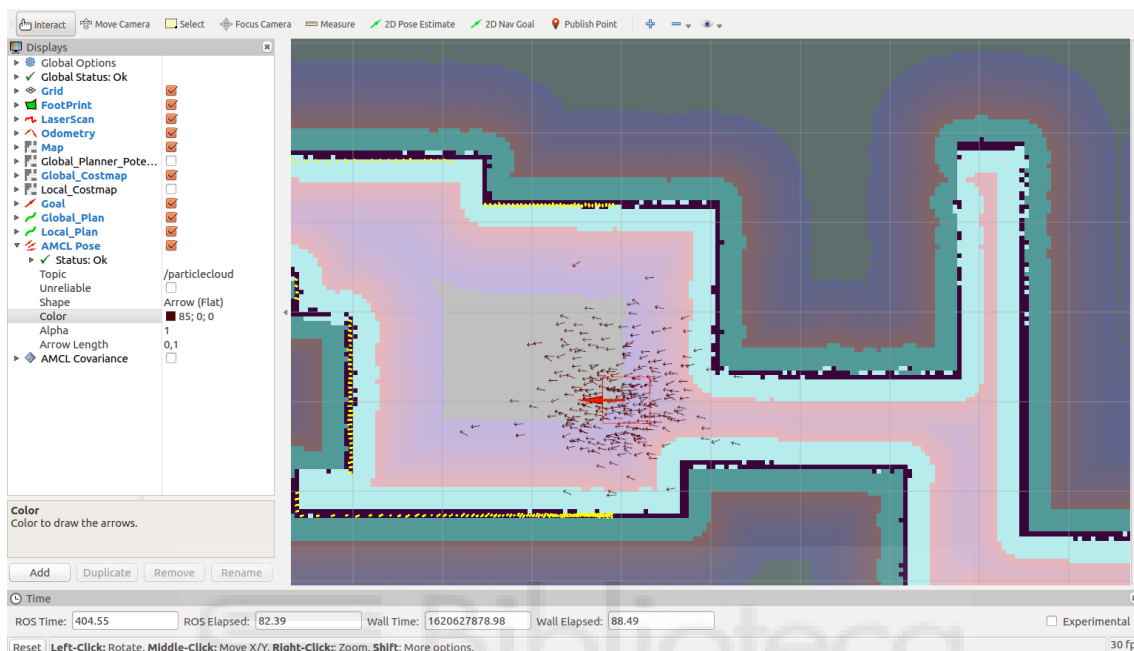


Figura 3.10: Ejemplo de partículas de *amcl* en RViz.

### 3.2.5. MOVIMIENTO DEL ROBOT

Para generar los comandos de velocidad, que guíen al robot a través del entorno de forma segura, se utiliza el paquete *move\_base*. Este paquete tiene como objetivo final, dado un punto de destino, generar los comandos de velocidad necesarios para alcanzar dicho punto.

Realizar esta tarea en un solo paquete sería muy complicado, por esa razón, el paquete *move\_base*, contiene distintos paquetes, los cuales realizan pequeñas tareas necesarias para alcanzar el objetivo final. Dichas tareas son las siguientes.

- Gestión de mapas de costes (*costmap\_2d*).
  - **Global\_costmap**: Mapa de costes global, empleado para planificación global.
  - **Local\_costmap**: Mapa de costes local, empleado para planificación local.

- Acciones de navegación (*nav\_core*).
  - **BaseGlobalPlanner**: Planificación de trayectoria global.
  - **BaseLocalPlanner**: Planificación y seguimiento de trayectoria local.
  - **RecoveryBehavior**: Recuperación de comportamiento.
- Integración de las acciones anteriores. (*move\_base*)

En primer lugar, se describen cada uno de los paquetes y nodos que realizan las tareas descritas anteriormente, y por último, se describe el paquete *move\_base*.

### 3.2.5.1. GESTIÓN DE MAPAS DE COSTES

Para la gestión de mapas de costes se emplea el paquete ***costmap\_2d***. Este paquete se encarga de crear un mapa de costes, partiendo del mapa de ocupación proporcionado y de los datos del sensor láser. Un mapa de costes aumenta el espacio ocupado por los obstáculos, realizando una reducción gradual del coste de las celdas ocupadas de un mapa. Este mapa describe una zona alrededor de las celdas ocupadas, en la cual el robot podría encontrarse en estado de colisión. Toda la información sobre este paquete se ha obtenido de [46].

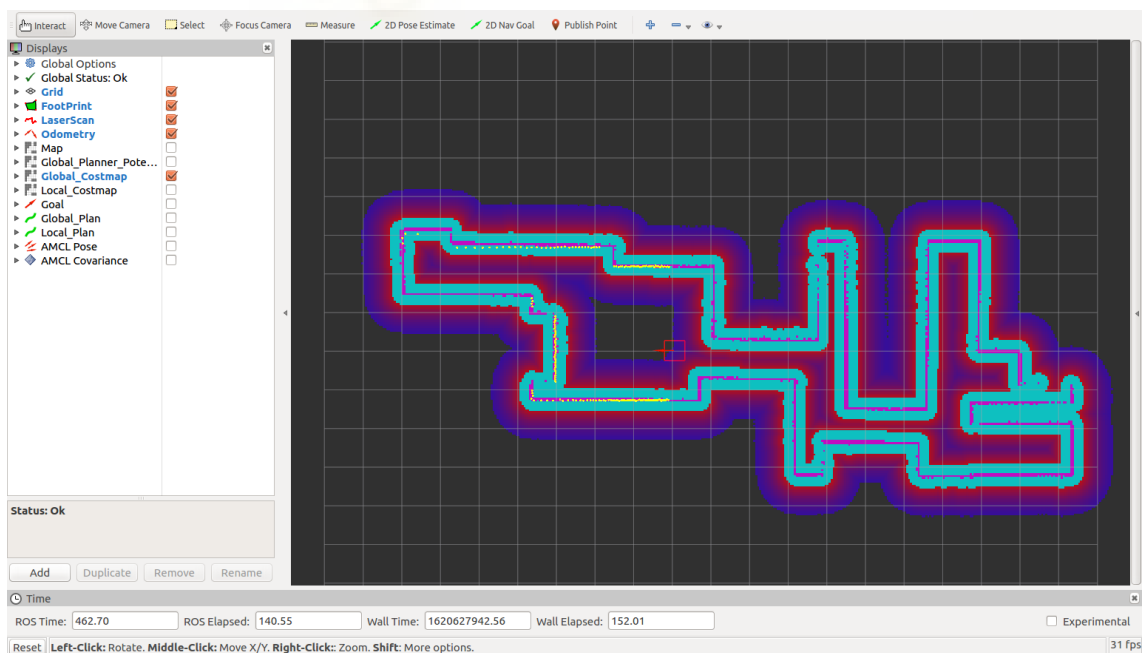


Figura 3.11: Ejemplo de mapa de costes global.

En la figura anterior, las celdas de color morado, representan los obstáculos del mapa, las celdas de color azul celeste, representan el aumento (también denominado inflación) de los obstáculos, las celdas rojas y azules representan la caída de los valores de las celdas, y el polígono rojo representa la forma del robot. Para una navegación segura del robot, el polígono rojo no debería de tocar nunca una celda roja, y el centro del polígono no debería cruzar nunca una celda azul.

En la siguiente figura se explica mejor el funcionamiento de la inflación de obstáculos.

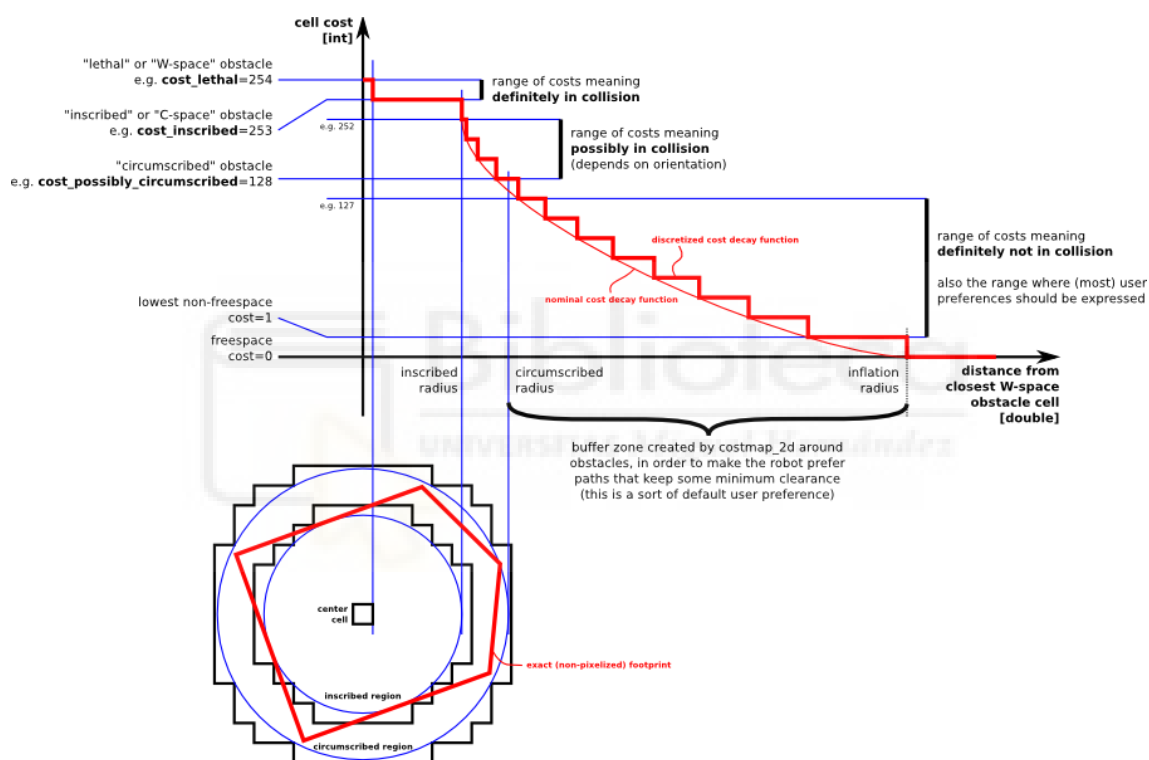


Figura 3.12: Gráfico de posibles estados del robot según la posición de su centro. Fuente: [46]

En el gráfico anterior, tomando como referencia el valor de la celda central del robot, la información que se obtiene es la siguiente:

- Si dicha celda tiene un valor entre 253 y 254, el robot está en colisión.
- Si tiene un valor de entre 128 y 252, el robot tiene probabilidad de estar en colisión.
- Con un valor por debajo de 127, se asegura que el robot no está en colisión.

Los rangos del valor de las celdas para cada uno de los posibles estados de colisión, dependen del tamaño del círculo inscrito en la forma del robot y del círculo que circunscribe al robot.

Cómo norma general se crean dos mapas de costes. Un mapa de costes global, constante, que se crea tomando sólo el mapa de ocupación proporcionado. Y un mapa de costes local que toma tanto el mapa de ocupación inicial como los datos del láser, para realizar la inflación de los obstáculos. El mapa de costes global (***global\_costmap***) se emplea para planificar la trayectoria global. Y el mapa de costes local (***local\_costmap***) se emplea para el seguimiento de trayectorias.

Una vez tenemos unos conocimientos básicos sobre qué es un mapa de costes, pasemos a explicar las características del paquete *costmap\_2d*.

### Tópicos

#### Suscripciones:

- ***/footprint***: contiene un mensaje del tipo *geometry\_msgs/Polygon*, donde se especifican puntos referidos a los vértices que conforman la forma del robot respecto de su centro.

#### Publicaciones:

- ***/costmap***: publica el mapa de costes generado.
- ***/costmap\_updates***: publica la actualización del mapa de costes para una determinada área.

### Parámetros

Cuando nos referimos a los parámetros de un *costmap* se pueden distinguir dos grupos, parámetros generales del *costmap*, y parámetros específicos de plugins. Existen tres posibles plugins para un *costmap*: ***static\_layer***, ***obstacle\_layer*** e ***inflation\_layer***.

- ***static\_layer***: parámetros de referencia para el mapa.



- ***obstacle\_layer***: parámetros de referencia para la detección de obstáculos.
- ***inflation\_layer***: parámetros de referencia para realizar la inflación del *costmap*.

### Parámetros generales

- ***global\_frame***: *frame* de referencia para *costmap*.
- ***robot\_base\_frame***: *frame* de la base del robot.
- ***transform\_tolerance***: diferencia de tiempo en segundos, máxima entre la publicación de una transformada y el tiempo actual.
- ***update\_frequency***: frecuencia a la que se actualiza el *costmap*.
- ***publish\_frequency***: frecuencia a la que se actualiza el *costmap* para visualización.
- ***static\_map***: valor booleano que indica si el mapa es estático o no.
- ***rolling\_window***: valor booleano que indica si el *costmap* se mueve junto con el *global\_frame*. Ha de ser opuesto al valor de *static\_map*.
- ***width***: anchura del *costmap* en metros.
- ***height***: altura del *costmap* en metros.
- ***resolution***: resolución del *costmap* en metros/celda.
- ***origin\_x***: origen en x del *costmap* respecto del *global\_frame*.
- ***origin\_y***: origen en y del *costmap* respecto del *global\_frame*.

Los últimos cinco parámetros, sólo tienen sentido para el *local\_costmap*, no siendo importantes en el *global\_costmap*.

### Parámetros de plugins

#### *Statick layer*

- ***unknown\_cost\_value***: valor de las celdas consideradas como desconocidas.
- ***lethal\_cost\_threshold***: valor de celda a partir del cual se considera que habría colisión.
- ***map\_topic***: tópico donde se publica el mapa.

- ***first\_map\_only***: se suscribe una única vez al mapa de ocupación publicado.
- ***subscribe\_to\_updates***: valor booleano que indica si se suscribe a actualizaciones del mapa.
- ***track\_unknown\_space***: valor booleano que permite o no navegar por celdas cuyo valor sea desconocido.
- ***trinary\_costmap***: valor booleano que puesto como verdadero, convierte el espectro de valores de celdas en tres posibles valores, desconocido, libre, u ocupado.

### **Obstacle layer**

- ***observation\_sources***: nombre arbitrario dado por el usuario para un sensor.  
Para cada *observation\_source* que se añade, tendrá los parámetros siguientes, y ellos empezaran con el nombre indicado para dicho *obserecation\_source*. Ej. `<source_name>/topic`.
- ***topic***: tópico donde se publican los datos de la *observation\_source*.
- ***sensor\_frame***: nombre del *frame* del cual se están tomando los datos.
- ***observation\_persistence***: durante cuánto tiempo mantener la lectura de un sensor. Un valor de 0, tomará el valor más reciente.
- ***expected\_update\_rate***: tiempo en segundos para el cual se espera una actualización de los datos publicados en el tópico suscrito. Si no se actualizan los datos a la frecuencia indicada el robot se detiene.
- ***data\_type***: tipo de mensaje que se recibe en el tópico sin el indicador *geometry\_msgs/*. Ej. para un láser se usa *LaserScan*.
- ***clearing***: valor booleano que indica si se debe usar o no, la *observation\_source* para limpiar valores del *costmap*.

- **marking:** valor booleano que indica si se debe usar o no, la *observation\_source* para colocar obstáculos en el *costmap*.
- **max\_obstacle\_height:** máxima altura en metros para la que una lectura del sensor se considera válida.
- **min\_obstacle\_height:** mínima altura en metros para la que una lectura del sensor se considera válida.
- **obstacle\_range:** máxima distancia en metros, a la que colocar obstáculos en el *costmap* usando los datos del sensor.
- **raytrace\_range:** máxima distancia en metros, a la que borrar obstáculos en el *costmap* usando los datos del sensor.
- **inf\_is\_valid:** valor booleano que permite o no, tomar infinitos valores del sensor.

#### Inflation layer

- **inflation\_radius:** distancia en metros, a la cual se inflarán los valores del coste de las celdas ocupadas.
- **cost\_scaling\_factor:** factor que define cuan agresivo es el descenso del valor de las celdas ocupadas hasta el valor de celda libre dentro del *inflation\_radius*.

En el capítulo de anexos de este trabajo podemos encontrar los distintos archivos de parámetros para la configuración del paquete *costmap\_2d*. En primer lugar tenemos el archivo "[costmap\\_common\\_params.yaml](#)", el cual se usa para configurar parámetros comunes tanto al *global\_costmap*, como al *local\_costmap*. En éste se describe el *footprint* del robot, que no es más que la forma del robot, especificando las coordenadas (x, y) de cada vértice de éste, con respecto a su centro. Además, se configura el parámetro *transform\_tolerance*.

Por otro lado tenemos el archivo "[global\\_costmap\\_params.yaml](#)", el cual se emplea para configurar todos los parámetros referentes al *global\_costmap*. Para el *global\_costmap* se tiene sólo en cuenta el mapa de ocupación inicial, por lo tanto, no se observan los datos del sensor.

Por último, tenemos el archivo “[local\\_costmap\\_params.yaml](#)”, que se utiliza para configurar todos los parámetros referentes al *local\_costmap*.

### 3.2.5.2. ACCIONES DE NAVEGACIÓN

Las acciones referentes a la navegación se realizan a través del paquete *nav\_core*. Como se ha comentado anteriormente, este paquete, está formado por tres componentes.

- ***BaseGlobalPlanner***: Planificación de trayectoria global.
- ***BaseLocalPlanner***: Planificación y seguimiento de trayectoria local.
- ***RecoveryBehavior***: Recuperación de comportamiento.

A continuación, se describen cada uno de estos componentes.

#### ***BaseGlobalPlanner***

Proporciona una interfaz para trabajar con planificadores globales. Los planificadores globales se añadirán como plugins al paquete *move\_base*. Existen tres planificadores que utilizan la interfaz de *BaseGlobalPlanner*.

- ***global\_planner***: planificador global diseñado para reemplazar a *navfn*.
- ***navfn***: planificador global basado en cuadrículas.
- ***carrot\_planner***: simple planificador global que intenta mover el robot lo más cerca posible del punto de destino.

#### ***BaseLocalPlanner***

Proporciona una interfaz para trabajar con planificadores locales. Estos planificadores se añadirán como plugins al paquete *move\_base*. Existen cinco planificadores que utilizan la interfaz *BaseLocalPlanner*.

- ***base\_local\_planner***: implementa el algoritmo *DWA (Dynamic Window Approach)* junto con un algoritmo de seguimiento de trayectorias para generar los comandos de velocidad enviados a la base del robot.

- ***dwa\_local\_planner***: implementa el algoritmo *DWA (Dynamic Window Approach)* para generar los comandos de velocidad enviados a la base del robot. Es más adecuado para robots holonómicos que *base\_local\_planner*.
- ***eband\_local\_planner***: implementa un algoritmo de banda elástica.
- ***teb\_local\_planner***: implementa un algoritmo de banda elástica en función del tiempo para la optimización de la trayectoria en línea.
- ***mpc\_local\_planner***: proporciona varios modelos de control predictivos.

### **RecoveryBehavior**

Proporciona una interfaz para trabajar con la recuperación del comportamiento de navegación. Estos recuperadores del comportamiento se añadirán como plugins al paquete *move\_base*. Existen dos recuperadores del comportamiento que utilizan la interfaz *RecoveryBehavior*. Su función es la de iniciarse cuando el sistema de navegación detecta que el robot está atrapado.

- ***clear\_costmap\_recovery***: convierte los *costmaps* al mapa estático a partir de una cierta distancia respecto del centro del robot.
- ***rotate\_recovery***: provoca un giro de 360° del robot, intentando borrar información errónea de los *costmaps*.

Para este trabajo, se ha elegido como planificador global, el plugin ***global\_planner***, y como planificador local, el plugin ***base\_local\_planner***. Además, se emplea tanto ***clear\_costmap\_recovery*** como ***rotate\_recovery***. A continuación, pasamos a describir en profundidad cada uno de estos paquetes.

### global planner

Este planificador global, tiene distintos comportamientos según la configuración de sus parámetros. A continuación, se muestran ejemplos de planificaciones.

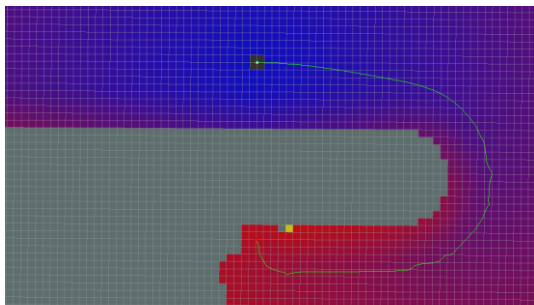


Figura 3.16: Comportamiento por defecto. Fuente: [58]

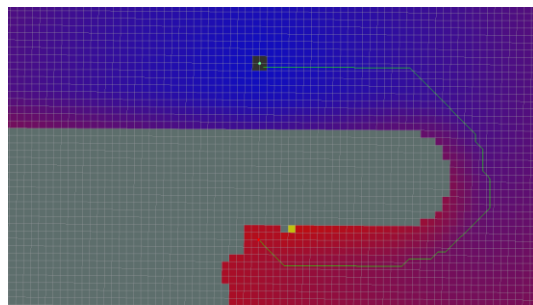


Figura 3.15: Planificador por cuadrículas. Fuente: [58]

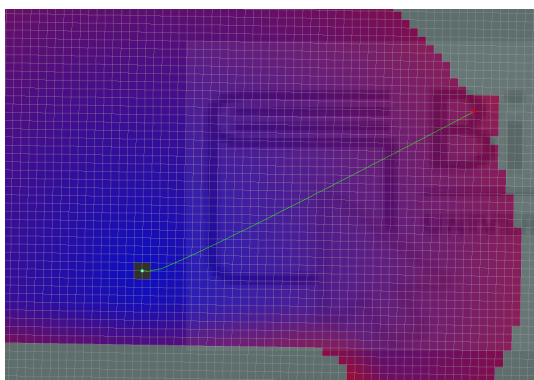


Figura 3.14: Dijkstra. Fuente: [58]

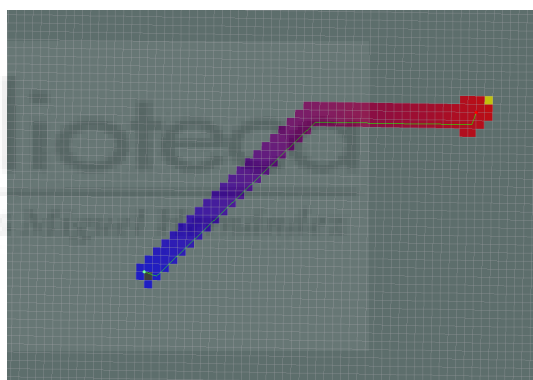


Figura 3.13: A\*. Fuente: [58]

### Tópicos

#### Suscripciones:

No realiza ninguna suscripción.

#### Publicaciones:

- **/plan:** publica el camino a seguir por el robot.

### Parámetros

- **allow\_unknown:** valor booleano que permite o no, planificar trayectorias a través de celdas del mapa cuyo estado se desconoce.

- **default\_tolerance:** tolerancia de la trayectoria al punto de destino.
- **visualize\_potential:** valor booleano que publica o no, el área de cómputo del planificador.
- **use\_dijkstra:** valor booleano que indica si se usa o no, el algoritmo *Dijkstra* para generar la trayectoria. En caso de falso, se usa el algoritmo *A\**.
- **use\_quadratic:** valor booleano que indica si se usa o no, una aproximación cuadrática.
- **use\_grid\_path:** valor booleano que indica si se usa o no, una planificación por cuadrículas.
- **old\_navfn\_behavior:** valor booleano que indica si se usa o no, el comportamiento del planificador *navfn*.
- **lethal\_cost:** valor para el cual una celda se considera ocupada.
- **neutral\_cost:** valor para el cual una celda se considera libre.
- **cost\_factor:** valor que multiplica el coste de las celdas del *costmap*.
- **publish\_potential:** valor booleano que indica si se publica o no, un *costmap* potencial.
- **orientation\_mode:** valor que configura la orientación en cada punto de la trayectoria.
  - 0: no se añaden orientaciones en los puntos de la trayectoria, sólo en el punto final.
  - 1: orientación hacia adelante, excepto orientación del punto final.
  - 2: orientación entre el punto inicial y final.
  - 3: orientación hacia adelante hasta el último tramo.
  - 4: orientación hacia atrás, excepto orientación del punto final.
  - 5: orientación hacia la izquierda, excepto orientación del punto final.
  - 6: orientación hacia la derecha, excepto orientación del punto final.

- **outline\_map:** valor booleano que indica si se describe o no, el *global\_costmap* con obstáculos letales.
- **orientation\_window\_size:** valor que indica la distancia a la cual calcular la orientación de cada punto según *orientation\_mode*.

Todos estos parámetros se recogen en el archivo “[global\\_planner\\_params.yaml](#)”, que se encuentra en el capítulo de anexos de este trabajo.

### **base local planner**

Este paquete, proporciona un planificador de trayectorias a nivel local y un seguidor para dichas trayectorias. La trayectoria planificada a nivel local se realiza tomando puntos de la trayectoria global como objetivo. La trayectoria a nivel local se actualiza constantemente a medida que el robot avanza.

Para planificar la trayectoria local, implementa el algoritmo *DWA (Dynamic Window Approach)*. Este algoritmo consta de los siguientes pasos.

- Genera muestras de velocidad para cada una de sus componentes (x, y, theta).
- Simula el comportamiento del robot al aplicar cada una de las muestras de velocidad durante un tiempo específico.
- Asigna una puntuación a cada una de las trayectorias simuladas en el apartado anterior.
- Toma la trayectoria con mayor puntuación y envía los comandos de velocidad a la base del robot para seguir dicha trayectoria.
- Repetir el proceso anterior.

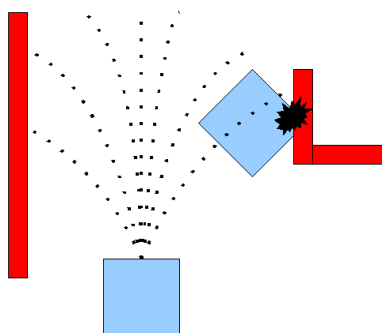


Figura 3.17: Ejemplo de simulación de trayectorias. Fuente: [59].



Pasemos ahora a hablar sobre los tópicos y parámetros de este paquete.

## Tópicos

### Suscripciones

- **/odom:** se suscribe a la odometría para obtener los valores de velocidad del robot en cada instante.

### Publicaciones

- **/global\_plan:** porción de la trayectoria global que se está intentando seguir.
- **/local\_plan:** trayectoria local con mejor puntuación.
- **/cost\_cloud:** cuadrícula de costes usada para puntuar las trayectorias.

## Parámetros

Los parámetros del *base\_local\_planner* se pueden dividir en varios grupos:

### Parámetros del robot

- **acc\_lim\_x:** aceleración máxima en x (m/s<sup>2</sup>).
- **acc\_lim\_y:** aceleración máxima en y (m/s<sup>2</sup>).
- **acc\_lim\_theta:** aceleración máxima en theta (rad/s<sup>2</sup>).
- **max\_vel\_x:** velocidad máxima en x (m/s).
- **min\_vel\_x:** velocidad mínima en x (m/s).
- **max\_vel\_y:** velocidad máxima en y (m/s).
- **min\_vel\_y:** velocidad mínima en y (m/s).
- **max\_vel\_theta:** velocidad angular máxima en theta (rad/s).
- **min\_vel\_theta:** velocidad angular mínima en theta (rad/s).
- **min\_in\_place\_vel\_theta:** velocidad angular mínima para una rotación sobre sí mismo (rad/s).
- **holonomic\_robot:** valor booleano que especifica si el robot es holonómico o no.

#### Parámetros de tolerancia al destino

- ***yaw\_goal\_tolerance***: tolerancia a la orientación del destino (rad).
- ***xy\_goal\_tolerance***: tolerancia de distancia al destino (m).
- ***latch\_xy\_goal\_tolerance***: valor booleano que permite una rotación sobre sí mismo una vez llegado al punto de destino, para alcanzar la orientación deseada.

#### Parámetros de simulación

- ***sim\_time***: tiempo de simulación para el DWA.
- ***sim\_granularity***: distancia entre puntos de la trayectoria global.
- ***vx\_samples***: número de muestras de velocidad lineal con las que simular las posibles trayectorias.
- ***vtheta\_samples***: número de muestras de velocidad angular con las que simular las posibles trayectorias.
- ***controller\_frequency***: frecuencia a la que se actualizan los comandos de velocidad.

#### Parámetros de puntuación de trayectorias

- ***meter\_scoring***: valor booleano que indica si los valores de fidelidad siguientes se expresan en metros o en celdas. True=metros, false=celdas.
- ***pdist\_scale***: fidelidad a la trayectoria global en rango (0-5).
- ***gdist\_scale***: fidelidad al destino local en rango (0-5).
- ***occdist\_scale***: peso para intentar esquivar objetos. Colocar valores muy pequeños, menores a 0.1, o el sistema se detendrá a la hora de pasar por zonas estrechas.

#### Parámetros para prevenir la oscilación

- ***oscillation\_reset\_dist***: distancia que ha de recorrer el robot antes de que se resetee la oscilación.

### Parámetros referentes a la trayectoria global

- ***prune\_plan***: valor booleano que, indica si se borra o no la trayectoria planeada a medida que el robot la sigue.

Estos parámetros se encuentran recogidos en el archivo "[base\\_local\\_planner\\_params.yaml](#)" en el capítulo de anexos de este trabajo.

### recovery behaviors

En este apartado se describen los dos recuperadores de comportamiento mencionados, ***clear\_costmap\_recovery***, y ***rotate\_recovery***. La información sobre su uso, y funcionamiento se ha extraído de las referencias [47] y [48] respectivamente.

move\_base Default Recovery Behaviors

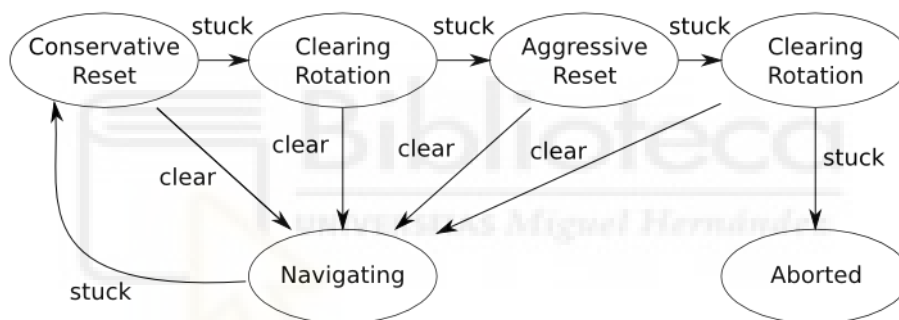


Figura 3.18: Diagrama de flujo *recovery\_behaviors* por defecto. Fuente: [60]

En el diagrama anterior se muestra el flujo que sigue el sistema de navegación cuando detecta que el robot está atascado.

- Primero, elimina los obstáculos del *costmap* a partir de una distancia conservadora.
- Segundo, intenta realizar una rotación sobre sí mismo para actualizar el espacio a su alrededor.
- Si el robot sigue atascado se realizará otro borrado del *costmap*, pero esta vez con un área de borrado mayor.
- Por último, intenta realizar una rotación sobre sí mismo para actualizar el espacio a su alrededor.

- Si el robot sigue atascado detendrá las acciones y lo notificará al usuario.

A continuación, se comentan los parámetros que definen cada comportamiento:

#### **clear costmap recovery**

- **reset\_distance**: longitud del lado de un cuadrado centrado en el robot, fuera del cual se eliminan los obstáculos del *costmap*.

#### **rotate recovery**

- **sim\_granularity**: cada cuanto comprobar si hay obstáculos (rad).

Los parámetros de ambos comportamientos se describen en el archivo "[recovery\\_behaviors\\_params.yaml](#)", que se encuentra en el apartado de anexos de este trabajo.

### **3.2.5.3. INTEGRACIÓN DE LAS ANTERIORES**

Para una mayor abstracción del sistema, se emplea el paquete **move\_base**, que se encarga de integrar los paquetes anteriores, de tal manera, que el usuario solo tenga que indicar un punto de destino y a partir de dicho punto, el sistema se encarga de planificar una trayectoria segura y generar los comandos de velocidad necesarios para seguir dicha trayectoria. Este paquete implementa también, mediante el paquete *actionlib*, un servicio de servidor y cliente. A continuación, se comentan los tópicos referentes a *actionlib*, pero para nuestro caso no se han empleado.

#### **Tópicos actionlib**

##### Suscripciones

- **move\_base/goal**: indica el punto de destino deseado.
- **move\_base/cancel**: permite cancelar un destino indicando publicando en este tópico el identificador de ese destino.

### Publicaciones

- ***move\_base/feedback***: publica la información de la posición del robot respecto del mapa.
- ***move\_base/status***: publica información sobre el estado actual del robot.
- ***move\_base/result***: este tópico no se emplea.

Al igual que cualquier paquete, el paquete *move\_base* trabaja con unos determinados tópicos y parámetros que veremos a continuación.

### Tópicos

#### Suscripciones

- ***move\_base\_simple/goal***: recibe la posición y orientación del punto de destino para el robot.

#### Tópicos

- ***cmd\_vel***: publica los comandos de velocidad necesarios para alcanzar el punto objetivo indicado en el tópico anterior.

Como ya se ha comentado, *move\_base* simplifica mucho la tarea de navegación, necesitando solo de un punto objetivo y generando los comandos de velocidad necesarios a partir de este.

### Parámetros

- ***base\_global\_planner***: nombre del plugin a usar como planificador global. Opciones disponibles:
  - *global\_planner/GlobalPlanner*
  - *navfn/NavfnROS*
  - *carrot\_planner/CarrotPlanner*
- ***base\_local\_planner***: nombre del plugin a usar como planificador local. Opciones disponibles:
  - *base\_local\_planner::TrajectoryPlannerROS*
  - *dwa\_local\_planner::DWAPlaner*

- *eband\_local\_planner/EBandPlannerROS*
- ***recovery\_behaviors***: lista de los posibles tipos de recuperación del comportamiento.
- ***controller\_frequency***: frecuencia a la que se generan los comandos de velocidad.
- ***planner\_patience***: tiempo máximo que espera el sistema intentando encontrar una trayectoria al destino antes de ejecutar *recovery\_behaviors*.
- ***controller\_patience***: tiempo máximo que espera el sistema sin recibir comandos de velocidad antes de ejecutar *recovery\_behaviors*.
- ***conservative\_reset\_dist***: la distancia en metros, más allá del robot partir de la cual se eliminarán los obstáculos del *costmap*.
- ***recovery\_behavior\_enabled***: valor booleano que activa o desactiva la utilización de *recovery\_behaviors*.
- ***clearing\_rotation\_allowed***: valor booleano que activa o desactiva la utilización de *rotate\_recovery*.
- ***shut\_down\_costmaps***: valor booleano que determina si los *costmaps* permanecen desactivados cuando *move\_base* está inactivo.
- ***oscillation\_timeout***: tiempo en segundos que el sistema permite oscilación antes de ejecutar *recovery\_behaviors*.
- ***oscillation\_distance***: distancia que ha de recorrer el robot para que el sistema considere que no está oscilando.
- ***planner\_frequency***: frecuencia a la que se actualiza el planificador global.
- ***max\_planning\_retries***: veces que el sistema intenta realizar un plan global antes de iniciar *recovery\_behaviors*.

Los parámetros referentes al paquete *move\_base* se encuentran recogidos en el archivo "[move\\_base\\_params.yaml](#)". Además, para iniciar *move\_base* con todos los componentes que lo forman y sus respectivos parámetros se dispone del *launch\_file* "[move\\_base\\_p3at.launch](#)". Estos archivos se encuentran en el apartado de anexos de esta memoria.

### 3.3. NAVEGACIÓN EXTERIORES

Para la navegación en exteriores se podría emplear un sensor GPS. Para ello necesitaríamos un paquete que ejerciese la función de enlace entre el GPS y ROS. Para esta tarea se podría emplear, por ejemplo, el paquete ***nmea\_navsat\_driver***, este paquete dispone del nodo ***nmea\_serial\_driver***. Este nodo publica un tópicos (*fix*), del que se pueden extraer los valores de latitud, longitud y altitud.

En segundo lugar, se necesitaría crear un nodo que se suscribiese a dicho tópicos, y almacenase los valores de latitud, longitud y altitud.

Por otro lado, se podrían convertir las coordenadas GPS, a coordenadas UTM (*Universal Transversal de Mercator*), que expresan la posición en valores de X e Y, siendo X el ecuador, e Y el meridiano. Las coordenadas UTM se expresan en metros, siendo más sencilla su comprensión, y más fácil indicar puntos de destino.

Para convertir las coordenadas GPS a UTM, se puede emplear el paquete ***geonav\_transform***, que dispone del nodo ***geonav\_transform\_node***. Publicando los valores de latitud, longitud y altitud, en el formato *nav\_msgs/Odometry*, bajo el tópicos *nav\_odom*, este nodo nos devuelve las coordenadas de latitud, longitud y altitud en formato UTM a través del tópicos ***geonav\_utm***.

Una vez obtenidas las coordenadas UTM del robot en todo momento, se necesitaría desarrollar un nodo, capaz de generar comandos de velocidad, para describir una trayectoria partiendo de la posición actual del robot ( $x_0, y_0$ ), y el punto de destino ( $x_1, y_1$ ). El proceso podría ser el siguiente, calcular la distancia entre ( $x_0, y_0$ ) e ( $x_1, y_1$ ), obtener la orientación del vector que va de ( $x_0, y_0$ ) a ( $x_1, y_1$ ), aplicar una velocidad angular hasta alcanzar dicha orientación y por último, publicar una velocidad lineal, hasta que la distancia entre el punto actual, y el de destino, sea menor que una cierta tolerancia.

Los puntos de destino se podrían obtener mediante la selección de puntos en Google Maps, y transformando dichos puntos GPS a valores UTM con el nodo mencionado anteriormente.

## 4. SIMULACIÓN

En este capítulo se redacta el proceso empleado para crear un paquete con el cual realizar la navegación de un robot móvil dentro del entorno de simulación de Gazebo. También se detalla el uso de este paquete y los resultados obtenidos.

### 4.1. IMPLEMENTACIÓN

En este apartado se describen en detalle los pasos realizados para poder implementar el sistema de navegación desde cero.

#### 4.1.1. CREACIÓN DE UN NUEVO PAQUETE

En primer lugar, se ha creado el paquete *p3at\_simulation* que engloba todos los archivos necesarios para realizar la navegación del Pioneer-3AT dentro del entorno de simulación de Gazebo.

- Abrir terminal de Ubuntu.
- **`$ cd programas_ros/catkin_ws/src`**  
Nos movemos a la carpeta `/src` de nuestro *catkin\_workspace*.
- **`$ catkin_create_pkg p3at_simulation roscpp geometry_msgs tf move_base`**  
Creamos un paquete que se denomina *p3at\_simulation* e indicamos los paquetes de los que depende.
- **`$ cd programas_ros/catkin_ws`**
- **`$ catkin_make`**  
Compilamos el paquete.

Una vez compilado el paquete, por defecto sólo contiene las carpetas `/include` y `/src`. De forma manual se le han añadido las siguientes carpetas:

- ***/bagfiles***: carpeta empleada para almacenar los *bagfiles* generados durante los experimentos mediante el comando *rosvbag*.
- ***/launch***: carpeta dónde se encuentran todos los *launch\_files* disponibles en este paquete.



- **/libs:** carpeta que contiene las librerías externas que emplea este paquete. Esta carpeta no es necesaria para el *Navigation Stack*. Se emplea para incluir una librería que se usa en el seguidor de trayectorias realizado de forma propia.
- **/models:** esta carpeta contiene a su vez una serie de carpetas, que se emplean para almacenar los archivos que describen los mapas de costes, modelos 3D que describen la forma del robot, archivos URDF y los archivos “.world” que describen el entorno de Gazebo.
- **/params:** dentro de esta carpeta se encuentran los archivos de extensión “.yaml” que contienen los parámetros de configuración para cada nodo implicado en el sistema de navegación.



Figura 4.1: Aspecto del paquete *p3at\_simulation*.

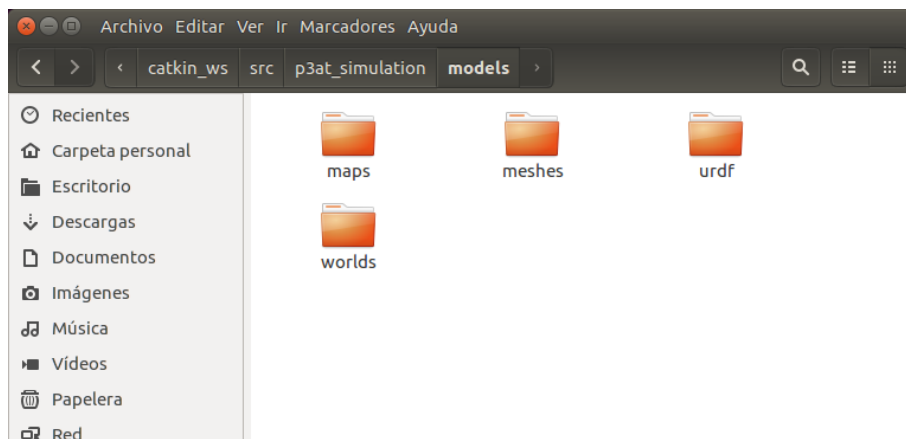


Figura 4.2: Carpeta *models* del paquete *p3at\_simulation*.

### 4.1.2. CREACIÓN DEL MODELO URDF

Una vez creado el paquete donde implementar nuestro proyecto, podemos comenzar a desarrollar el archivo URDF que describe nuestro robot.

En primer lugar, se ha descargado el paquete ***amr-ros-config***, que podemos encontrar en [49], que contiene archivos URDF y configuraciones para robots de Adept Mobile Robots (*amr*).

Dentro de este paquete, podemos encontrar la carpeta */description* y dentro de ella la carpeta */urdf*. De ese directorio se ha extraído el archivo *“pioneer3at.urdf”*, el cual se ha usado como base para crear el archivo *“my\_p3at.urdf”* que describe el robot usado en este trabajo.

El archivo *“my\_p3at.urdf”* se puede dividir en siete partes:

- **Primera parte:** se describen los elementos (*links*) principales del robot excepto sus ruedas.



```

1 <?xml version="1.0" ?>
2 <!-- ===== -->
3
4 <!-- ===== -->
5 robot name="pioneer3at" xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller" xmlns:interface="http://playerstage.sourceforge.net/gazebo/
6 <!-- For pointers on inertial and gazebo-related parameters see
7 * http://wiki.ros.org/urdf/Tutorials/Adding%20Physical%20and%20Collision%20Properties%20to%20a%20URDF%20Model
8 * http://answers.gazebosim.org/question/4372/the-inertia-matrix-explained/
9 * http://gazebosim.org/tutorials?tut=inertia&cat=build_robot_and
10 * http://gazebosim.org/tutorials?tut=ros_urdf
11 * http://en.wikipedia.org/wiki/List_of_moment_of_inertia_tensors
12 -->
13
14 <!-- ===== -->
15 <!-- PRIMERA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS (LINKS DEL ROBOT) -->
16 <!-- ===== -->
17
18 <!-- LINKS DESCRIPTIONS -->
19 <!-- Chassis -->
20 <link name="base_link">
21 <inertial>
22 <mass value="21.3"/>
23 <!-- P3AT_12 kg + x3 batteries_2 kg + PC_3kg - x4 wheels_1.4 kg - top_plate_0.1 kg -->
24 <origin xyz="0.1 0 0.177"/>
25 <inertia ixx="0.3338" ixy="0.0" ixz="0.0" iyy="0.4783" iyz="0.0" izz="0.3338"/>
26 </inertial>
27 <visual>
28 <origin rpy="0 0 0" xyz="0 0 0.177"/>
29 <geometry name="pioneer_geom">
30 <mesh filename="package://amr_robots_description/meshes/p3at_meshes/chassis.stl"/>
31 </geometry>
32 <material name="ChassisRed">
33 <color rgba="0.851 0.0 0.0 1.0"/>
34 </material>
35 </visual>
36 <collision>
37 <origin rpy="0 0 0" xyz="0 0 0.177"/>
38 <geometry>
39 <mesh filename="package://amr_robots_description/meshes/p3at_meshes/chassis.stl"/>
40 </geometry>
41 </collision>
42 </link>
43 <!-- Top Plate -->
44 <link name="top_plate">
45 <inertial>
46 <mass value="0.1"/>
47 <origin xyz="0.025 0 -0.223"/>
48 <inertia ixx="0.0001" ixy="0.0" ixz="0.0" iyy="0.0001" iyz="0.0" izz="0.0001"/>

```

Figura 4.3:Primera parte archivo my\_p3at.urdf.

- **Segunda parte:** se describen las uniones (transformadas) entre cada uno de los links anteriores.

```

132 <!-- SEGUNDA PARTE: DESCRIPCIÓN DE LAS UNIONES ENTRE ELEMENTOS DEL ROBOT (JOINTS) -->
133 <!-- SEGUNDA PARTE: DESCRIPCIÓN DE LAS UNIONES ENTRE ELEMENTOS DEL ROBOT (JOINTS) -->
134 <!-- SEGUNDA PARTE: DESCRIPCIÓN DE LAS UNIONES ENTRE ELEMENTOS DEL ROBOT (JOINTS) -->
135
136 <!-- JOINT DESCRIPTIONS -->
137 <!-- base_link + top_plate -->
138 <joint name="base_top_joint" type="fixed">
139   <origin rpy="0 0 0" xyz="0.003 0 0.274"/>
140   <parent link="base_link"/>
141   <child link="top_plate"/>
142 </joint>
143 <!-- base_link + front_sonar -->
144 <joint name="base_front_joint" type="fixed">
145   <origin rpy="0 0 0" xyz="0.193 0 0.25"/>
146   <parent link="base_link"/>
147   <child link="front_sonar"/>
148 </joint>
149 <!-- base_link + back_sonar -->
150 <joint name="base_back_joint" type="fixed">
151   <origin rpy="0 0 0" xyz="-0.187 0 0.247"/>
152   <parent link="base_link"/>
153   <child link="back_sonar"/>
154 </joint>
155 <!-- top_plate + case_laser -->
156 <joint name="top_laser_joint" type="fixed">
157   <origin rpy="0 0 0" xyz="0.127 0 0.0975"/>
158   <parent link="top_plate"/>
159   <child link="case_laser"/>
160 </joint>
161 <!-- case_laser + base_laser -->
162 <joint name="case_base_laser_joint" type="fixed">
163   <origin rpy="0 0 0" xyz="0.025 0 -0.03"/>
164   <parent link="case_laser"/>
165   <child link="laser"/>
166 </joint>
167
168 <!-- TERCERA PARTE: DESCRIPCIÓN DE LOS MATERIALES DE CADA ELEMENTO PARA GAZEBO -->
169 <!-- TERCERA PARTE: DESCRIPCIÓN DE LOS MATERIALES DE CADA ELEMENTO PARA GAZEBO -->
170 <!-- TERCERA PARTE: DESCRIPCIÓN DE LOS MATERIALES DE CADA ELEMENTO PARA GAZEBO -->
171
172 <!-- GAZEBO MATERIAL REFERENCES -->
173 <!-- base_link -->
174 <gazebo reference="base_link">
175   <material value="Gazebo/Red"/>
176 </gazebo>
177 <!-- top_plate -->
178 <gazebo reference="top_plate">
179   <material value="Gazebo/Black"/>
180 </gazebo>
181 <!-- front_sonar -->
182 <gazebo reference="front_sonar">
183   <material value="Gazebo/Yellow"/>
184 </gazebo>
185 <!-- back_sonar -->
186 <gazebo reference="back_sonar">
187   <material value="Gazebo/Yellow"/>
188 </gazebo>
189 <!-- case_laser -->
190 <gazebo reference="case_laser">
191   <material value="Gazebo/Blue"/>
192 </gazebo>
193 <!-- base_laser -->
194 <gazebo reference="laser">
195   <material value="Gazebo/Black"/>
196 </gazebo>
197 <!-- usb_cam -->
198 <gazebo reference="usb_cam">
199   <material value="Gazebo/Orange"/>
200 </gazebo>
201
202
203 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->
204 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->
205 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->

```

Figura 4.5: Segunda parte archivo my\_p3at.urdf

- **Tercera parte:** se describen los materiales de Gazebo asociados a cada elemento.

```

161 <!-- case_laser + base_laser -->
162 <joint name="case_base_laser_joint" type="fixed">
163   <origin rpy="0 0 0" xyz="0.025 0 -0.03"/>
164   <parent link="case_laser"/>
165   <child link="laser"/>
166 </joint>
167
168 <!-- TERCERA PARTE: DESCRIPCIÓN DE LOS MATERIALES DE CADA ELEMENTO PARA GAZEBO -->
169 <!-- TERCERA PARTE: DESCRIPCIÓN DE LOS MATERIALES DE CADA ELEMENTO PARA GAZEBO -->
170 <!-- TERCERA PARTE: DESCRIPCIÓN DE LOS MATERIALES DE CADA ELEMENTO PARA GAZEBO -->
171
172 <!-- GAZEBO MATERIAL REFERENCES -->
173 <!-- base_link -->
174 <gazebo reference="base_link">
175   <material value="Gazebo/Red"/>
176 </gazebo>
177 <!-- top_plate -->
178 <gazebo reference="top_plate">
179   <material value="Gazebo/Black"/>
180 </gazebo>
181 <!-- front_sonar -->
182 <gazebo reference="front_sonar">
183   <material value="Gazebo/Yellow"/>
184 </gazebo>
185 <!-- back_sonar -->
186 <gazebo reference="back_sonar">
187   <material value="Gazebo/Yellow"/>
188 </gazebo>
189 <!-- case_laser -->
190 <gazebo reference="case_laser">
191   <material value="Gazebo/Blue"/>
192 </gazebo>
193 <!-- base_laser -->
194 <gazebo reference="laser">
195   <material value="Gazebo/Black"/>
196 </gazebo>
197 <!-- usb_cam -->
198 <gazebo reference="usb_cam">
199   <material value="Gazebo/Orange"/>
200 </gazebo>
201
202
203 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->
204 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->
205 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->

```

Figura 4.6: Tercera parte del archivo my\_p3at.urdf

- **Cuarta parte:** se describen tanto los elementos asociados a las ruedas, como sus uniones y sus materiales de referencia en Gazebo. Primero se describen los elementos de la izquierda, y luego los de la derecha.

```

202
203 <!-- ===== -->
204 <!-- CUARTA PARTE: DESCRIPCIÓN DE LOS ELEMENTOS RUEDAS, EJES Y ESPACIADORES -->
205 <!-- ===== -->
206
207 <!-- ===== -->
208 <!-- ELEMENTOS DE LA IZQUIERDA -->
209 <!-- ===== -->
210
211 <!--LINKS DESCRIPTIONS-->
212 <!-- front_axles -->
213 <link name="p3at_front_left_axle">
214   <inertial>
215     <mass value="0.1"/>
216     <origin xyz="0 0 0"/>
217     <inertia lxx="1.0" lxy="0.0" lxz="0.0" lyy="1.0" lyz="0.0" lzz="1.0"/>
218   </inertial>
219   <visual>
220     <origin rpy="0 0 0" xyz="0 0 0"/>
221     <geometry name="pioneer_geom">
222       <mesh filename="package://anr_robots_description/meshes/p3at_meshes/axle.stl"/>
223     </geometry>
224     <material name="AxleGrey">
225       <color rgba="0.5 0.5 0.5 1"/>
226     </material>
227   </visual>
228 </link>
229 <!-- front_hubs -->
230 <link name="p3at_front_left_hub">
231   <inertial>
232     <mass value="0.1"/>
233     <origin xyz="0 0 0"/>
234     <inertia lxx="1.0" lxy="0.0" lxz="0.0" lyy="1.0" lyz="0.0" lzz="1.0"/>
235   </inertial>
236   <visual>
237     <origin rpy="0 0 0" xyz="0 0 0"/>
238     <geometry name="pioneer_geom">
239       <mesh filename="package://anr_robots_description/meshes/p3at_meshes/left_hubcap.stl"/>
240     </geometry>
241     <material name="HubcapYellow">
242       <color rgba="1.0 0.811 0.151 1.0"/>
243     </material>
244   </visual>
245 </link>
246 <!-- front_wheels -->
247 <link name="p3at_front_left_wheel">
248   <inertial>
249     <mass value="1.2"/>

```

Figura 4.7: Cuarta parte del archivo my\_p3at.urdf

- **Quinta parte:** se describe el láser plugin de Gazebo con los valores correspondientes al láser Sick-LMS200.

```

614 <!-- ===== -->
615 <!-- QUINTA PARTE: DESCRIPCIÓN DEL LÁSER PLUGIN PARA GAZEBO -->
616 <!-- ===== -->
617
618 <!-- RAY SENSOR FOR GAZEBO -->
619 <gazebo reference="Laser">
620   <sensor name="stick_lms200" type="gpu_ray">
621     <pose>0 0 0 0 0 0</pose>
622     <visualize>true</visualize>
623     <update_rate>75</update_rate>
624     <ray>
625       <scan>
626         <horizontal>
627           <samples>180</samples>
628           <resolution>1</resolution>
629           <min_angle>-1.570796</min_angle>
630           <max_angle>1.570796</max_angle>
631         </horizontal>
632       </scan>
633       <range>
634         <min>0.1</min>
635         <max>30.0</max>
636         <resolution>0.01</resolution>
637       </range>
638       <noise>
639         <type>gaussian</type>
640         <!-- Noise parameters based on published spec for Hokuyo laser
641             achieving "+-30mm" accuracy at range < 10m. A mean of 0.0m and
642             stddev of 0.01m will put 99.7% of samples within 0.03m of the true
643             reading. -->
644         <mean>0.00</mean>
645         <stddev>0.01</stddev>
646       </noise>
647     </ray>
648     <plugin filename="libgazebo_ros_gpu_laser.so" name="gazebo_ros_head_hokuyo_controller">
649       <topicName>/sim/scan</topicName>
650       <frameName>Laser</frameName>
651     </plugin>
652   </sensor>
653 </gazebo>
654

```

Figura 4.8: Quinta parte del archivo my\_p3at.urdf

- **Sexta parte:** se describen las propiedades dinámicas de las ruedas.

```

664
665 <!-- ===== -->
666 <!-- SEXTA PARTE: DESCRIPCIÓN DE LAS PROPIEDADES DINÁMICAS DE LAS RUEDAS -->
667 <!-- ===== -->
668
669 <!-- PROPERTIES OF LEFT WHEEL "MOTORS"-->
670 <!-- see http://gazebo.m.org/tutorials/?tut=ros\_urdf -->
671 <gazebo reference="p3at_back_left_wheel">
672 <kp> 1000000.0 </kp>
673 <!-- kp and kd for rubber -->
674 <kd> 100.0 </kd>
675 <mu1> 1.0 </mu1>
676 <!-- was 10 -->
677 <mu2> 1.0 </mu2>
678 <!-- how to get these into <surface><friction><ode>... ? slp1 0.5 /slp1 slp2 0 /slp2 -->
679 <!-- fdir1 0 1 0 /fdir1 -->
680 <!-- see http://github.com/MobileRobots/anr-ros-config/issues/6 -->
681 <maxVel> 0.00 </maxVel>
682 <minDepth> 0.001 </minDepth>
683 <material values="Gazebo/Black"/>
684 </gazebo>
685 <gazebo reference="p3at_front_left_wheel">
686 <kp> 1000000.0 </kp>
687 <!-- kp and kd for rubber -->
688 <kd> 100.0 </kd>
689 <mu1> 1.0 </mu1>
690 <!-- was 10 -->
691 <mu2> 1.0 </mu2>
692 <!-- how to get these into <surface><friction><ode>... ? slp1 0.5 /slp1 slp2 0 /slp2 -->
693 <!-- fdir1 0 1 0 /fdir1 -->
694 <!-- see http://github.com/MobileRobots/anr-ros-config/issues/6 -->
695 <maxVel> 0.00 </maxVel>
696 <minDepth> 0.001 </minDepth>
697 <material values="Gazebo/Black"/>
698 </gazebo>
699
700 <!-- PROPERTIES OF RIGHT WHEEL "MOTORS"-->
701 <!-- see http://gazebo.m.org/tutorials/?tut=ros\_urdf -->
702 <gazebo reference="p3at_back_right_wheel">
703 <kp> 1000000.0 </kp>
704 <!-- kp and kd for rubber -->
705 <kd> 100.0 </kd>
706 <mu1> 1.0 </mu1>
707 <!-- was 10 -->
708 <mu2> 1.0 </mu2>
709 <!-- how to get these into <surface><friction><ode>... ? slp1 0.5 /slp1 slp2 0 /slp2 -->
710 <!-- fdir1 0 1 0 /fdir1 -->

```

Figura 4.9: Sexta parte del archivo my\_p3at.urdf

- **Séptima parte:** se describe el plugin `skid_steer_drive` para Gazebo, con los valores que describen al Pioneer-3at.

```

708 <!-- was 10 -->
709 <mu2> 1.0 </mu2>
710 <!-- how to get these into <surface><friction><ode>... ? slp1 0.5 /slp1 slp2 0 /slp2 -->
711 <!-- fdir1 0 1 0 /fdir1 -->
712 <!-- see http://github.com/MobileRobots/anr-ros-config/issues/6 -->
713 <maxVel> 0.00 </maxVel>
714 <minDepth> 0.001 </minDepth>
715 <material values="Gazebo/Black"/>
716 </gazebo>
717 <gazebo reference="p3at_front_right_wheel">
718 <kp> 1000000.0 </kp>
719 <!-- kp and kd for rubber -->
720 <kd> 100.0 </kd>
721 <mu1> 1.0 </mu1>
722 <!-- was 10 -->
723 <mu2> 1.0 </mu2>
724 <!-- how to get these into <surface><friction><ode>... ? slp1 0.5 /slp1 slp2 0 /slp2 -->
725 <!-- fdir1 0 1 0 /fdir1 -->
726 <!-- see http://github.com/MobileRobots/anr-ros-config/issues/6 -->
727 <maxVel> 0.00 </maxVel>
728 <minDepth> 0.001 </minDepth>
729 <material values="Gazebo/Black"/>
730 </gazebo>
731
732 <!-- ===== -->
733 <!-- SÉPTIMA PARTE: DESCRIPCIÓN DEL SKID_STEER_DRIVE PLUGIN PARA GAZEBO -->
734 <!-- ===== -->
735
736 <gazebo>
737 <plugin filename="libgazebo_ros_skid_steer_drive.so" name="skid_steer_drive_controller">
738 <updateRate> 100.0 </updateRate>
739 <robotNamespace> sim </robotNamespace>
740 <leftFrontJoint> p3at_front_left_wheel_joint </leftFrontJoint>
741 <rightFrontJoint> p3at_front_right_wheel_joint </rightFrontJoint>
742 <leftRearJoint> p3at_back_left_wheel_joint </leftRearJoint>
743 <rightRearJoint> p3at_back_right_wheel_joint </rightRearJoint>
744 <wheelSeparation> 0.4 </wheelSeparation>
745 <wheelDiameter> 0.215 </wheelDiameter>
746 <robotBaseFrame> base_link </robotBaseFrame>
747 <maxForce> 5 </maxForce>
748 <torque> 15 </torque>
749 <commandTopic> cmd_vel </commandTopic>
750 <odometryTopic> odon </odometryTopic>
751 <odometryFrame> odon </odometryFrame>
752 <broadcastTF> 1 </broadcastTF>
753 </plugin>
754 </gazebo>
755 </robot>

```

Figura 4.10: Séptima parte del archivo my\_p3at.urdf

Para más información sobre la estructura y sintaxis de los archivos URDF se proporciona la referencia [35]. Además, los plugins disponibles en Gazebo y como utilizarlos se pueden encontrar en [50].

### 4.1.3. CREACIÓN DEL ENTORNO DE GAZEBO

Para crear el entorno de gazebo se ha seguido el procedimiento descrito en la sección [2.2.3. Gazebo](#) de este trabajo y se ha obtenido el archivo “*real\_lab.world*” que describe el siguiente entorno:

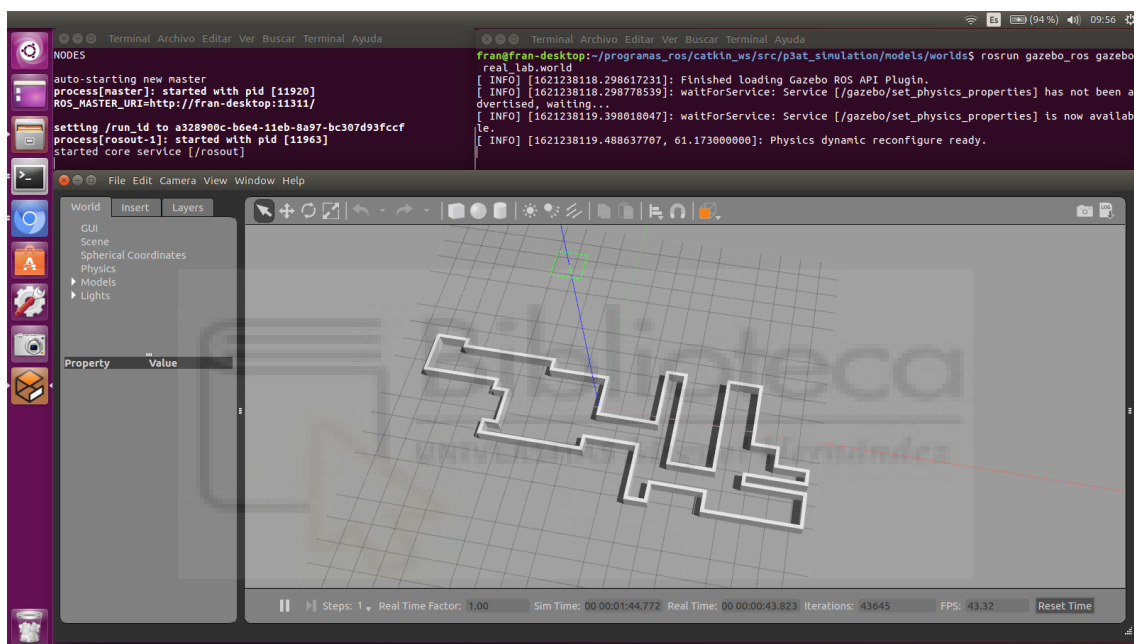


Figura 4.11: Aspecto del entorno descrito por el archivo *real\_lab.world*.

#### 4.1.4. PARÁMETROS DEL NAVIGATION STACK

Los parámetros que configuran el funcionamiento de los nodos del *Navigation Stack* se han descrito anteriormente y se encuentran en el apartado de anexos de este trabajo. Para su configuración se ha seguido la guía disponible en [51].

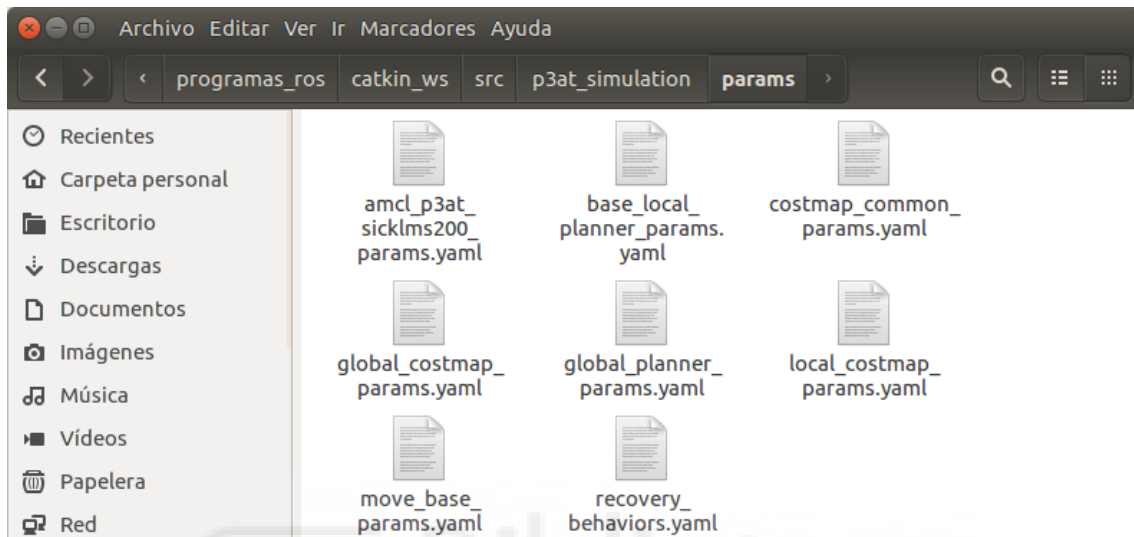


Figura 4.12: Archivos de parámetros disponibles en el paquete *p3at\_simulation*.

#### 4.1.5. LAUNCH FILES

El paquete *p3at\_simulation* contiene dentro de la carpeta */launch* los siguientes *launch\_files*.

- **[robot\\_p3at\\_empty\\_world\\_simulated](#)**: lanza Gazebo junto con el modelo “*my\_p3at.urdf*” en un entorno vacío y publica sus transformadas.
- **[robot\\_p3at\\_lab\\_world\\_simulated](#)**: lanza la simulación con el modelo de robot “*my\_p3at.urdf*” en el entorno de Gazebo “*real\_lab.world*” y publica todas las transformadas del robot.
- **[amcl\\_p3at](#)**: inicia el nodo *amcl* con sus respectivos parámetros.
- **[map\\_server](#)**: inicia el nodo *map\_server* con sus respectivos parámetros.
- **[move\\_base\\_p3at](#)**: inicia el nodo *move\_base* con sus respectivos parámetros.
- **[navigation\\_p3at](#)**: inicia simultáneamente los *launch\_files* *amcl\_p3at*, *map\_server* y *move\_base\_p3at*.

Todos los *launch\_files* se pueden consultar en el apartado de anexos de esta memoria.

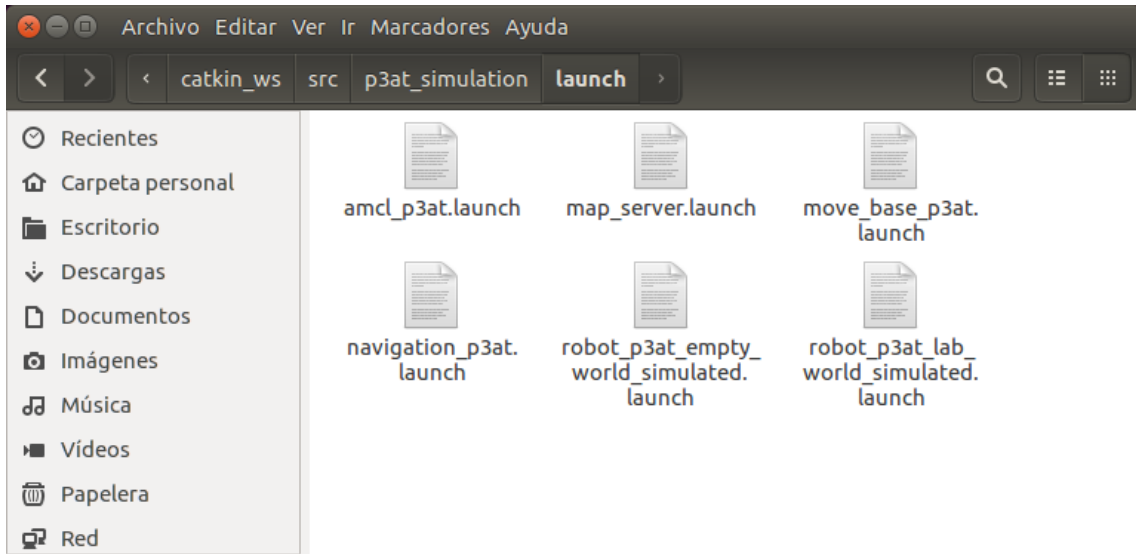


Figura 4.13: Contenido carpeta /launch del paquete *p3at\_simulation*.

## 4.2. USO

En esta sección se detalla el uso del paquete *p3at\_simulation*.

En primer lugar, se ha de lanzar la simulación del Pioneer-3AT, para ello se utiliza el archivo “*robot\_p3at\_lab\_world\_simulated.launch*”.

- Abrir terminal de Ubuntu.
- **`$ roslaunch p3at_simulation robot_p3at_lab_world_simulated.launch`**

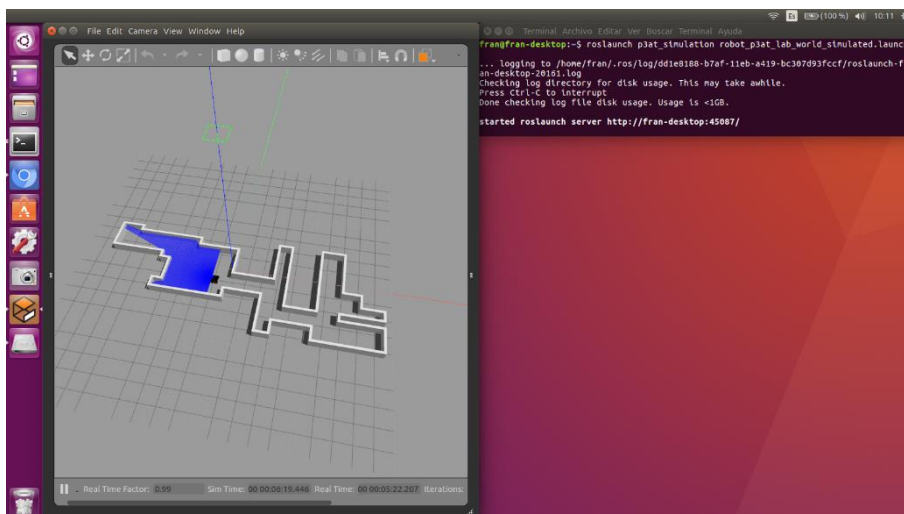
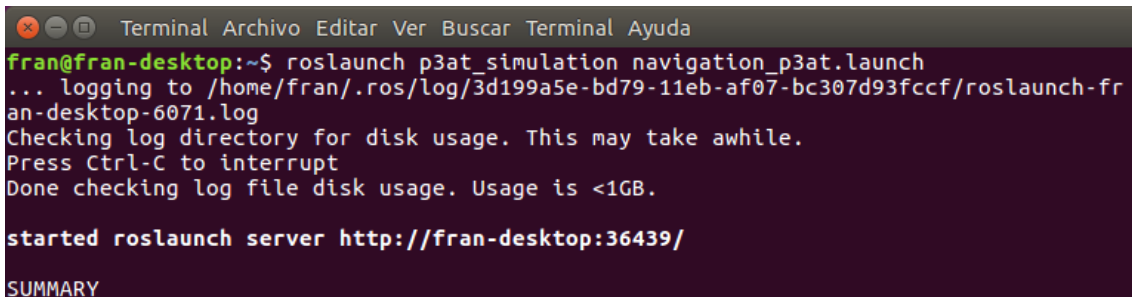


Figura 4.14: Inicio de simulación en Gazebo.



En segundo lugar, se hace uso del archivo “*navigation\_p3at.launch*” para iniciar todos los nodos que conforman el sistema de navegación.

- Abrir nuevo terminal de Ubuntu.
- **\$ roslaunch p3at\_simulation navigation\_p3at.launch**



```


Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ roslaunch p3at_simulation navigation_p3at.launch
... logging to /home/fran/.ros/log/3d199a5e-bd79-11eb-af07-bc307d93fccf/roslaunch-fran-desktop-6071.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://fran-desktop:36439/

SUMMARY

```

Figura 4.15: Ejemplo de uso del archivo *navigation\_p3at.launch*.

En tercer lugar, se inicia RViz como interfaz gráfica para indicar puntos de destino al robot. El archivo de configuración de Rviz, “*sim\_navigation.rviz*”, se encuentra en los archivos adjuntos a este trabajo. 

- Abrir nuevo terminal de Ubuntu.
- **\$ rosrund rviz rviz**

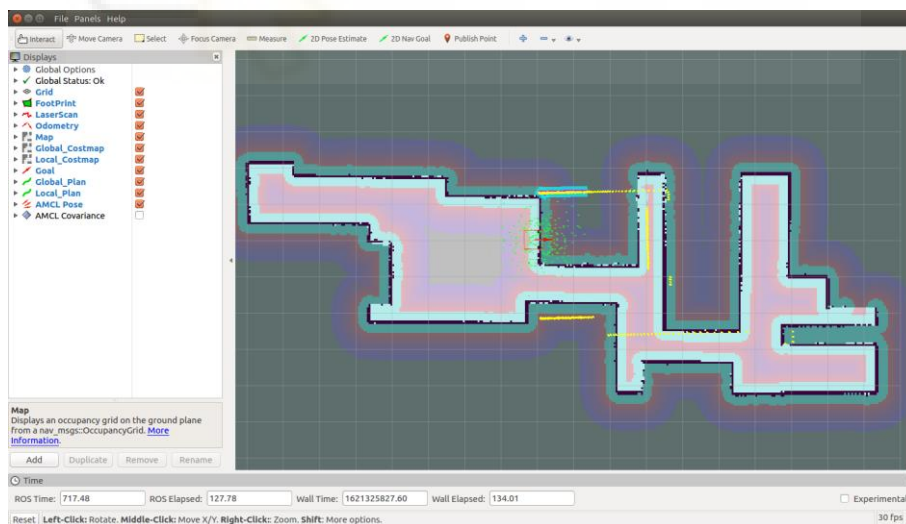


Figura 4.16: RViz para navegación simulada.

Una vez realizados todos los pasos anteriores, ya se pueden indicar puntos de destino al robot, para ello se usa el procedimiento siguiente:

**Primero:** mediante la opción “2D Pose Estimate” de RViz, se ha de indicar la posición y orientación estimada del robot, no es necesaria una gran precisión, pero sí permite que el sistema de navegación ubique al robot más rápidamente. Una vez seleccionada la opción “2D Pose Estimate”, se ha de colocar el cursor sobre el mapa de ocupación y con click izquierdo indicar la posición, y manteniendo pulsado el ratón, mover éste hasta que la flecha coincida con la orientación estimada.

Una vez hecho esto, las líneas amarillas que representan el láser cambiarán y el cuadro delimitador del robot también. La flecha roja indica la odometría del robot, activando y desactivando de la barra lateral izquierda este tópico se actualizará y se mostrará en la ubicación seleccionada con “2D POSE ESTIMATE”.

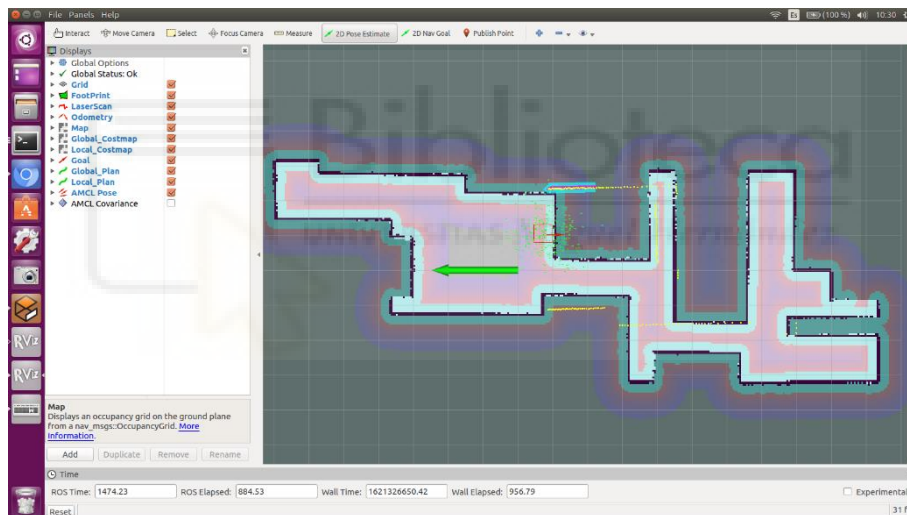


Figura 4.17: 2D POSE ESTIMATE RViz.

**Segundo:** mediante la opción “2D NAV GOAL”, indicar la posición y orientación deseadas del robot de igual forma que para “2D POSE ESTIMATE”.

Una vez indicado el punto de destino y su orientación el robot comenzará a moverse para alcanzar dicho destino. La línea de color azul representa la trayectoria global creada por el planificador global y la flecha azul indica el punto de destino y su orientación.

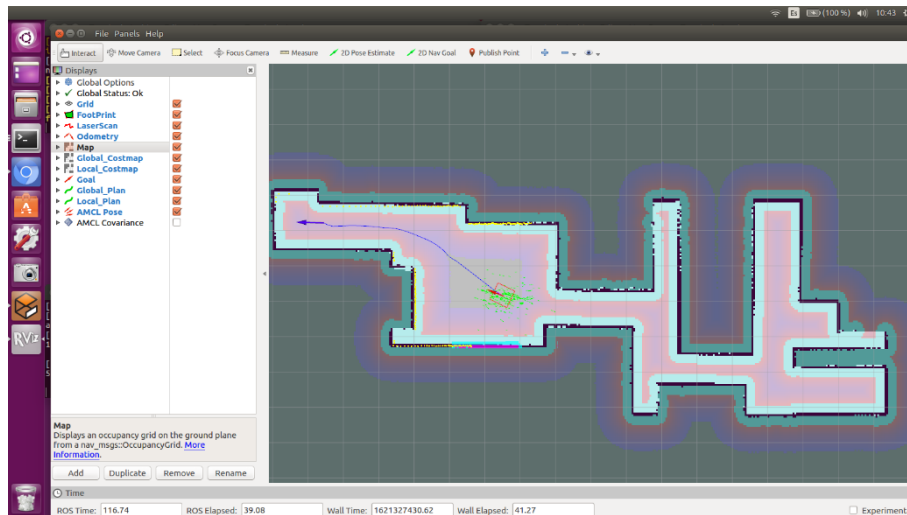


Figura 4.18: 2D NAV GOAL RViz.

### 4.3. RESULTADOS

Durante los experimentos realizados se han observado las siguientes características del sistema de navegación:

- Al realizar giros demasiado cerrados las ruedas del Pioneer-3AT deslizan puesto que funcionan mediante un sistema “*skid-steer*”. Debido a esto la odometría pierde credibilidad y la localización se vuelve errónea pudiendo provocar colisiones.
- El sistema irá localizando la posición del robot a medida que este se desplace por el entorno.
- Resulta conveniente que se tenga en cuenta la posición del siguiente punto de destino a la hora de indicar la orientación del destino actual, para evitar problemas de giros demasiado cerrados.
- Al sólo contar con un sensor láser como fuente de datos del entorno, objetos con una altura menor que el haz horizontal de rayos láser del sensor no son detectados.
- Las trayectorias calculadas por el planificador local son arcos de circunferencia, a mayor tiempo de *forward-simulation* para el DWA, estos arcos tendrán un radio mayor y no serán tan fieles a la trayectoria global.
- Para poder ubicarse, el sistema de navegación, necesita que el sensor láser tenga una cantidad elevada de referencias para poder funcionar

correctamente, esto hace que este sistema sea adecuado para interiores, pero no para exteriores.

- El sistema de navegación permite indicar un nuevo destino mientras se está ejecutando otro.

A continuación se comentan los principales parámetros y los resultados obtenidos al modificar cada uno estos.

- ***amcl\_p3at\_sicklms200\_params***
  - ***kld\_err***: un aumento de este valor hace más permisivo al filtro de partículas produciendo una localización más rápida pero menos precisa. Rango recomendado de 0,01 - 0,1.
- ***global\_costmap\_params***
  - ***inflation\_radius***: si este valor es demasiado alto, el *global\_planner* no podrá generar trayectorias a través de zonas estrechas. Se suele usar el radio del círculo que circunscribe al robot.
  - ***cost\_scaling\_factor***: valores altos de este parámetro implican un descenso muy brusco del coste de las celdas que puede provocar una trayectoria menos segura para el robot.

Se ha de jugar con los parámetros anteriores para adecuar el *global\_costmap* a nuestro mapa, de tal manera que el *global\_planner* pueda generar trayectorias por todas partes del mapa. El *global\_costmap* ha de dejar una zona de bajo coste entre las paredes del mapa para que el planificador global pueda crear una trayectoria a través de ellas.

- ***local\_costmap\_params***
  - ***width y height***: describen la longitud de los lados del cuadrado que define el tamaño del *local\_costmap*. Sus valores dependerán de la aplicación deseada, rangos normales son desde 1 hasta 5 metros.

- **global\_planner\_params**
  - **default\_tolerance:** tolerancia al objetivo, se recomienda colocar la misma que en el planificador local.
  - **neutral\_cost:** mínimo valor de las celdas del costmap que se considera válido para realizar una trayectoria a través de estas. Se recomiendan valores entre 40 - 70.
  - **cost\_factor:** valor que multiplica al coste de las celdas del *costmap*. Se recomiendan valores entre 0,4 - 0,7.

Los dos valores anteriores se deben configurar de forma conjunta, teniendo en cuenta la fórmula:

$$\text{cost} = \text{COST\_NEUTRAL} + \text{COST\_FACTOR} * \text{costmap\_cost\_value}.$$

Los dos valores anteriores se pueden ajustar durante la ejecución del sistema de navegación usando el nodo **rqt\_reconfigure** del paquete **rqt\_configure** a medida que se observa el *global\_costmap* y el *global\_plan* en RViz para dejar una zona libre por donde planificar un camino entre las zonas más estrechas del mapa.

- **base\_local\_planner**
  - **yaw\_goal\_tolerance:** tolerancias muy pequeñas provocan que el robot oscile sin poder alcanzar su objetivo. Se recomiendan tolerancias superiores a 0,2 radianes ( $\sim 10^\circ$ ).
  - **xy\_goal\_tolerance:** tolerancias muy pequeñas provocan que el robot oscile sin poder alcanzar su objetivo. Se recomiendan tolerancias superiores a 0,05 m.
  - **sim\_time:** valores muy pequeños provocan que el robot se mueva de forma muy escalonada cambiando de orientación constantemente y de forma brusca. Valores muy grandes provocan que el robot se mueva realizando arcos con un radio bastante elevado y no se mantenga fiel a la trayectoria global. Se recomiendan valores mínimos de 1 segundo y máximos de 3.

- ***pdist\_scale***: valores altos provocan constantes cambios de orientación bruscos en el robot. Aumentar su valor implica que la trayectoria del robot se asemeje más a la trayectoria descrita por el planificador global.
- ***gdist\_scale***: valores muy altos provocan que el robot ignore la trayectoria global indicada y puede provocar trayectorias peligrosas.
- ***occdist\_scale***: se recomiendan valores muy pequeños, del orden de 0,01-0,05 porque si no el robot se detendrá cuando tenga que pasar cerca de un obstáculo.



## 5. INTEGRACIÓN EN ROBOT REAL

En este capítulo se redacta el proceso empleado para pasar del sistema de navegación realizado en simulación, al robot real.

### 5.1. IMPLEMENTACIÓN

En este apartado se comenta el proceso que se ha seguido para iniciar todos los sistemas del robot y el sistema de navegación.

#### 5.1.1. CREACIÓN DE UN NUEVO PAQUETE

Al igual que en simulación, se ha creado un paquete que contenga todos los archivos necesarios para iniciar el sistema de navegación. El paquete en cuestión se ha denominado *p3at\_real*.

- Abrir terminal de Ubuntu.
- **`$ cd programas_ros/catkin_ws/src`**  
Nos movemos a la carpeta `/src` de nuestro *catkin\_workspace*.
- **`$ catkin_create_pkg p3at_real roscpp geometry_msgs tf move_base`**  
Creamos un paquete que se denomina *p3at\_real* e indicamos los paquetes de los que depende.
- **`$ cd programas_ros/catkin_ws`**
- **`$ catkin_make`**  
Compilamos el paquete.

Una vez compilado el paquete se crearán de forma manual las carpetas:

- ***/bagfiles***: carpeta empleada para almacenar los *bagfiles* generados durante los experimentos mediante el comando *rosvbag*.
- ***/launch***: carpeta dónde se encuentran todos los *launch\_files* disponibles en este paquete.
- ***/params***: dentro de esta carpeta se encuentran los archivos de extensión *“.yaml”* que contienen los parámetros de configuración para cada nodo implicado en el sistema de navegación.

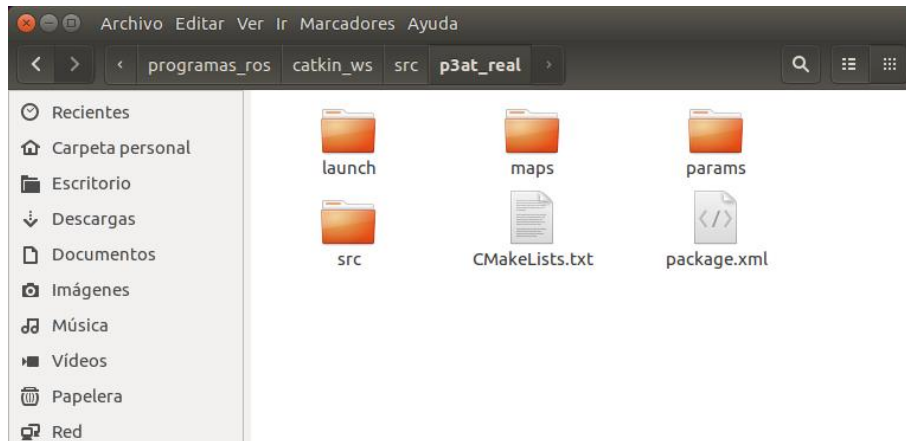


Figura 5.1: Contenido del paquete `p3at_real`.

### 5.1.2. PARÁMETROS DEL NAVIGATION STACK

Los archivos que configuran los parámetros de los nodos implicados en el sistema de navegación son los mismos que los empleados para el sistema de navegación simulado. Al paquete `p3at_real` se le añaden los archivos `rosaria_params.yaml` y `sick_lms200_params.yaml` que configuran los nodos `RosAria` y `sicklms` respectivamente. Además, en el archivo `local_costmap_params.yaml` el tópico del láser cambia.



Figura 5.2: Archivos de parámetros paquete `p3at_real`.



### 5.1.3. LAUNCH\_FILES

La gran mayoría de *launch\_files* del paquete ***p3at\_real*** coinciden con los *launch\_files* del paquete ***p3at\_simulation***. A continuación se enumeran los *launch\_files* disponibles en el paquete *p3at\_real*.

- ***rosaria\_p3at***: inicia RosAria con sus respectivos parámetros.
- ***sick\_lms200\_p3at***: inicia *sicktoolbox\_wapper* con sus respectivos parámetros.
- ***tf\_laser***: publica la transformada entre el láser y *base\_link*.
- ***p3at***: inicia los tres *launch\_files* anteriores a la vez.
- ***amcl\_p3at***: inicia el nodo *amcl* con sus respectivos parámetros.
- ***map\_server***: inicia el nodo *map\_server* con sus respectivos parámetros.
- ***move\_base\_p3at***: inicia el nodo *move\_base* con sus parámetros y los parámetros referentes a todos los nodos que engloba *move\_base*.
- ***navigation\_p3at***: llama a los *launch\_files* *amcl\_p3at*, *map\_server* y *move\_base\_p3at* simultáneamente.

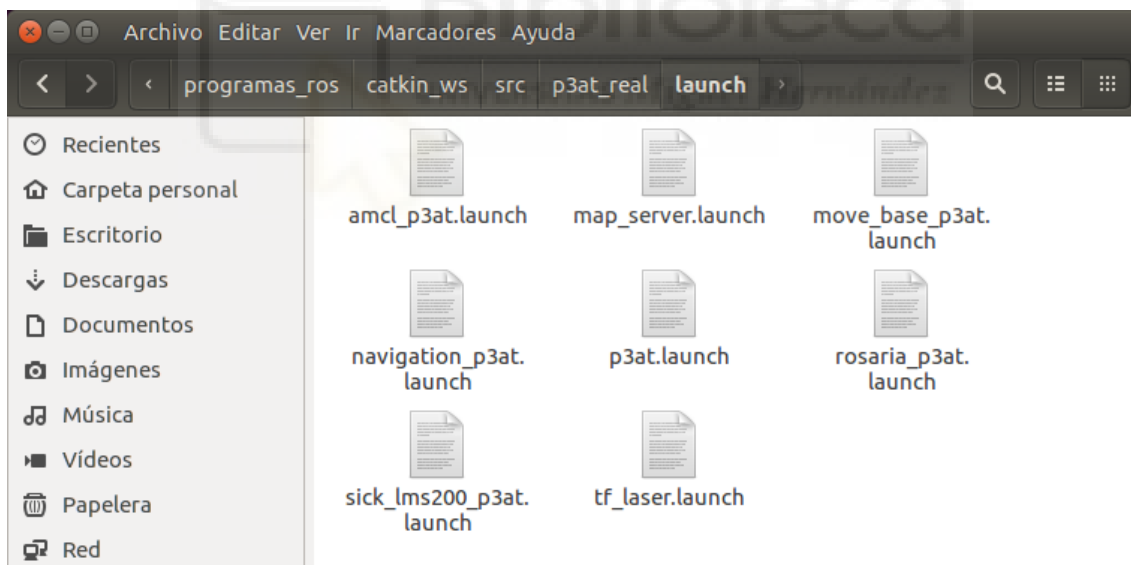


Figura 5.3: Launch\_files del paquete *p3at\_real*.

## 5.2. USO

Para hacer uso del paquete **p3at\_real**, éste tendrá que estar ubicado en el ordenador equipado sobre el Pioneer-3AT. Para ello se puede seguir el procedimiento indicado en [2.2.2.4](#) para pasar archivos de un equipo a otro.

**Primero:** establecer la conexión remota entre el ordenador equipado en el Pioneer-3AT y el ordenador de control. Se diferenciarán terminales de control y terminales remotos, que corresponden al equipo de control y al equipo montado sobre el Pioneer-3AT respectivamente.

### Terminal de control

- Abrir terminal de Ubuntu.
- **\$ export ROS\_IP:=192.168.120.126**
- **\$ export ROS\_MASTER\_URI:= https://192.168.120.125:11311**  
En caso de tener los comandos anteriores en el archivo “.bashrc” como se explica en el punto [2.2.2.3](#) no será necesario volver a indicarlos.
- **\$ ssh arvc@192.168.120.125**
- Introducir contraseña de *arvc* y pulsar intro.

Una vez hecho esto estaremos conectados a un terminal del equipo montado sobre el Pioneer-3AT.

### Terminal remoto


- **\$ export ROS\_IP:=192.168.120.125**
- **\$ export ROS\_MASTER\_URI:= https://192.168.120.125:11311**  
En caso de tener los comandos anteriores en el archivo “.bashrc” como se explica en el punto [2.2.2.3](#) no será necesario volver a indicarlos.
- **\$ roslaunch p3at\_real p3at.launch**  
Inicia *RosAria*, *sicklms* y *tf\_laser*.
- **\$ roslaunch p3at\_real navigation\_p3at.launch**  
Inicia todos los nodos involucrados en el sistema de navegación.

Llegados a este punto tanto el Pioneer-3AT, el láser Sick-LMS200 y el sistema de navegación están activos, sólo falta indicar puntos de destino del mismo modo que se ha explicado para la navegación simulada.

#### Terminal de control

➤ Abrir nuevo terminal de Ubuntu.

➤ **\$ rosrun rviz rviz**

Una vez abierto RViz cargar la configuración “*real\_navigation.rviz*” que se proporciona en los archivos adjuntos a este trabajo. 

### 5.3. RESULTADOS

La configuración de los parámetros del sistema de navegación es la misma tanto para simulación como para navegación real, por lo tanto, como cabía esperar, los resultados obtenidos son los mismos que los obtenidos para la navegación simulada.

- Si la conexión del láser falla la transformada entre */laser* y */base\_link* se perderá. Esto provoca que el *tf\_tree* no esté completo y genera fallos en la navegación.

## 6. SEGUIDOR DE TRAYECTORIAS

En este apartado se redacta el desarrollo de una aplicación que permite almacenar las trayectorias realizadas por un robot y para su posterior reproducción. El proceso que emplea consiste en tres partes, una primera parte donde almacena todas las posiciones del robot respecto del mapa mientras se está moviendo, una segunda parte donde se hace un remuestreo de dichas posiciones, y una última parte que genera los comandos de velocidad para que el robot intente alcanzar dichas posiciones. Cabe destacar que el nodo que genera los comandos de velocidad no está completo, realiza un seguimiento muy básico de las posiciones almacenadas, pero se deja como posible base para futuros desarrollos.

Para el desarrollo de la aplicación, las posiciones del robot se han almacenado en un archivo que contiene un conjunto de objetos de tipo JSON. Para más información sobre el formato JSON [52]. Para poder trabajar con este formato de archivos se ha de indicar en el archivo “*CMakeLists.txt*” de nuestro paquete, que se emplea la librería “*jsoncpp*”.

La aplicación se ha desarrollado dentro del paquete *p3at\_simulation*, por lo tanto, se han añadido al archivo “*CMakeLists.txt*” de dicho paquete las siguientes líneas:

```

1 cmake_minimum_required(VERSION 3.0.2)
2 project(p3at_simulation)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 add_compile_options(-std=c++11) #...WAS COMENTED...#
6 add_compile_options(-ljsoncpp) #...WASN'T EXIST...#
7
8 ## Find catkin macros and libraries
9 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
10 ## is used, also find other catkin packages
11 find_package(catkin REQUIRED COMPONENTS
12   move_base
13   geometry_msgs
14   roscpp
15   tf
16 )
17
18 find_package(PkgConfig REQUIRED) #...ADDED...#
19 pkg_check_modules(JSONCPP jsoncpp) #...ADDED...#
20 link_libraries(${JSONCPP_LIBRARIES}) #...ADDED...#
~

```

```

118 #####
119 ## Build ##
120 #####
121
122 ## Specify additional locations of header files
123 ## Your package locations should be listed before other locations
124 include_directories(
125 # include
126   ${catkin_INCLUDE_DIRS}
127   ${JSONCPP_INCLUDE_DIRS}   #...ADDED...#
128 )

```

Figura 6.1: Líneas añadidas a CMakeLists de paquete *p3at\_simulation*.

Además, para el proceso de guardado de las posiciones del robot se ha empleado el nodo ***hector\_mapping*** del paquete ***hector\_mapping***. Este nodo publica bajo el tópico ***/slam\_out\_pose*** un mensaje de tipo *geometry\_msgs/PoseStamped*, que será el que empleemos para guardar las posiciones del robot al realizar una trayectoria.

## 6.1. NODOS DISPONIBLES

Para esta aplicación se han desarrollado 4 nodos. Los códigos fuente de estos nodos se encuentran debidamente comentados en el apartado de anexos de este trabajo.

- ***poses\_saver.cpp***: este nodo se suscribe al tópico */slam\_out\_pose* y almacena las poses publicadas en dicho tópico en formato JSON en el archivo “*full\_poses.json*”.
- ***poses\_resampling.cpp***: accede al archivo “*full\_poses.json*” y hace un remuestreo de todas las posiciones y las guarda en el archivo “*resampled\_poses.json*”.
- ***path\_publisher.cpp***: accede al archivo “*full\_poses.json*” y publica dichas posiciones bajo el tópico */globalpath* con formato *nav\_msgs/Path* para tener una visualización de la trayectoria seguida por el robot en Rviz.
- ***point\_follower.cpp***: accede al archivo “*resampled\_poses.json*” y genera los comandos de velocidad necesarios para que el robot alcance dichas posiciones.

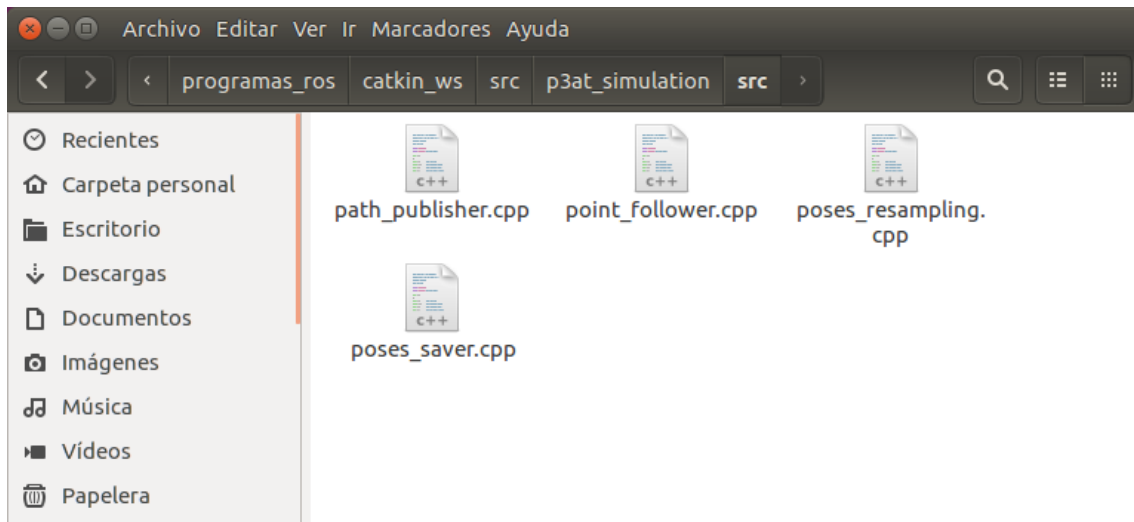


Figura 6.2: Nodo disponibles para seguidor de trayectorias.

## 6.2. USO

A continuación se detalla el uso de cada uno de los nodos mencionados en la sección anterior.

### 6.2.1. GUARDADO DE TRAYECTORIAS

En primer lugar se necesita almacenar los puntos que describen la trayectoria descrita por el robot. Para esta tarea vamos a emplear los nodos ***hector\_mapping*** y ***poses\_saver***.

- Abrir terminal de Ubuntu.
- `$ roslaunch p3at_simulation robot_p3at_point_follower_world_simulated.launch`  
Se usa el archivo “*robot\_p3at\_point\_follower\_world\_simulated.launch*” para iniciar una simulación del robot en un determinado entorno.

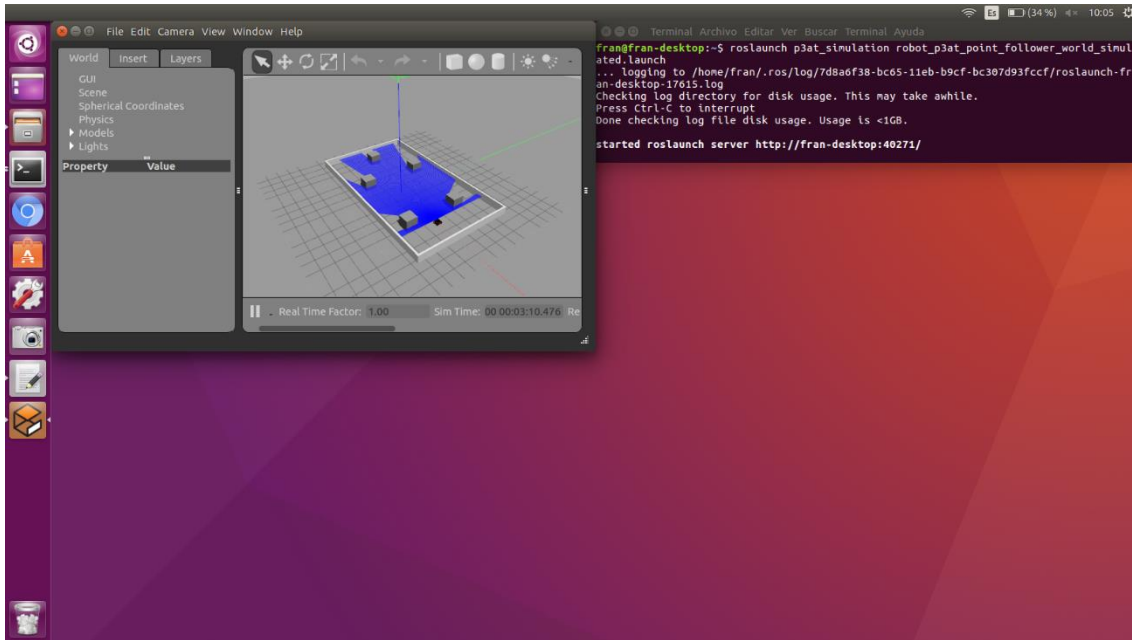


Figura 6.3: Ejemplo de uso del archivo `robot_p3at_point_follower_world_simulated.launch`

- Abrir nuevo terminal de Ubuntu.
- **`$ roslaunch p3at_simulation hector_mapping.launch`**  
Se usa el archivo [“`hector\_mapping.launch`”](#) para iniciar el nodo `hector_mapping` con sus respectivos parámetros.

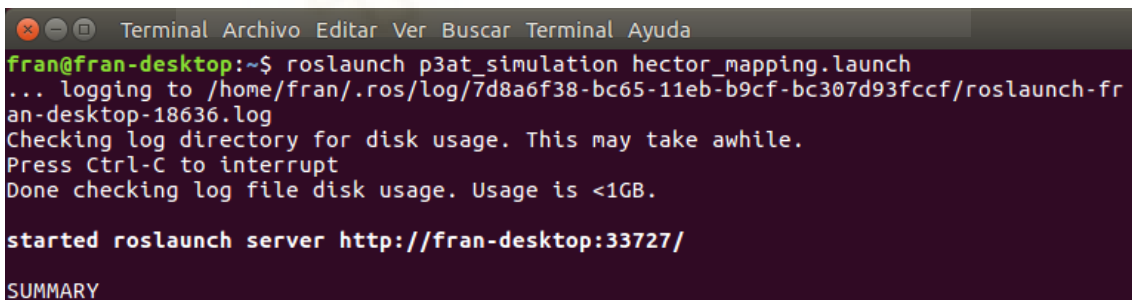


Figura 6.4: Ejemplo de uso del archivo `hector_mapping.launch`.

- Abrir nuevo terminal de Ubuntu.
- **`$ roslaunch p3at_simulation poses_saver`**  
Este nodo comenzará a capturar las poses en cuanto estas se publiquen bajo el nodo `/slam_out_pose`.

```

Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ rosrund p3at_simulation poses_saver
Nodo listo para capturar poses.
^C
Numero de poses capturadas: 3392
fran@fran-desktop:~$ |

```

Figura 6.5: Ejemplo de captura de poses con el nodo `poses_saver`.

- Abrir nuevo terminal de Ubuntu.
- **\$ `rosvrun key_teleop key_teleop`**  
 Hacemos uso del nodo `key_teleop` para mover el robot alrededor del entorno.
- Una vez llegados a este punto, realizar la trayectoria deseada del robot mediante el nodo `key_teleop`. Una vez completada la trayectoria deseada, bastará con matar el nodo `poses_saver` para que genere el archivo `full_poses.json` en el directorio donde se haya ejecutado el nodo.
- Además, si no se dispone de un mapa del entorno, en este momento se puede hacer uso del nodo `map_saver` para almacenar el mapa de ocupación tal y como se ha explicado en capítulos anteriores.

```

Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Abrir [icon] Guardar
1 [
2 {
3   "Header" : {
4     "frame_id" : "map",
5     "seq" : 661,
6     "stamp" : {
7       "nsec" : 739000000,
8       "sec" : 140
9     }
10  },
11  "Pose" : {
12    "1 Translation" : {
13      "x" : 0.012577056884765625,
14      "y" : -0.00302886962890625,
15      "z" : 0
16    },
17    "2 Rotation" : {
18      "qx" : 0,
19      "qy" : 0,
20      "qz" : 0.0014782312064764134,
21      "w" : 0.99999890741565323
22    }
23  }
24 },
25 {
26   "Header" : {
27     "frame_id" : "map",
28     "seq" : 663.

```

JSON Anchura de la pestaña: 4 Ln 281, Col 22 INS

Figura 6.6: Ejemplo archivo `full_poses.json`.

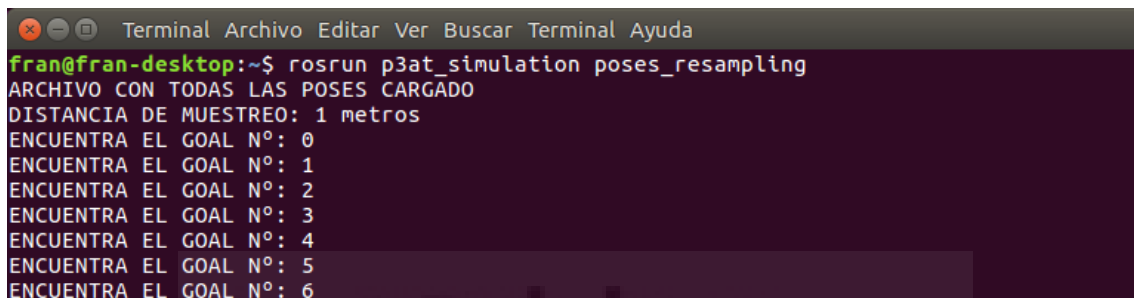


### 6.2.2. REMUESTREO DE POSICIONES

Para realizar un remuestreo de las posiciones se emplea el nodo `poses_resampling.cpp`.

- Abrir terminal de Ubuntu.
- **\$ rosrund p3at\_simulation poses\_resampling**

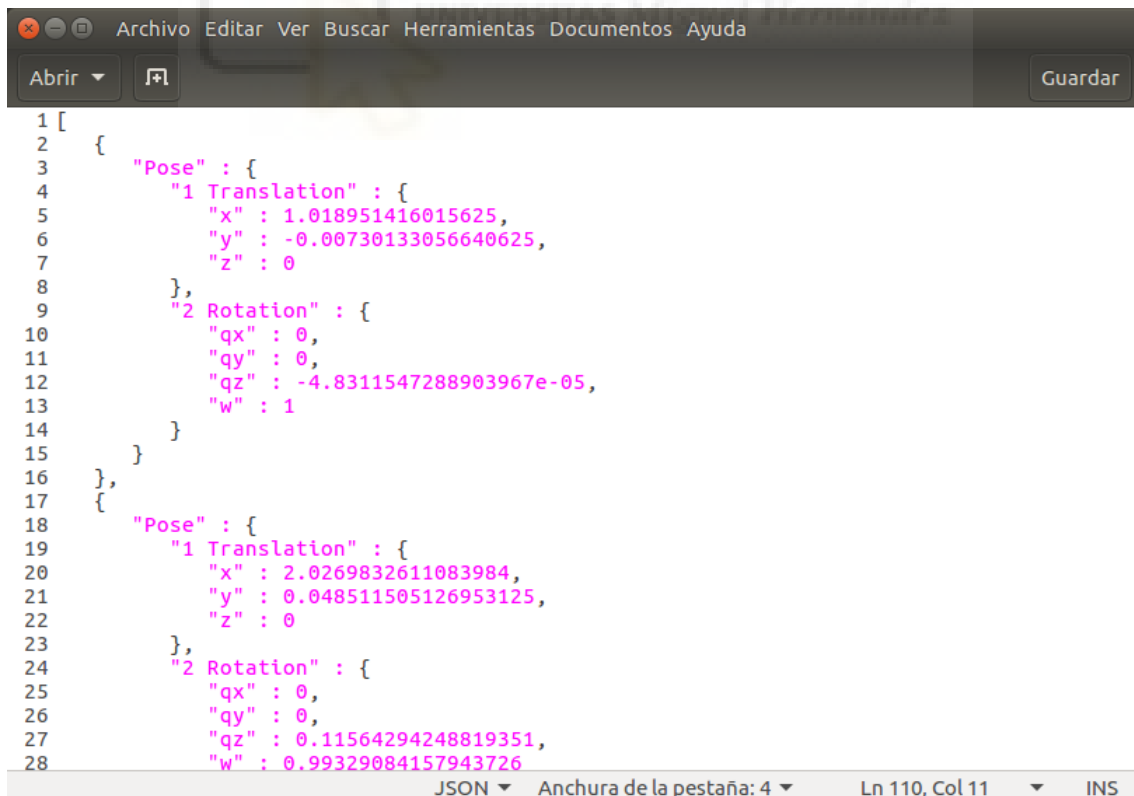
Hará un remuestreo de las posiciones buscando puntos cuya distancia sea de aproximadamente 1 metro, este valor se puede cambiar accediendo al código fuente del nodo. Una vez haya recorrido todo el archivo `full_poses.json` generará el archivo `resampled_poses.json`.



```

Terminal Archivo Editar Ver Buscar Terminal Ayuda
fran@fran-desktop:~$ rosrund p3at_simulation poses_resampling
ARCHIVO CON TODAS LAS POSES CARGADO
DISTANCIA DE MUESTREO: 1 metros
ENCUENTRA EL GOAL N°: 0
ENCUENTRA EL GOAL N°: 1
ENCUENTRA EL GOAL N°: 2
ENCUENTRA EL GOAL N°: 3
ENCUENTRA EL GOAL N°: 4
ENCUENTRA EL GOAL N°: 5
ENCUENTRA EL GOAL N°: 6
  
```

Figura 6.8: Ejemplo de uso del nodo `poses_resampling`.



```

Archivo Editar Ver Buscar Herramientas Documentos Ayuda
Abrir Guardar
1 [
2   {
3     "Pose" : {
4       "1 Translation" : {
5         "x" : 1.018951416015625,
6         "y" : -0.00730133056640625,
7         "z" : 0
8       },
9       "2 Rotation" : {
10        "qx" : 0,
11        "qy" : 0,
12        "qz" : -4.8311547288903967e-05,
13        "w" : 1
14      }
15    }
16  },
17  {
18    "Pose" : {
19      "1 Translation" : {
20        "x" : 2.0269832611083984,
21        "y" : 0.048511505126953125,
22        "z" : 0
23      },
24      "2 Rotation" : {
25        "qx" : 0,
26        "qy" : 0,
27        "qz" : 0.11564294248819351,
28        "w" : 0.99329084157943726
  
```

Figura 6.7: Ejemplo archivo `resampled_poses.json`.

### 6.2.3. PUBLICACIÓN DE LA TRAYECTORIA

En ocasiones puede ser necesario obtener de forma visual la trayectoria que debe realizar el robot, esto se obtiene mediante el nodo `path_publisher.cpp`. Este nodo publica la trayectoria descrita por los puntos almacenados en el archivo `“full_poses.json”`.

- Abrir terminal de Ubuntu.
- **\$ `roslaunch p3at_simulation path_publisher`**

Publica bajo el tópic `/globalpath` la trayectoria seguida por el robot. Esta trayectoria se puede ver haciendo uso de Rviz.

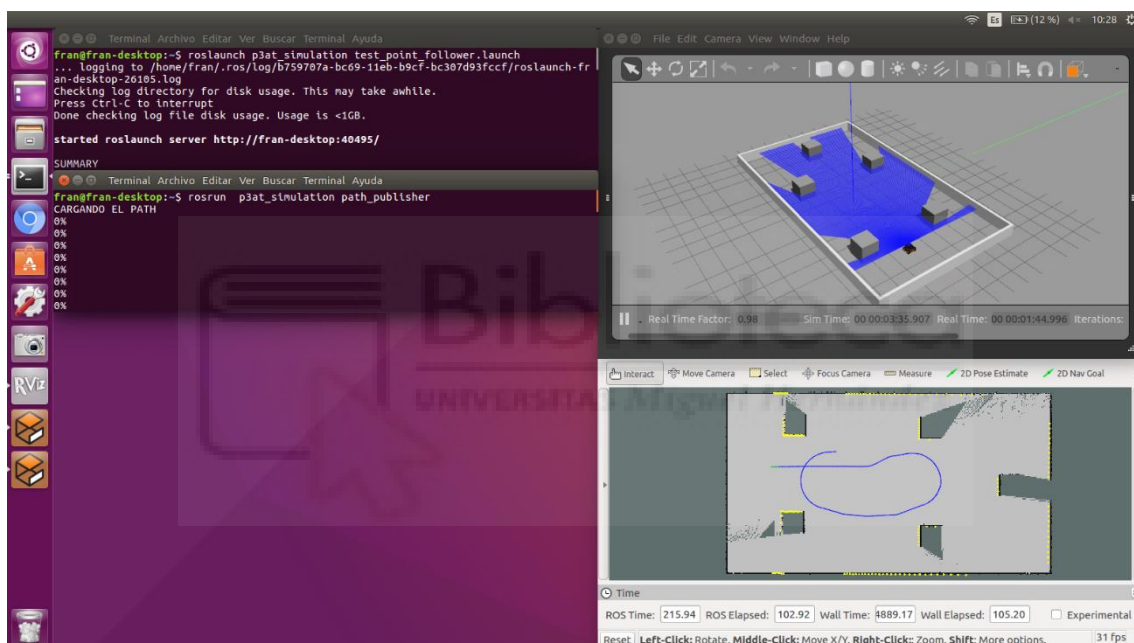


Figura 6.10: Ejemplo de uso del nodo `path_publisher`.

### 6.2.4. SEGUIMIENTO DE LA TRAYECTORIA

Para hacer el seguimiento de la trayectoria se necesita del nodo `amcl`, para que publique la posición real del robot y la publicación de un mapa sobre el que ubicarse.

- Abrir terminal de Ubuntu.
- **\$ `roslaunch p3at_simulation points_follower`**

Tras ejecutar este nodo el robot comenzará a moverse intentando alcanzar los puntos indicados en el archivo `“resampled_poses.json”`.

Para lanzar *amcl*, junto con un mapa y Rviz, está disponible el archivo [“test\\_point\\_follower.launch”](#) que permite comprobar la trayectoria realizada por el robot desde Rviz.

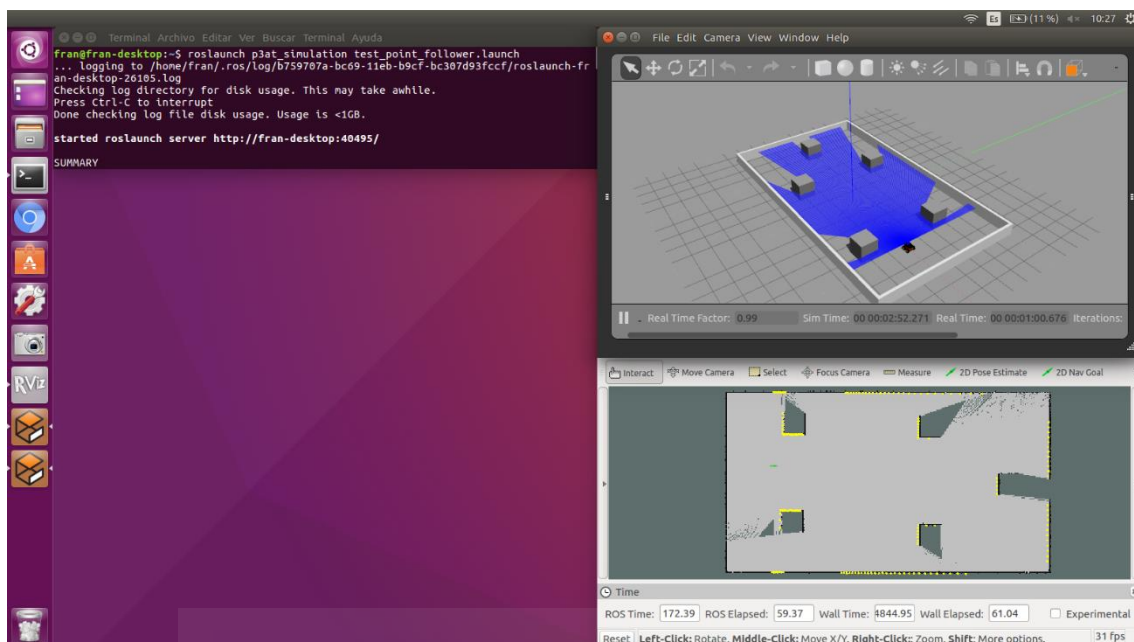


Figura 6.11: Ejemplo de uso del archivo *test\_point\_follower.launch*.

### 6.3. RESULTADOS

Estos nodos son una primera aproximación a lo que sería un desarrollo de un seguidor de trayectorias desarrollado de forma propia. Respecto al guardado de las posiciones del robot, y su remuestreo no parece haber ningún problema. El problema se encuentra al intentar seguir dichos puntos, que a la hora de realizar giros el sistema que controla la velocidad angular que tiene implementado no es suficiente para que el robot alcance de forma efectiva su destino. Se deja como base para poder profundizar en el desarrollo de un sistema de control que permita seguir de forma correcta una trayectoria partiendo de unos puntos dados.

## 7. CONCLUSIONES Y TRABAJOS FUTUROS

### 7.1. CONCLUSIONES

Tras desarrollar el presente trabajo usando las herramientas que ofrece ROS para implementar la navegación de robots móviles se han obtenido las siguientes conclusiones:

- La plataforma ROS, ofrece un mundo de posibilidades para implementar aplicaciones en robótica. Siendo cierto que su curva de aprendizaje es lenta, una vez alcanzas suficiente conocimiento sobre su uso, te percatas de que permite implementar aplicaciones con un enorme grado de abstracción respecto al código que las lleva a cabo. Es decir, tan sólo necesitas saber la serie de tareas que necesita realizar tu sistema para cumplir la función que deseas, una vez sabido esto, buscar los paquetes que realicen dichas tareas y usar estos paquetes como cajas negras, las cuales necesitan de unas ciertas entradas para funcionar y proporcionan unas salidas. Haciendo coincidir las entradas y salidas de unos paquetes con otros puedes entrelazarlos para implementar aplicaciones de forma rápida y sencilla. Además, ROS dispone de una comunidad muy grande de usuarios que comparten sus experiencias, problemas y soluciones ayudándose unos a otros. Por añadir algo más, todos los paquetes de ROS suelen estar bien documentados.
- El entorno de simulación de Gazebo, permite realizar simulaciones prácticamente de cualquier robot existente. Su curva de aprendizaje es bastante sencilla y permite a los usuarios realizar simulaciones de sistemas robóticos sin necesidad de disponer de ellos. Dispone de una amplia cantidad de sensores como cámaras, cámaras con sensor de profundidad, láseres, IMU, etc. Además cuenta con varios plugins para simular el movimiento de distintos tipos de robots móviles.
- Por último, cabe destacar que para el correcto funcionamiento de los paquetes implementados mediante ROS, se necesita realizar una

correcta configuración de los parámetros de dichos paquetes en concordancia con el equipo y la aplicación que se quiera desarrollar.

## 7.2. TRABAJOS FUTUROS

El seguidor de trayectorias desarrollado, además de realizar un seguimiento de una trayectoria dentro de un mapa con la ayuda de un localizador mediante láser, se podría realizar el seguimiento de trayectorias mediante una localización GPS. Sería relativamente sencillo, cambiar el tópic al que se suscribe el nodo encargado de almacenar los puntos de la trayectoria y colocar un tópic donde se publique la ubicación GPS del robot y adecuar el formato del archivo JSON. Teniendo como localizador el GPS y como mapa la “Tierra” el robot podría seguir una trayectoria descrita por puntos GPS.



## 8. ANEXOS

### Anexo I: *rosaria\_params.yaml*

```
#RosAria PARAMETERS

port: /dev/ttyUSB0 #Puerto del PC al que está conectado el robot.
#baud: '' #Baudios a los que trabaja RosAria. Comentado para que use
valor predeterminado.
debug_aria: false #Permite lanzar el nodo RosAria en modo depuración y
los datos son guardados en un archivo de registro.

#aria_log_filename:      Aria.log      #Nombre de archivo de
registros en el que guardar los datos anteriores.

publish_aria_lasers: false #Usa la librería Aria como driver para
láseres conectados y configurados dentro de Aria.

cmd_vel_timeout: 0.6 #Si no se reciben comandos de velocidad durante
ese tiempo RosAria detiene el robot.

#MOVEMENT PARAMETERS
trans_accel: 0.0 #Aceleración de traslación. 0 = valores
internos del robot.
trans_decel: 0.0 #Deceleración de traslación. 0 = valores
internos del robot.
rot_accel: 0.0 #Aceleración de rotación. 0 = valores internos
del robot.
rot_decel: 0.0 #Deceleración de rotación. 0 = valores
internos del robot.
lat_accel: 0.0 #Aceleración de traslación lateral. 0 =
valores internos del robot. No aplicable al Pioneer-3AT.
lat_decel: 0.0 #Deceleración de traslación lateral. 0 =
valores internos del robot. No aplicable al Pioneer-3AT.

#FRAME NAMES
odom_frame:      odom      #Identifica el origen asociado a la
odometría para las transformadas.
base_link_frame: base_link #Identifica el origen asociado al
base_link para las transformadas.
sonar_frame:     sonar     #Identifica el origen asociado al
sonar para las transformadas. No aplicado en este caso.
bumpers_frame:  bumpers   #Identifica el origen asociado a los
bumpers para las transformadas. No aplicable al Pioneer-3AT.

#ODOMETRY CONFIGURATION
DriftFactor: -99999 #Numero de pulsos a quitar o sumar al
encoder izquierdo para corregir diferencias entre ruedas.
RevCount: 0 #Pulsos emitidos por los encoders para
realizar un giro sobre sí mismo de 180°.
TicksMM: 0 #Numero de pulsos emitidos por el
encoder para 1 mm de traslación de las ruedas.
```

## Anexo II: *rosaria\_p3at.launch*

```
<launch>

<!--INITIALIZE ROSARIA-->
<node name="RosAria" pkg="rosaria" type="RosAria">

<rosparam file="$(find p3at_real)/params/rosaria_params.yaml"
command="load" />

</node>
</launch>
```

## Anexo III: *sick\_lms200\_params.yaml*

```
#CONFIGURACION PARA LASER SICK LMS 200

use_rep_117:    true           #INDICA SI SE USA O NO, EL ESTADAR REP
117 PARA SENSORES. DEFAULT= TRUE
port:          /dev/ttyUSB1   #INDICA EL PUERTO AL QUE ESTA
CONECTADO EL LASER. DEFAULT=/dev/ttyUSB1
baud:         38400           #INDICA LOS BAUDIOS A LOS QUE TRABAJA
EL LASER. DEFAULT=38400
inverted:     false          #INDICA SI SE INVIERTE O NO, LA
CONVENCIÓN PARA ANGULOS POSITIVOS. DEFAULT= FALSE
frame_id:    laser           #IDENTIFICADOR DEL ELEMENTO (FRAME)
DEL LASER. DEFAULT= LASER
angle:       180              #RANGO DE TRABAJO DEL LASER EN GRADOS.
DEFAULT= 0 (0 = RANGE DEVUELTO POR EL LASER)
resolution:  0.0              #RESOLUCIÓN ANGULAR DEL LASER.
DEFAULT= 0 (0 = RESOLUTION DEVUELTA POR EL LASER)
connect_delay: 30.0          #TIEMPO EN SEGUNDOS A ESPERAR ENTRER
ABRIR EL PUERTO Y CONECTAR EL LASER. DEFAULT= 30
```

## Anexo IV: *sick\_lms200\_p3at.launch*

```
<launch>

<!--INITIALIZE LASER SICK LMS200-->
<node name="laser_lms200" pkg="sicktoolbox_wrapper" type="sicklms">

<rosparam file="$(find p3at_real)/params/sick_lms200_params.yaml"
command="load"/>

</node>
</launch>
```

## Anexo V: `robot_p3at_lab_world_simulated.launch`

```
<launch>
```

```

<!-- INDICAMOS DONDE ESTA LA CARPETA CON NUESTROS MODELOS -->
<env name="GAZEBO_MODEL_PATH"
value="$GAZEBO_MODEL_PATH:$(find p3at_simulation)/models/urdf"
/>
<env name="GAZEBO_RESOURCE_PATH"
value="$GAZEBO_RESOURCE_PATH:$(find
p3at_simulation)/models/urdf" />

<!-- CREAMOS ARGUMENTOS PARA CONFIGURAR LA SIMULACIÓN -->
<arg name="paused"                default="false" />
<arg name="use_sim_time"           default="true" />
<arg name="extra_gzbo_args"        default="" />
<arg name="gui"                    default="true" />
<arg name="headless"               default="false" />
<arg name="debug"                  default="false" />
<arg name="verbose"                default="true" />
<arg name="output"                 default="screen" />
<arg name="physics"                default="ode" />
<arg name="respawn_gazebo"         default="false" />
<arg name="server_required"        default="false" />
<arg name="gui_required"           default="false" />
<arg name="world"                  default="$(find
p3at_simulation)/models/worlds/real_lab.world" />

<!-- INDICAMOS MODELO DEL ROBOT -->
<arg name="urdf"                   default="$(find
p3at_simulation)/models/urdf/my_p3at.urdf" />

<!-- SETEAMOS EL PARÁMETRO ROBOT_DESCRIPTION -->
<param name="robot_description" command="$(find xacro)/xacro -i
$(arg urdf)"/>

<!-- POSICION INICIAL DEL ROBOT EN ENTORNO SIMULACION -->
<arg name="x_off" value="-1" /> <!--Desplazamiento en x-->
<arg name="y_off" value="-1" /> <!--Desplazamiento en y-->
<arg name="Y_off" value="3.1416"/> <!--Rotación respecto z-->

<!-- SETEAMOS EL PARAMETRO /use_sim_time A TRUE -->
<param name="/use_sim_time" value="$(arg use_sim_time)" />

<!-- INSERTAMOS NUESTRO ROBOT EN GAZEBO -->
<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
args="-param robot_description -urdf -x $(arg x_off) -y $(arg y_off) -
Y $(arg Y_off) -model Pioneer-3AT" />

<!-- INICIAMOS NODO EL CUAL PUBLICA NUESTRAS TRANSFORMADAS -->
<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />

<!-- INICIAMOS NODO PUBLICA EL ESTADO DE LAS UNIONES DEL ROBOT -->
<node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />

```



```

<!-- SET COMMAND ARGUMENTS -->
<arg unless="$(arg paused)" name="command_arg1" value="" />
<arg if="$(arg paused)" name="command_arg1" value="-u" />
<arg unless="$(arg headless)" name="command_arg2" value="" />
<arg if="$(arg headless)" name="command_arg2" value="-r" />
<arg unless="$(arg verbose)" name="command_arg3" value="" />
<arg if="$(arg verbose)" name="command_arg3" value="--verbose" />
<arg unless="$(arg debug)" name="script_type" value="gzserver" />
<arg if="$(arg debug)" name="script_type" value="debug" />

<!-- INICIAMOS GAZEBO SERVER -->
<node name="gazebo"
  pkg="gazebo_ros"
  type="$(arg script_type)"
  respawn="$(arg respawn_gazebo)"
  output="$(arg output)"
  args="$(arg command_arg1) $(arg command_arg2) $(arg
command_arg3) -e $(arg physics) $(arg extra_gzbo_args)
$(arg world)"
  required="$(arg server_required)" />

<!-- INICIAMOS GAZEBO CLIENT -->
<group if="$(arg gui)">
  <node name="gazebo_gui"
    pkg="gazebo_ros"
    type="gzclient"
    respawn="false"
    output="$(arg output)"
    args="$(arg command_arg3)"
    required="$(arg gui_required)"/>
</group>
</launch>

```

## Anexo VI: *tf\_laser.launch*

```

<launch>

<!--PUBLICA LA TRANSFORMADA ENTRE base_link Y laser-->
<node pkg="tf" type="static_transform_publisher"
name="base_to_laser_broadcaster" args=" 0.15 0 0.1 0 0 0 base_link
laser 100" />

</launch>

```

## Anexo VII: *p3at.launch*

```

<launch>

<!--INICIA ROSARIA-->
<include file="$(find p3at_real)/launch/rosaria_p3at.launch"/>

<!--INICIA LASER SICK LMS200-->
<include file="$(find p3at_real)/launch/sick_lms200_p3at.launch"/>

```

```

<!--PUBLICA LA TRANSFORMADA ENTRE base_link Y laser-->
<include file="$(find p3at_real)/launch/tf_laser.launch"/>

</launch>

```

## Anexo VIII: *map\_server.launch*

```

<launch>
<!-- NODO MAP SERVER -->

<!--SIMULACIÓN-->
<!--<node name="map_server" pkg="map_server" type="map_server"
args="$(find p3at_simulation)/models/maps/real_lab.yaml"> </node-->

<!--REAL-->
<node name="map_server" pkg="map_server" type="map_server"
args="$(find p3at_real)/maps/laboratorio.yaml"> </node>

</launch>

```

## Anexo IX: *amcl\_p3at\_sicklms200\_params.yaml*

```

#CONFIGURATION FOR AMCL

#FILTER PARAMETERS

min_particles: 50 #MINIMO NUMERO DE PARTICULAS A USAR.
DEFAULT= 100
max_particles: 500 #MAXIMO NUMERO DE PARTICULAS A USAR.
DEFAULT= 5000
kld_err: 0.05 #MAXIMO ERROR ENTRE LA DISTRIBUCION REAL Y
LA ESTIMADA. DEFAULT= 0.01
kld_z: 0.99 #DEFAULT= 0.99

update_min_d: 0.2 #TRASLACION EN METROS A LA CUAL ACTUALIZAR
EL FILTRO DE PARTICULAS. DEFAULT= 0.2
update_min_a: 0.5 #ROTACION EN RADIANTES A LA CUAL ACTUALIZAR
EL FILTRO DE PARTICULAS. DEFAULT= 0.5
resample_interval: 1 #NUMERO DE ACTUALIZACIONES DEL FILTRO PARA
VOLVER A TOMAR MUESTRAS. DEFAULT= 2
transform_tolerance: 0.2 #TIEMPO DURANTE EL CUAL UNA TRANSFORMADA
ES VALIDA EN EL FUTURO
recovery_alpha_slow: 0.0 #0.0 INDICA QUE ESTA DESACTIVADO. DEFAULT=
0.0
recovery_alpha_fast: 0.0 #0.0 INDICA QUE ESTA DESACTIVADO. DEFAULT=
0.0
initial_pose_x: 0.0 #POSE EN X INICIAL PARA INICIAR EL FILTRO
DE PARTICULAS. DEFAULT= 0
initial_pose_y: 0.0 #POSE EN Y INICIAL PARA INICIAR EL FILTRO
DE PARTICULAS. DEFAULT= 0
initial_pose_a: 0.0 #GIRO EN Z INICIAL PARA INICIAR EL FILTRO
DE PARTICULAS. DEFAULT= 0

initial_cov_xx: 0.5*0.5 #COVARIANZA INICIAL DEL FILTRO (x*x).
DEFAULT= 0.5*0.5

```

```

initial_cov_yy: 0.5*0.5      #COVARIANZA INICIAL DEL FILTRO (y*y).
DEFAULT= 0.5*0.5
initial_cov_aa: 0.26*0.26  #COVARIANZA INICIAL DEL FILTRO (yaw*yaw).
DEFAULT= 0.26*0.26

gui_publish_rate: 9.0      #FRECUENCIA A LA QUE SE PUBLICAN LOS
RESULTADOS. DEFAULT= -1.0 (-1= DESACTIVADO)
save_pose_rate: 0.5       #FRECUENCIA A LA QUE ALMACENAR LA ULTIMA
POSE Y COVARIANZA. DEFAULT= 0.5

use_map_topic: false      #TRUE= AMCL SE SUSCRIBE AL TOPICO /map.
FALSE= AMCL USA EL SERVICIO DEL MAPA PARA SOLICITARLO.
first_map_only: false     #TRUE= AMCL OBTIENE EL PRIMER MAPA AL QUE
SE SUSCRIBE SIN TENER EN CUENTA SUS ACTUALIZACIONES.
selective_resampling: false #TRUE= REDUCE LA FRECUENCIA DE TOMAR
MUESTRAS DE PARTICULAS.

#LASER MODEL PARAMETERS

laser_min_range: 0.0      #DISTANCIA MINIMA A LA QUE TOMAR VALORES
DEL LASER. DEFAULT= -1 (-1= MINIMA DISTANCIA DEVUELTA POR EL LASER)
laser_max_range: 30.0     #DISTANCIA MAXIMA A LA QUE TOMAR VALORES
DEL LASER. DEFAULT= -1 (-1= MAXIMA DISTANCIA DEVUELTA POR EL LASER)
laser_max_beams: 30      #CUANTOS RAYOS USADOS DE CADA ESCANEO USAR
PARA ACTUALIZAR EL FILTRO. DEFAULT= 30
laser_z_hit: 0.95        #Mixture weight for the z_hit part of the
model.
laser_z_short: 0.1        #Mixture weight for the z_short part of
the model.
laser_z_max: 0.05        #Mixture weight for the z_max part of the
model.
laser_z_rand: 0.05       #Mixture weight for the z_rand part of the
model.
laser_sigma_hit: 0.2     #Standard deviation for Gaussian model
used in z_hit part of the model.
laser_lambda_short: 0.1   #Exponential decay parameter for z_short
part of model.
laser_likelihood_max_dist: 2.0 #Maximum distance to do
obstacle inflation on map, for use in likelihood_field model.
laser_model_type: "likelihood_field" #Which model to use, either
beam, likelihood_field, or likelihood_field_prob.

#ODOMETRY MODEL PARAMETERS

odom_model_type: diff     #Which model to use, either "diff",
"omni", "diff-corrected" or "omni-corrected".
odom_alpha1: 0.2         #Specifies the expected noise in
odometry's rotation estimate from the rotational component of the
robot's motion.
odom_alpha2: 0.2         #Specifies the expected noise in
odometry's rotation estimate from translational component of the
robot's motion.
odom_alpha3: 0.8         #Specifies the expected noise in
odometry's DEFAULT 0.2 translation estimate from the translational
component of the robot's motion.
odom_alpha4: 0.2         #Specifies the expected noise in
odometry's translation estimate from the rotational component of the
robot's motion.
odom_alpha5: 0.1         #Translation-related noise parameter (only
used if model is "omni").

```

```
odom_frame_id:      odom      #IDENTIFICADOR DEL FRAME ASOCIADO A LA
ODOMETRIA
base_frame_id:      base_link  #IDENTIFICADOR DEL FRAME ASOCIADO A LA
BASE DEL ROBOT
global_frame_id:    map        #IDENTIFICADOR DEL FRAME ASOCIADO AL MAPA
tf_broadcast:       true       #TRUE= AMCL PUBLICA LA TRANSFORMADA ENTRE
/map Y /odom. DEFAULT= TRUE
```

## Anexo X: *amcl\_p3at.launch*

```
<launch>
<!-- NODO AMCL -->
<node pkg="amcl" type="amcl" name="amcl" output="screen">
<!-- SIMULACIÓN -->
<!-- <remap from="/scan" to="/sim/scan"/> -->
<!-- <rosparam file="$(find
p3at_simulation)/params/amcl_p3at_sicklms200_params.yaml"
command="load" /> -->

<!-- REAL -->
<rosparam file="$(find
p3at_real)/params/amcl_p3at_sicklms200_params.yaml" command="load" />

</node>
</launch>
```

## Anexo XI: *costmap\_common\_params.yaml*

```
#COMMON CONFIGURATION FOR LOCAL_COSTMAP AND GLOBAL_COSTMAP

#PAQUETE costmap_2d

#PARAMETROS
footprint: [[0.254, 0.2485], [-0.254, 0.2485], [-0.254, -0.2485],
[0.254, -0.2485]]

transform_tolerance: 0.2      #RETRASO ADMISIBLE EN LAS TRANSFORMADAS
```

## Anexo XII: *global\_costmap\_params.yaml*

```
#CONFIGURATION FOR GLOBAL_COSTMAP
#LAS TABULACIONES (ESPACIOS) HAN DE SER IGUALES COMO SE MUESTRA

global_costmap:
  global_frame: map          #FRAME DONDE DEBE INICIARSE EL COSTMAP
  robot_base_frame: base_link #BASE_LINK OF THE ROBOT
  update_frequency: 2.0     #Hz A LA QUE SE ACTUALIZA EL COSTMAP
  publish_frequency: 2.0   #Hz A LA QUE SE PUBLICA LA INFORMACION
  static_map: true
  rolling_window: false    #DETERMINA SI EL COSTMAP SE MANTIENE
                           CENTRADO ALREDEDOR DEL ROBOT

#   width:                  20.0      #ANCHO EN METROS DEL COSTMAP
```

```

#   height:                20.0      #ALTO EN METROS DEL COSTMAP
#   resolution:            0.05      #RESOLUCION EN METROS/CELDA DEL COSTMAP
#   origin_x:              -10.0
#   origin_y:              -10.0

#OBSTACLE PARAMETERS #SE DEBE SEGUIR ESTE ORDEN
plugins:
  - {name: static_layer,          type: "costmap_2d::StaticLayer"}
  - {name: obstacle_layer,       type: "costmap_2d::VoxelLayer"}
  - {name: inflation_layer,      type: "costmap_2d::InflationLayer"}

static_layer:
  unknown_cost_value:    -1
  lethal_cost_threshold: 100
  map_topic:             /map
  fist_map_only:        true
  subscribe_to_updates: false
  track_unknown_space:  true
  use_maximum:          true
  trinary_costmap:      true

#   obstacle_layer:
#     observation_sources: sick_lms200
#     sick_lms200:
#       topic: /sim/scan    #/scan PARA p3at_real
#       sensor_frame: laser
#       observation_persistence: 0.0
#       expected_update_rate: 0.0
#       data_type: LaserScan
#       clearing: true
#       marking: true
#       max_obstacle_height: 0.5
#       min_obstacle_height: 0.0
#       obstacle_range: 6.0
#       raytrace_range: 7.0
#       inf_is_valid: true

inflation_layer:
  inflation_radius: 1.0
  cost_scaling_factor: 2.0

```

## Anexo XIII: *local\_costmap\_params.yaml*

```

#CONFIGURATION FOR LOCAL_COSTMAP

local_costmap:
  global_frame:      odom          #FRAME DONDE DEBE INICIARSE EL COSTMAP
  robot_base_frame: base_link     #BASE_LINK OF THE ROBOT
  update_frequency:  9.0          #Hz A LA QUE SE ACTUALIZA EL COSTMAP
  publish_frequency: 9.0          #Hz A LA QUE SE PUBLICA LA INFORMACION
  static_map:        false
  rolling_window:    true         #DETERMINA SI EL COSTMAP SE MANTIENE
  #CENTRADO ALREDEDOR DEL ROBOT
  width:             5.0          #ANCHO EN METROS DEL COSTMAP
  height:            5.0          #ALTO EN METROS DEL COSTMAP
  resolution:        0.05        #RESOLUCION EN METROS/CELDA DEL COSTMAP
  origin_x:          0.0
  origin_y:          0.0

#OBSTACLE PARAMETERS
plugins:
  - {name: obstacle_layer,          type: "costmap_2d::VoxelLayer"}
  - {name: inflation_layer,         type: "costmap_2d::InflationLayer"}

```

```

obstacle_layer:
  observation_sources: sick_lms200
  sick_lms200:
    topic: /scan #/sim/scan PARA p3at_simulation
    sensor_frame: laser
    observation_persistence: 0.0
    expected_update_rate: 0.0
    data_type: LaserScan
    clearing: true
    marking: true
    max_obstacle_height: 1.0
    min_obstacle_height: 0.0
    obstacle_range: 5.0
    raytrace_range: 5.0
    inf_is_valid: true

inflation_layer:
  inflation_radius: 0.1
  cost_scaling_factor: 1.0

```

## Anexo XIV: *global\_planner\_params.yaml*

```

#Base global Planner Params (navfn, global_planner, carrot_planner)
#PARAMETROS PARA EL BASE_GLOBAL_PLANNER
GlobalPlanner:

  #PARAMETROS COMUNES ENTRE TODOS LOS PLANIFICADORES
  allow_unknown: true # PERMITE AL PLANIFICADOR REALIZAR
  TRAYECTORIAS A TRAVES DE ESPACIOS DESCONOCIDOS. DEFAULT: TRUE

  default_tolerance: 0.1 # CREA UNA TRAYECTORIA CON ESTA
  TOLERANCIA AL DESTINO. DEFAULT: 0.0

  visualize_potential: true # PERMITE VISUALIZAR EL ÁREA
  PROCESADA POR EL PLANIFICADOR. DEFAULT: FALSE

  #GlobalPlanner. NOMBRE DEL PLUGIN PARA move_base
  (global_planner/GlobalPlanner)

  use_dijkstra: true # TRUE USA ALGORITMO DIJKSTRA.
  FALSE USA ALGORITMO A*. DEFAULT: TRUE

  use_quadratic: true # USA APROXIMACIÓN CUADRÁTICA.
  DEFAULT: TRUE

  use_grid_path: false # CREA UNA TRAYECTORIA QUE SIGUE
  LÍNEAS CUADRICULARES. DEFAULT: FALSE

  old_navfn_behavior: false # REALIZA UNA TRAYECTORIA IGUAL A
  LA DE navfn. DEFAULT: FALSE

  lethal_cost: 253 # VALOR QUE SE CONSIDERA COMO
  CELDA OCUPADA. DEFAULT: 253

  neutral_cost: 66 # VALOR QUE SE CONSIDERA COMO
  CELDA LIBRE. DEFAULT: 50

  cost_factor: 0.55 # VALOR QUE MULTIPLICA EL VALOR DE
  LAS CELDAS DEL COSTMAP. DEFAULT: 3

```

```

    publish_potential: true          # PUBLICA EL MAPA POTENCIAL.
DEFAULT: TURE
    orientation_mode: 0              # None=0, Forward=1,
Interpolate=2, ForwardThenInterpolate=3, Backward=4, Leftward=5,
Rightward=6. DEFAULT: 0

    outline_map: true               # DESCRIBE EL GLOBAL COSTMAP CON
OBJETOS LETALES. DEFAULT: TRUE

    orientation_window_size: 1      # VALOR QUE INDICA LA DISTANCIA A
LA CUAL CALCULAR LA ORIENTACIÓN DE CADA PUNTO SEGÚN ORIENTATION_MODE.
DEFAULT: 1

#NavfnROS. NOMBRE DEL PLUGIN PARA move_base (navfn/NavfnROS)
#   planner_window_x: 0.0          # TAMAÑO EN X DE VENTANA PARA
RESTRINGIR EL PLANIFICADOR. DEFAULT: 0.0

#   planner_window_y: 0.0          # TAMAÑO EN Y DE VENTANA PARA
RESTRINGIR EL PLANIFICADOR. DEFAULT: 0.0

#CarrotPlaner. NOMBRE DEL PLUGIN PARA move_base
(carrot_planner/CarrotPlanner)
#   step_size:                     # DISTANCIA A LA QUE COMPROBAR SI
SE HA ALCANZADO EL DESTINO. DEFAULT: RESOLUCIÓN DEL MAPA

#   min_dist_from_robot: 1         # DISTANCIA MINIMA ENTRE EL ROBOT
Y EL DESTINO PARA LA CUAL SE ENVIARA DICHO DESTINO AL PLANIFICADOR.
DEFAULT: 0.10

```

## Anexo XV: *base\_local\_planner\_params.yaml*

```
#CONFIGURATION FILE FOR BASE LOCAL PLANNER
```

### TrajectoryPlannerROS:

```

#PARAMETROS DEL ROBOT
#ACELERACIONES
acc_lim_x: 0.4          #ACELERACIÓN MAXIMA EN X (m/s2)
acc_lim_y: 0.0          #ACELERACIÓN MAXIMA EN Y (m/s2)
acc_lim_theta: 2.0      #ACELERACIÓN MAXIMA DE ROTACIÓN EN Z (rad/s2)

#VELOCIDADES
max_vel_x: 0.1          #VELOCIDAD MAXIMA EN X (m/s)
min_vel_x: 0.1          #VELOCIDAD MINIMA EN X (m/s)
max_vel_y: 0.0          #VELOCIDAD MAXIMA EN Y (m/s)
min_vel_y: 0.0          #VELOCIDAD MINIMA EN Y (m/s)
max_vel_theta: 0.3      #VELOCIDAD ANGULAR MAXIMA (rad/s)
min_vel_theta: -0.3     #VELOCIDAD ANGULAR MINIMA (rad/s)
min_in_place_vel_theta: 0.6 #VELOCIDAD ANGULAR MINIMA PARA GIRO
SOBRE SÍ MISMO (rad/s)

holonomic_robot: false  #INDICA SI EL ROBOT ES HOLONOMICO O NO

#PARAMETROS DE LA TOLERANCIA AL DESTINO
yaw_goal_tolerance: 0.4 #TOLERANCIA ANGULAR DE ORIENTACIÓN (rad)
0.17rad ~ 10°

```

```

xy_goal_tolerance: 0.1 #TOLERANCIA DISTANCIA A PUNTO DE DESTINO (m)

latch_xy_goal_tolerance: false #ROTACION SOBRE SÍ MISMO CUANDO LLEGA
AL DESTINO

#PARAMETROS DEL - DWA - LOCAL PLANNER
#PARAMETROS DE FORWARD-SIMULATION
sim_time: 2 #TIEMPO DE FORWARD-SIMULATION
sim_granularity: 0.025 #CADA CUANTO COMPROBAR LOS PUNTOS THE UNA
TRAYECTORIA (m)

vx_samples: 20 #Nº DE MUESTRAS DE VELOCIDAD PARA FORWARD-
SIMULATION. DEFAULT: 3

vtheta_samples: 40 #Nº DE MUESTRAS DE VELOCIDAD DE ROTACIÓN
PARA FORWARD-SIMULATION. DEFAULT: 20

controller_frequency: 20 #FRECUENCIA LA QUE SE ACTUALIZAN LOS
COMANDOS DE VELOCIDAD. DEFAULT: 20

#PARAMETROS DEL TRAJECTORY-SCORING
meter_scoring: true #TRUE= FIDELIDAD EN METROS. FALSE= FIDELIDAD
EN PORCENTAJE.

pdist_scale: 0.5 #FIDELIDAD A LA TRAYECTORIA GLOBAL EN RANGO(0-5)
gdist_scale: 0.9 #FIDELIDAD AL DESTINO LOCAL EN RANGO(0-5)
occdist_scale: 0.01 #PESO PARA INTENTAR ESQUIVAR OBJETOS

#PARAMETROS PARA PREVENIR LA OSCILACION
oscillation_reset_dist: 0.05 #DISTANCIA QUE HA DE RECORRER EL ROBOT
ANTES DE QUE SE RESETEE LA OSCILACION. DEFAULT: 0.05

#PARAMETROS DEL GLOBAL PLAN
prune_plan: true #INDICA SI SE BORRA O NO LA
TRAYECTORIA PLANEADA A MEDIDA QUE EL ROBOT LA SIGUE. DEFAULT: TRUE

```

## Anexo XVI: *recovery\_behaviors\_params.yaml*

```

recovery_behaviors:
  #PRIMERO SE DECLARAN LOS COMPORTAMIENTOS EXISTENTES
  - name: 'conservative_reset'
    type: 'clear_costmap_recovery/ClearCostmapRecovery'
  - name: 'aggressive_reset'
    type: 'clear_costmap_recovery/ClearCostmapRecovery'
  - name: 'rotate_recovery_1'
    type: 'rotate_recovery/RotateRecovery'
  - name: 'rotate_recovery_2'
    type: 'rotate_recovery/RotateRecovery'

  #SEGUNDO, SE DEFINEN ESTOS COMPORTAMIENTOS
conservative_reset:
  reset_distance: 2.0 #LONGITUD DEL LADO DE UN CUADRADO CENTRADO
  EN EL ROBOT, FUERA DEL CUAL SE ELIMINAN LOS OBSTACULOS DEL COSTMAP.

aggressive_reset:
  reset_distance: 1.0

```



```
rotate_recovery_1:
  sim_granularity: 0.017 #CADA CUANTO COMPROBAR SI HAY OBSTACULOS
(rad).
```

```
rotate_recovery_2:
  sim_granularity: 0.017
```

## Anexo XVII: *move\_base\_params.yaml*

```
#move_base PARAMS

  base_global_planner: 'global_planner/GlobalPlanner' #INDICA EL
PLANIFICADOR GLOBALR A USAR

  base_local_planner: 'base_local_planner/TrajectoryPlannerROS'
#INDICA EL PLANIFICADOR LOCAL A USAR

#LOS RECOVERY_BEHAVIORS SE ESPECIFICAN EN SU PROPIO YAML

  controller_frequency: 9.0 #FRECUENCIA A LA QUE SE
ACTUALIZAN LOS COMANDOS DE VELOCIDAD. DEFAULT: 20

  planner_patience: 5.0 #TIEMPO MÁXIMO QUE ESPERA EL
SISTEMA INTENTANDO ENCONTRAR UNA TRAYECTORIA AL DESTINO ANTES DE
EJECUTAR RECOVERY_BEHAVIORS. DEFAULT: 5.0

  controller_patience: 15.0 #TIEMPO MÁXIMO QUE ESPERA EL
SISTEMA SIN RECIBIR COMANDOS DE VELOCIDAD ANTES DE EJECUTAR
RECOVERY_BEHAVIORS. DEFAULT: 15

  conservative_reset_dist: 3.0 #LA DISTANCIA, EN METROS, MÁS
ALLÁ DEL ROBOT PARTIR DE LA CUAL SE ELIMINARÁN LOS OBSTÁCULOS DEL
COSTMAP. DEFAULT: 3

  recovery_behavior_enabled: true #VALOR BOOLEANO QUE ACTIVA O
DESACTIVA LA UTILIZACIÓN DE RECOVERY_BEHAVIORS. DEFAULT: TRUE

  clearing_rotation_allowed: true #VALOR BOOLEANO QUE ACTIVA O
DESACTIVA LA UTILIZACIÓN DE ROTATE_RECOVERY. DEFAULT: TRUE

  shutdown_costmaps: false #VALOR BOOLEANO QUE DETERMINA
SI LOS COSTMAPS PERMANECEN DESACTIVADOS CUANDO MOVE_BASE ESTÁ
INACTIVO. DEFAULT: FALSE

  oscillation_timeout: 0.0 #TIEMPO EN SEGUNDOS QUE EL
SISTEMA PERMITE OSCILACIÓN ANTES DE EJECUTAR RECOVERY_BEHAVIORS.
DEFAULT: 0.0

  oscillation_distance: 0.5 #DISTANCIA EN METROS QUE HA DE
RECORRER EL ROBOT PARA QUE EL SISTEMA CONSIDERE QUE NO ESTÁ OSCILANDO.
DEFAULT: 0.5

  planner_frequency: 1.0 #FRECUENCIA A LA QUE SE
ACTUALIZA EL PLANIFICADOR GLOBAL. DEFAULT: 0.0

  max_planning_retries: -1 #VECES QUE EL SISTEMA INTENTA
REALIZAR PLAN GLOBAL ANTES DE INICIAR RECOVERY_BEHAVIORS. -1 REALIZA
INFINITOS INTENTOS.
```

## Anexo XVIII: *move\_base\_p3at.launch*

```

<launch>

<!-- run MOVE BASE -->
<node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">

    <!-- CAMBIAMOS EL TOPICO AL QUE PUBLICA MOVE BASE -->
    <remap from="/cmd_vel" to="/RosAria/cmd_vel"/>

    <!-- CARGAMOS ARCHIVOS DE PARAMETROS -->
<rosparam file="$(find p3at_real)/params/costmap_common_params.yaml"
command="load" ns="global_costmap" />

<rosparam file="$(find p3at_real)/params/costmap_common_params.yaml"
command="load" ns="local_costmap" />

<rosparam file="$(find p3at_real)/params/local_costmap_params.yaml"
command="load" />

<rosparam file="$(find p3at_real)/params/global_costmap_params.yaml"
command="load" />

<rosparam file="$(find p3at_real)/params/base_local_planner_params.yaml"
command="load" />

<rosparam file="$(find p3at_real)/params/global_planner_params.yaml"
command="load" />

<rosparam file="$(find p3at_real)/params/recovery_behaviors.yaml"
command="load" />

<rosparam file="$(find p3at_real)/params/move_base_params.yaml"
command="load" />

    </node>

</launch>

```

## Anexo XIX: *my\_p3at.urdf*

```

<?xml version="1.0" ?>
<!-- ===== -->
<!-- |   EDITING THIS FILE BY HAND IS NOT RECOMMENDED   |-->
<!-- ===== -->
<robot name="pioneer3at"
xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema/#controller"
xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema/#interface"
xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor" xmlns:xacro="http://ros.org/wiki/xacro">
    <!-- For pointers on inertial and gazebo-related parameters see
    *
    http://wiki.ros.org/urdf/Tutorials/Adding%20Physical%20and%20Collision
    %20Properties%20to%20a%20URDF%20Model
    * http://answers.gazebosim.org/question/4372/the-inertia-
    matrix-explained/

```

```

*
http://gazebosim.org/tutorials?tut=inertia&cat=build_robot, and
* http://gazebosim.org/tutorials/?tut=ros_urdf
*
http://en.wikipedia.org/wiki/List_of_moment_of_inertia_tensors
-->

<!-- ===== -->
<!-- | PRIMERA PARTE: DESCRIPCION DE LOS ELEMENTOS (LINKS DEL ROBOT)
|-->
<!-- ===== -->

<!--LINKS DESCRIPTIONS-->
<!-- Chassis -->
<link name="base_link">
  <inertial>
    <mass value="21.3"/>
    <!-- P3AT_12 kg + x3 batteries_2 kg + PC_3kg - x4 wheels_1.4 kg
- top_plate_0.1 kg-->
    <origin xyz="-0.1 0 0.177"/>
    <inertia ixx="0.3338" ixy="0.0" ixz="0.0" iyy="0.4783" iyz="0.0"
izz="0.3338"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0.177"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/chassis.
stl"/>
      </geometry>
      <material name="ChassisRed">
        <color rgba="0.851 0.0 0.0 1.0"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0.177"/>
      <geometry>
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/chassis.
stl"/>
      </geometry>
    </collision>
  </link>
  <!-- Top_Plate -->
  <link name="top_plate">
    <inertial>
      <mass value="0.1"/>
      <origin xyz="-0.025 0 -0.223"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry name="pioneer_geom">
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/top.stl"
/>
        </geometry>
        <material name="TopBlack">
          <color rgba="0.038 0.038 0.038 1.0"/>
        </material>

```

```

    </visual>
  </link>
  <!--Font_Sonar-->
  <link name="front_sonar">
    <inertial>
      <mass value="0.1"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry name="pioneer_geom">
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/front_so
nar.stl"/>
      </geometry>
      <material name="SonarYellow">
        <color rgba="0.715 0.583 0.210 1.0"/>
      </material>
    </visual>
  </link>
  <!--Back_Sonar-->
  <link name="back_sonar">
    <inertial>
      <mass value="0.1"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry name="pioneer_geom">
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/back_son
ar.stl"/>
      </geometry>
      <material name="SonarYellow">
        <color rgba="0.715 0.583 0.210 1.0"/>
      </material>
    </visual>
  </link>
  <!--case_laser-->
  <link name="case_laser">
    <inertial>
      <mass value="0.1"/>
      <!--kg-->
      <origin xyz="0 0 0"/>
      <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0"
izz="0.0"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry name="pioneer_geom">
        <mesh
filename="package://p3at_simulation/models/meshes/CASE_LASER.stl"/>
      </geometry>
      <material name="SickBlue">
        <color rgba="0.0 0.170 0.255 1.0"/>
      </material>
    </visual>

```

```

</link>
<!--base_laser-->
<link name="laser">
  <inertial>
    <mass value="0.1"/>
    <!--kg-->
    <origin xyz="0 0 0"/>
    <inertia ixx="0.0" ixy="0.0" ixz="0.0" iyy="0.0" iyz="0.0"
izz="0.0"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://p3at_simulation/models/meshes/BASE_LASER.stl"/>
    </geometry>
    <material name="SickBlack">
      <color rgba="0.0 0.0 0.0 1.0"/>
    </material>
  </visual>
</link>

<!-- ===== -->
<!-- | SEGUNDA PARTE: DESCRIPCION DE LAS UNIONES ENTRE ELEMENTOS DEL
ROBOT (JOINTS) | -->
<!-- ===== -->

<!--JOINT DESCRIPTIONS-->
<!-- base_link + top_plate -->
<joint name="base_top_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0.003 0 0.274"/>
  <parent link="base_link"/>
  <child link="top_plate"/>
</joint>
<!-- base_link + front_sonar-->
<joint name="base_front_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0.193 0 0.25"/>
  <parent link="base_link"/>
  <child link="front_sonar"/>
</joint>
<!--base_link + back_sonar-->
<joint name="base_back_joint" type="fixed">
  <origin rpy="0 0 0" xyz="-0.187 0 0.247"/>
  <parent link="base_link"/>
  <child link="back_sonar"/>
</joint>
<!-- top_plate + case_laser -->
<joint name="top_laser_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0.127 0 0.0975"/>
  <parent link="top_plate"/>
  <child link="case_laser"/>
</joint>
<!-- case_laser + base_laser -->
<joint name="case_base_laser_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0.025 0 -0.03"/>
  <parent link="case_laser"/>
  <child link="laser"/>
</joint>

```

```

<!-- ===== -->
<!-- | TERCERA PARTE: DESCRIPCION DE LOS MATERIALES DE CADA ELEMENTO
PARA GAZEBO      |-->
<!-- ===== -->

<!--GAZEBO MATERIAL REFERENCES-->
<!--base_link-->
<gazebo reference="base_link">
  <material value="Gazebo/Red"/>
</gazebo>
<!--top_plate-->
<gazebo reference="top_plate">
  <material value="Gazebo/Black"/>
</gazebo>
<!--front_sonar-->
<gazebo reference="front_sonar">
  <material value="Gazebo/Yellow"/>
</gazebo>
<!--back_sonar-->
<gazebo reference="back_sonar">
  <material value="Gazebo/Yellow"/>
</gazebo>
<!--case_laser-->
<gazebo reference="case_laser">
  <material value="Gazebo/Blue"/>
</gazebo>
<!--base_laser-->
<gazebo reference="laser">
  <material value="Gazebo/Black "/>
</gazebo>
<!--usb_cam-->
<gazebo reference="usb_cam">
  <material value="Gazebo/Orange "/>
</gazebo>

<!-- ===== -->
<!-- | CUARTA PARTE: DESCRIPCION DE LOS ELEMENTOS RUEDAS, EJES Y
ESPACIADORES      |-->
<!-- ===== -->

<!-- ===== -->
<!-- | ELEMENTOS DE LA IZQUIERDA
|-->
<!-- ===== -->

<!--LINKS DESCRIPTIONS-->
<!-- front_axles -->
<link name="p3at_front_left_axle">
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/axle.stl
"/>
    </geometry>

```

```

    <material name="AxleGrey">
      <color rgba="0.5 0.5 0.5 1"/>
    </material>
  </visual>
</link>
<!--front_hubs-->
<link name="p3at_front_left_hub">
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
  izz="1.0"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/left_hub
cap.stl"/>
    </geometry>
    <material name="HubcapYellow">
      <color rgba="1.0 0.811 0.151 1.0"/>
    </material>
  </visual>
</link>
<!-- front_wheels -->
<link name="p3at_front_left_wheel">
  <inertial>
    <mass value="1.2"/>
    <!-- 1.4 kg - axle 0.1 kg - hub 0.1 kg -->
    <origin xyz="0 0 0"/>
    <inertia ixx="0.012411765597" ixy="0" ixz="0"
  iyy="0.015218160428" iyz="0" izz="0.011763977943"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
1"/>
    </geometry>
    <material name="WheelBlack">
      <color rgba="0.117 0.117 0.117 1"/>
    </material>
  </visual>
  <collision>
    <origin rpy="-1.57079635 0 0" xyz="0 0 0"/>
    <geometry>
      <!--mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
1"/-->
      <cylinder length="0.075" radius="0.111"/>
    </geometry>
  </collision>
</link>
<!-- back_axles -->
<link name="p3at_back_left_axle">
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>

```

```

    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
  izz="1.0"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/axle.stl
"/>
      </geometry>
      <material name="AxleGrey">
        <color rgba="0.5 0.5 0.5 1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/axle.stl
"/>
      </geometry>
    </collision>
  </link>
  <!-- back_hubs -->
  <link name="p3at_back_left_hub">
    <inertial>
      <mass value="0.1"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
  izz="1.0"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry name="pioneer_geom">
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/left_hub
cap.stl"/>
      </geometry>
      <material name="HubcapYellow"/>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/left_hub
cap.stl"/>
      </geometry>
    </collision>
  </link>
  <!-- back_wheels -->
  <link name="p3at_back_left_wheel">
    <inertial>
      <mass value="1.2"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="0.012411765597" ixy="0" ixz="0"
  iyy="0.015218160428" iyz="0" izz="0.011763977943"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry name="pioneer_geom">

```



```

    <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
1"/>
    </geometry>
    <material name="WheelBlack"/>
  </visual>
  <collision>
    <origin rpy="-1.57079635 0 0" xyz="0 0 0"/>
    <geometry>
      <!--mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
1"/-->
      <cylinder length="0.075" radius="0.111"/>
    </geometry>
  </collision>
</link>

<!--JOINT DESCRIPTIONS-->
<!-- base_link + front_axles -->
<joint name="base_front_left_axle_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0.135 0.156 0.111"/>
  <parent link="base_link"/>
  <child link="p3at_front_left_axle"/>
</joint>
<!-- front_axles + front_hubs -->
<joint name="base_front_left_hub_joint" type="fixed">
  <origin rpy="0 0 0" xyz="0 0.041 0"/>
  <parent link="p3at_front_left_axle"/>
  <child link="p3at_front_left_hub"/>
</joint>
<!-- front_hub + wheels -->
<joint name="p3at_front_left_wheel_joint" type="continuous">
  <axis xyz="0 1 0"/>
  <anchor xyz="0 0 0"/>
  <limit effort="100" velocity="6.3"/>
  <joint_properties damping="0.7"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <parent link="p3at_front_left_hub"/>
  <child link="p3at_front_left_wheel"/>
</joint>
<!-- base_link + back_axles -->
<joint name="p3at_back_left_axle_joint" type="fixed">
  <origin rpy="0 0 0" xyz="-0.134 0.156 0.111"/>
  <parent link="base_link"/>
  <child link="p3at_back_left_axle"/>
</joint>
<!-- back_axles + back_hubs -->
<joint name="p3at_back_left_hub_joint" type="fixed">
  <origin rpy="0 0 0" xyz="-0 0.041 0"/>
  <parent link="p3at_back_left_axle"/>
  <child link="p3at_back_left_hub"/>
</joint>
<!-- back_hubs + back_wheels -->
<joint name="p3at_back_left_wheel_joint" type="continuous">
  <axis xyz="0 1 0"/>
  <anchor xyz="0 0 0"/>
  <limit effort="100" velocity="6.3"/>
  <joint_properties damping="0.7"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <parent link="p3at_back_left_hub"/>
  <child link="p3at_back_left_wheel"/>

```

```

</joint>

<!--GAZEBO MATERIAL REFERENCES-->
<!--front_axles-->
<gazebo reference="p3at_front_left_axle">
  <material value="Gazebo/Grey"/>
</gazebo>
<!--front_hubs-->
<gazebo reference="p3at_front_left_hub">
  <material value="Gazebo/Yellow"/>
</gazebo>
<!--front_wheels-->
<gazebo reference="p3at_front_left_wheel">
  <material value="Gazebo/Black"/>
</gazebo>
<!-- back_axles -->
<gazebo reference="p3at_back_left_axle">
  <material value="Gazebo/Grey"/>
</gazebo>
<!-- back_hubs -->
<gazebo reference="p3at_back_left_hub">
  <material value="Gazebo/Yellow"/>
</gazebo>
<!-- back_wheels -->
<gazebo reference="p3at_back_left_wheel">
  <material value="Gazebo/Black"/>
</gazebo>

<!-- ===== -->
<!-- | ELEMENTOS DE LA DERECHA -->
<!-- | -->
<!-- ===== -->

<!--LINKS DESCRIPTIONS-->
<!-- front_axles -->
<link name="p3at_front_right_axle">
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
  izz="1.0"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/axle.stl
"/>
      </geometry>
      <material name="AxleGrey">
        <color rgba="0.5 0.5 0.5 1"/>
      </material>
    </visual>
  </link>
<!--front_hubs-->
<link name="p3at_front_right_hub">
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
  izz="1.0"/>

```

```

</inertial>
<visual>
  <origin rpy="0 0 0" xyz="0 0 0"/>
  <geometry name="pioneer_geom">
    <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/right_hu
bcap.stl"/>
    </geometry>
    <material name="HubcapYellow">
      <color rgba="1.0 0.811 0.151 1.0"/>
    </material>
  </visual>
</link>
<!-- front_wheels -->
<link name="p3at_front_right_wheel">
  <inertial>
    <mass value="1.2"/>
    <!-- 1.4 kg - axle 0.1 kg - hub 0.1 kg -->
    <origin xyz="0 0 0"/>
    <inertia ixx="0.012411765597" ixy="0" ixz="0"
iyy="0.015218160428" iyz="0" izz="0.011763977943"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
1"/>
    </geometry>
    <material name="WheelBlack">
      <color rgba="0.117 0.117 0.117 1"/>
    </material>
  </visual>
  <collision>
    <origin rpy="-1.57079635 0 0" xyz="0 0 0"/>
    <geometry>
      <!--mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
1"/-->
      <cylinder length="0.075" radius="0.111"/>
    </geometry>
  </collision>
</link>
<!-- back_axles -->
<link name="p3at_back_right_axle">
  <inertial>
    <mass value="0.1"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry name="pioneer_geom">
      <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/axle.stl
"/>
    </geometry>
    <material name="AxleGrey">
      <color rgba="0.5 0.5 0.5 1"/>
    </material>

```

```

    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/axle.stl
"/>
        </geometry>
      </collision>
    </link>
    <!-- back_hubs -->
    <link name="p3at_back_right_hub">
      <inertial>
        <mass value="0.1"/>
        <origin xyz="0 0 0"/>
        <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
izz="1.0"/>
      </inertial>
      <visual>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <geometry name="pioneer_geom">
          <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/right_hu
bcap.stl"/>
          </geometry>
          <material name="HubcapYellow"/>
        </visual>
        <collision>
          <origin rpy="0 0 0" xyz="0 0 0"/>
          <geometry>
            <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/right_hu
bcap.stl"/>
            </geometry>
          </collision>
        </link>
        <!-- back_wheels -->
        <link name="p3at_back_right_wheel">
          <inertial>
            <mass value="1.2"/>
            <origin xyz="0 0 0"/>
            <inertia ixx="0.012411765597" ixy="0" ixz="0"
iyy="0.015218160428" iyz="0" izz="0.011763977943"/>
          </inertial>
          <visual>
            <origin rpy="0 0 0" xyz="0 0 0"/>
            <geometry name="pioneer_geom">
              <mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
l"/>
              </geometry>
              <material name="WheelBlack"/>
            </visual>
            <collision>
              <origin rpy="-1.57079635 0 0" xyz="0 0 0"/>
              <geometry>
                <!--mesh
filename="package://amr_robots_description/meshes/p3at_meshes/wheel.st
l"/-->
                <cylinder length="0.075" radius="0.111"/>
              </geometry>

```

```

    </collision>
  </link>

  <!--JOINT DESCRIPTIONS-->
  <!-- base_link + front_axles -->
  <joint name="base_front_right_axle_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0.135 -0.156 0.111"/>
    <parent link="base_link"/>
    <child link="p3at_front_right_axle"/>
  </joint>
  <!-- front_axles + front_hubs -->
  <joint name="base_front_right_hub_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0 -0.041 0"/>
    <parent link="p3at_front_right_axle"/>
    <child link="p3at_front_right_hub"/>
  </joint>
  <!-- front_hub + wheels -->
  <joint name="p3at_front_right_wheel_joint" type="continuous">
    <axis xyz="0 1 0"/>
    <anchor xyz="0 0 0"/>
    <limit effort="100" velocity="6.3"/>
    <joint_properties damping="0.7"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <parent link="p3at_front_right_hub"/>
    <child link="p3at_front_right_wheel"/>
  </joint>
  <!-- base_link + back_axles -->
  <joint name="p3at_back_right_axle_joint" type="fixed">
    <origin rpy="0 0 0" xyz="-0.134 -0.156 0.111"/>
    <parent link="base_link"/>
    <child link="p3at_back_right_axle"/>
  </joint>
  <!-- back_axles + back_hubs -->
  <joint name="p3at_back_right_hub_joint" type="fixed">
    <origin rpy="0 0 0" xyz="-0 -0.041 0"/>
    <parent link="p3at_back_right_axle"/>
    <child link="p3at_back_right_hub"/>
  </joint>
  <!-- back_hubs + back_wheels -->
  <joint name="p3at_back_right_wheel_joint" type="continuous">
    <axis xyz="0 1 0"/>
    <anchor xyz="0 0 0"/>
    <limit effort="100" velocity="6.3"/>
    <joint_properties damping="0.7"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <parent link="p3at_back_right_hub"/>
    <child link="p3at_back_right_wheel"/>
  </joint>

  <!--GAZEBO MATERIAL REFERENCES-->
  <!--front_axles-->
  <gazebo reference="p3at_front_right_axle">
    <material value="Gazebo/Grey"/>
  </gazebo>
  <!--front_hubs-->
  <gazebo reference="p3at_front_right_hub">
    <material value="Gazebo/Yellow"/>
  </gazebo>
  <!--front_wheels-->
  <gazebo reference="p3at_front_right_wheel">
    <material value="Gazebo/Black"/>
  </gazebo>

```

```

</gazebo>
<!-- back_axles -->
<gazebo reference="p3at_back_right_axle">
  <material value="Gazebo/Grey"/>
</gazebo>
<!-- back_hubs -->
<gazebo reference="p3at_back_right_hub">
  <material value="Gazebo/Yellow"/>
</gazebo>
<!-- back_wheels -->
<gazebo reference="p3at_back_right_wheel">
  <material value="Gazebo/Black"/>
</gazebo>

<!-- ===== -->
<!-- | QUINTA PARTE: DESCRIPCION DEL LASER PLUGIN PARA GAZEBO
|-->
<!-- ===== -->

<!-- RAY SENSOR FOR GAZEBO -->
<gazebo reference="laser">
  <sensor name="sick_lms200" type="gpu_ray">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>75</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>180</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.1</min>
        <max>80.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo
laser
achieving "+-30mm" accuracy at range < 10m. A mean of
0.0m and
stddev of 0.01m will put 99.7% of samples within 0.03m
of the true
reading. -->
        <mean>0.00</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin filename="libgazebo_ros_gpu_laser.so"
name="gazebo_ros_head_hokuyo_controller">
      <topicName>/sim/scan</topicName>
      <frameName>laser</frameName>
    </plugin>
  </sensor>
</gazebo>
<!-- SENSOR DE RAYOS, SICK LMS 200 -->
<sensor name="lms200" update_rate="10">

```

```

    <parent link="laser"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <ray>
      <horizontal max_angle="1.5708" min_angle="-1.5708"
resolution="1" samples="180"/>
    </ray>
  </sensor>

<!-- ===== -->
<!-- | SEXTA PARTE: DESCRIPCION DE LAS PROPIEDADES DINAMICAS DE LAS
RUEDAS | -->
<!-- ===== -->

<!--PROPERTIES OF LEFT WHEEL "MOTORS"-->
<!-- see http://gazebosim.org/tutorials/?tut=ros_urdf -->
<gazebo reference="p3at_back_left_wheel">
  <kp> 1000000.0 </kp>
  <!-- kp and kd for rubber -->
  <kd> 100.0 </kd>
  <mu1> 1.0 </mu1>
  <!-- was 10 -->
  <mu2> 1.0 </mu2>
  <!-- how to get these into <surface><friction><ode>... ? slip1
0.5 /slip1 slip2 0 /slip2 -->
  <!-- fdir1 0 1 0 /fdir1 -->
  <!-- see http://github.com/MobileRobots/amr-ros-config/issues/6 --
>
  <maxVel> 0.00 </maxVel>
  <minDepth> 0.001 </minDepth>
  <material value="Gazebo/Black"/>
</gazebo>
<gazebo reference="p3at_front_left_wheel">
  <kp> 1000000.0 </kp>
  <!-- kp and kd for rubber -->
  <kd> 100.0 </kd>
  <mu1> 1.0 </mu1>
  <!-- was 10 -->
  <mu2> 1.0 </mu2>
  <!-- how to get these into <surface><friction><ode>... ? slip1
0.5 /slip1 slip2 0 /slip2 -->
  <!-- fdir1 0 1 0 /fdir1 -->
  <!-- see http://github.com/MobileRobots/amr-ros-config/issues/6 --
>
  <maxVel> 0.00 </maxVel>
  <minDepth> 0.001 </minDepth>
  <material value="Gazebo/Black"/>
</gazebo>

<!--PROPERTIES OF RIGHT WHEEL "MOTORS"-->
<!-- see http://gazebosim.org/tutorials/?tut=ros_urdf -->
<gazebo reference="p3at_back_right_wheel">
  <kp> 1000000.0 </kp>
  <!-- kp and kd for rubber -->
  <kd> 100.0 </kd>
  <mu1> 1.0 </mu1>
  <!-- was 10 -->
  <mu2> 1.0 </mu2>
  <!-- how to get these into <surface><friction><ode>... ? slip1
0.5 /slip1 slip2 0 /slip2 -->
  <!-- fdir1 0 1 0 /fdir1 -->

```

```

    <!-- see http://github.com/MobileRobots/amr-ros-config/issues/6 --
>
    <maxVel>      0.00      </maxVel>
    <minDepth>    0.001 </minDepth>
    <material value="Gazebo/Black"/>
  </gazebo>
  <gazebo reference="p3at_front_right_wheel">
    <kp>      1000000.0 </kp>
    <!-- kp and kd for rubber -->
    <kd>      100.0      </kd>
    <mu1>     1.0        </mu1>
    <!-- was 10 -->
    <mu2>     1.0        </mu2>
    <!-- how to get these into <surface><friction><ode>... ?      slip1
0.5 /slip1 slip2 0 /slip2 -->
    <!-- fdir1 0 1 0 /fdir1 -->
    <!-- see http://github.com/MobileRobots/amr-ros-config/issues/6 --
>
    <maxVel>      0.00      </maxVel>
    <minDepth>    0.001 </minDepth>
    <material value="Gazebo/Black"/>
  </gazebo>

<!-- ===== -->
<!-- | SEPTIMA PARTE: DESCRIPCION DEL SKID_STEER_DRIVE PLUGIN PARA
GAZEBO      |-->
<!-- ===== -->

  <gazebo>
    <plugin filename="libgazebo_ros_skid_steer_drive.so"
name="skid_steer_drive_controller">
      <updateRate>      100.0
      </updateRate>
      <robotNamespace>  sim
      </robotNamespace>
      <leftFrontJoint>  p3at_front_left_wheel_joint
      </leftFrontJoint>
      <rightFrontJoint> p3at_front_right_wheel_joint
      </rightFrontJoint>
      <leftRearJoint>    p3at_back_left_wheel_joint
      </leftRearJoint>
      <rightRearJoint>  p3at_back_right_wheel_joint
      </rightRearJoint>
      <wheelSeparation> 0.4
      </wheelSeparation>
      <wheelDiameter>   0.215
      </wheelDiameter>
      <robotBaseFrame>  base_link
      </robotBaseFrame>
      <MaxForce>        5
      </MaxForce>
      <torque>          15
      </torque>
      <commandTopic>    cmd_vel
      </commandTopic>
      <odometryTopic>   odom
      </odometryTopic>
      <odometryFrame>   odom
      </odometryFrame>
      <broadcastTF>     1
      </broadcastTF>

```



```

    </plugin>
  </gazebo>
</robot>

```

## Anexo XX: *robot\_p3at\_empty\_world\_simulated.launch*

```

<launch>

  <!-- INDICAMOS DONDE ESTA LA CARPETA CON NUESTROS MODELOS -->
  <env name="GAZEBO_MODEL_PATH"
    value="$GAZEBO_MODEL_PATH:$(find p3at_simulation)/models/urdf"
  />
  <env name="GAZEBO_RESOURCE_PATH"
    value="$GAZEBO_RESOURCE_PATH:$(find
p3at_simulation)/models/urdf" />

  <!-- CREAMOS ARGUMENTOS PARA CONFIGURAR LA SIMULACIÓN -->
  <arg name="paused"          default="false" />
  <arg name="use_sim_time"    default="true" />
  <arg name="gui"             default="false" />
  <arg name="headless"       default="false" />
  <arg name="debug"          default="false" />
  <arg name="verbose"        default="true" />
  <arg name="world"          default="$(find
p3at_simulation)/models/urdf/empty.world" />
  <arg name="urdf"           default="$(find
p3at_simulation)/models/urdf/my_p3at.urdf" />

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <!--<arg name="world" value="$(arg world)"/>-->
  </include>

  <!-- SETEAMOS EL PARÁMETRO ROBOT_DESCRIPTION -->
  <param name="robot_description" command="$(find xacro)/xacro
-i $(arg urdf)" />

  <!-- INICIAMOS NODOS CON ARGUMENTOS DESCRITOS ANTERIORMENTE-->
  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
args="-param robot_description -urdf -model Pioneer-3AT" >
    <remap from="/p3at/laser/scan" to="/scan"/>
  </node>
  <!--<node name="robot_state_publisher"
pkg="robot_state_publisher" type="robot_state_publisher" />
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
-->

  <!-- LANZAMOS RViz-->
  <!--<node name="rviz" pkg="rviz" type="rviz" args="-d
~/rviz/p3at.rviz"/>-->

</launch>

```

## Anexo XXI: *navigation\_p3at.launch*

```
<launch>

<!-- run MAP SERVER -->
<include file="$(find p3at_simulation)/launch/map_server.launch"/>

<!-- run AMCL -->
<include file="$(find p3at_simulation)/launch/amcl_p3at.launch"/>

<!-- run MOVE BASE -->
<include file="$(find p3at_simulation)/launch/move_base_p3at.launch"/>

</launch>
```

## Anexo XXII: *poses\_saver.cpp*

```
#include <ros/ros.h>
#include <jsoncpp/json/json.h>
#include <nav_msgs/Path.h>
#include <geometry_msgs/PoseStamped.h>
#include <math.h>
#include <iostream>
#include <fstream>

using namespace std;

//DECLARACIÓN DE VARIABLES
int tmp=0; //VARIABLE TEMPORAL

//VARIABLES JSON
ofstream json_file; //PERMITE DECLARAR ARCHIVO JSON
Json::StyledWriter styledWriter; //PERMITE ESCRIBIR EN FORMATO JSON
Json::Value posesArray(Json::arrayValue); //ARRAY DE POSES
geometry_msgs::PoseStamped hector_pose; //ALMACENAN POSES
TEMPORALMENTE

//GUARDA UNA POSICIÓN EN ARCHIVO CON FORMATO JSON
void saveJsonArray(geometry_msgs::PoseStamped h_pose);

//CALLBACK DE LA SUSCRIPCIÓN AL TÓPICO /slam_out_pose
void savepose(const geometry_msgs::PoseStamped::ConstPtr& pos){
    hector_pose= *pos; //GUARDA LA POSE EN UNA VARIABLE
    saveJsonArray(hector_pose); //LLAMA A LA FUNCIÓN PARA GUARDAR
    LA POSE COMO OBJETO JSON
}

//COMIENZA FUNCIÓN MAIN
int main(int argc, char** argv){

    ros::init(argc, argv, "poses_saver"); //NOMBRE DEL NODO
    ros::NodeHandle node;
    ros::Rate rate(10.0);

    cout << "Nodo listo para capturar poses." << endl;
```

```

while (node.ok()){
    ros::Subscriber sub=node.subscribe("/slam_out_pose", 1000,
    savepose);
    ros::spin();
    rate.sleep();
}

    json_file.open("full_poses.json");           //ABRE EL ARCHIVO
JSON
    json_file << styledWriter.write(posesArray); //ESCRIBE UN
ARRAY DE OBJETOS JSON
    json_file.close();                          //CIERRA EL
ARCHIVO JSON

    cout << endl <<"Numero de poses capturadas: " <<
posesArray.size() << endl;

    return 0;}
//DEFINICION DE FUNCIÓN PARA ESCRIBIR EN FORMATO JSON
void saveJsonArray(geometry_msgs::PoseStamped h_pose) {
//DAMOS FORMATO AL ARCHIVO JSON
    Json::Value translation(Json::objectValue);
    translation["x"]= h_pose.pose.position.x;
    translation["y"]= h_pose.pose.position.y;
    translation["z"]= h_pose.pose.position.z;

    Json::Value rotation(Json::objectValue);
    rotation["qx"]   = h_pose.pose.orientation.x;
    rotation["qy"]   = h_pose.pose.orientation.y;
    rotation["qz"]   = h_pose.pose.orientation.z;
    rotation["w"]    = h_pose.pose.orientation.w;

    Json::Value transform(Json::objectValue);
    transform["1 Translation"]= translation;
    transform["2 Rotation"]   = rotation;

    Json::Value header(Json::objectValue);
    header["seq"]             = h_pose.header.seq;
    header["stamp"]["sec"]    = h_pose.header.stamp.sec;
    header["stamp"]["nsec"]   = h_pose.header.stamp.nsec;
    header["frame_id"]        = h_pose.header.frame_id;

    Json::Value transforms(Json::objectValue);
    transforms["Header"]     = header;

    transforms["Pose"]       = transform;

//DA VALOR AL ARRAY DE OBJETOS JSON
    posesArray[tmp]=transforms;
//AUMENTA LA POSICIÓN DEL ARRAY
    tmp++;
}

```

### Anexo XXIII: *poses\_resampling.cpp*

```

#include <ros/ros.h>
#include <geometry_msgs/Twist.h>

```

```

#include <geometry_msgs/Pose.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <iostream>
#include <jsoncpp/json/json.h>
#include <fstream>
#include <math.h>

using namespace std;

// DECLARACION DE VARIABLES //

//ARCHIVO CON LOS DATOS
ofstream target_poses;           //ARCHIVO JSON DE SALIDA
ifstream ifs("full_poses.json"); //ARCHIVO JSON DE ENTRADA

//VARIABLES PARA TRABAJAR CON JSON FILE
Json::Reader      reader;           //PERMITE LEER DE UN
JSON
Json::StyledWriter styledWriter;    //PERMITE ESCRIBIR EN
UN JSON
Json::Value       in_poses(Json::arrayValue); //ARRAY DE OBJETOS
JSON DE ENTRADA
Json::Value       out_poses(Json::arrayValue); //ARRAY DE OBJETOS
JSON DE SALIDA

//VARIABLES PARA ALMACENAR DATOS
geometry_msgs::Pose pre_pose, next_pose;

//VARIABLES DE TRABAJO
float R_diff=0;
int   tmp = 0;
int   target_n=0;
const float sample_dist = 1; //DISTANCIA EN METROS

// DECLARACION DE FUNCIONES //

//FUNCION QUE OBTIENE EL SIGUIENTE GOAL
geometry_msgs::Pose get_next_goal(Json::Value poses_file,
geometry_msgs::Pose actual_p, int tmp_i, const float sampledist);

//TRANSFORMA LAS POSES DEL ARCHIVO JSON A FORMATO geometry_msgs::Pose
geometry_msgs::Pose get_data_from_file(Json::Value poses_file, int i);

//CALCULA LA DISTANCIA ENTRE DOS PUNTOS
float calc_R_diff(geometry_msgs::Pose actual, geometry_msgs::Pose
goal);

//GUARDA UNA POSE EN UN ARCHIVO JSON
void saveJsonArray(geometry_msgs::Pose data, int tmp_x);

// COMIENZO FUNCION MAIN //

int main(int argc, char** argv)
{
    ros::init(argc, argv, "poses_resampler");
    reader.parse(ifs, in_poses);
    cout << "ARCHIVO CON TODAS LAS POSES CARGADO" << endl;
    cout << "DISTANCIA DE MUESTREO: " << sample_dist << " metros" <<
endl;
}

```

```

    while(tmp<in_poses.size()){

        //LA PRIMERA VEZ ASIGNA A LA POSE ACTUAL LA PRIMERA POSE DEL
        ARCHIVO JSON
        if(target_n==0)
            pre_pose = get_data_from_file(in_poses, 0);

            next_pose=get_next_goal(in_poses, pre_pose, tmp,
sample_dist);
        cout << "ENCUENTRA EL GOAL N°: " << target_n << endl;
        saveJsonArray(next_pose, target_n);
        target_n++;
        pre_pose=next_pose;
    }

    cout << endl << "REMUESTREO COMPLETO" << endl;
    target_poses.open("resampled_poses.json");
    target_poses << styledWriter.write(out_poses);
    target_poses.close();

    return 0;
}

// FUNCIONES //

// ... FUNCION DE MUESTREO .....//

geometry_msgs::Pose get_next_goal(Json::Value poses_array,
geometry_msgs::Pose actual_p, int tmp_i, const float sampledist){
    geometry_msgs::Pose goal;
    float R_tmp=0;
    const int json_size = poses_array.size();

    //BUSCA UN PUNTO CUYA DISTANCIA AL PUNTO ACTUAL/ANTERIOR SEA MAYOR
    QUE LA DISTANCIA DE MUESTREO
    while(R_tmp <= sampledist){
        if(tmp_i < json_size){
            //POSITION
            goal.position.x=poses_array[tmp_i]["Pose"]["1
Translation"]["x"].asFloat();
            goal.position.y=poses_array[tmp_i]["Pose"]["1
Translation"]["y"].asFloat();
            goal.position.z=poses_array[tmp_i]["Pose"]["1
Translation"]["z"].asFloat();
            //ORIENTATION
            goal.orientation.x=poses_array[tmp_i]["Pose"]["2
Rotation"]["qx"].asFloat();
            goal.orientation.y=poses_array[tmp_i]["Pose"]["2
Rotation"]["qy"].asFloat();
            goal.orientation.z=poses_array[tmp_i]["Pose"]["2
Rotation"]["qz"].asFloat();
            goal.orientation.w=poses_array[tmp_i]["Pose"]["2
Rotation"]["w"].asFloat();
        }
        else{
            //POSITION
            goal.position.x=poses_array[(json_size-1)]["Pose"]["1
Translation"]["x"].asFloat();
            goal.position.y=poses_array[(json_size-1)]["Pose"]["1
Translation"]["y"].asFloat();

```

```

        goal.position.z=poses_array[(json_size-1)]["Pose"]["1
Translation"]["z"].asFloat();
        //ORIENTATION
        goal.orientation.x=poses_array[(json_size-
1)]["Pose"]["2 Rotation"]["qx"].asFloat();
        goal.orientation.y=poses_array[(json_size-
1)]["Pose"]["2 Rotation"]["qy"].asFloat();
        goal.orientation.z=poses_array[(json_size-
1)]["Pose"]["2 Rotation"]["qz"].asFloat();
        goal.orientation.w=poses_array[(json_size-
1)]["Pose"]["2 Rotation"]["w"].asFloat();

        R_tmp = calc_R_diff(actual_p, goal);
        cout << "EL ULTIMO GOAL NO CUMPLE LA DISTANCIA DE
MUESTREO" << endl;
        cout << "LA DISTANCIA ENTRE LOS DOS ULTIMOS PUNTOS DE
MUESTREO ES DE: " << R_tmp << endl;

        tmp = tmp_i;
        return goal;
    }

    //CALCULA LA DISTANCIA ENTRE EL PUNTO ACTUAL Y LA SIGUIENTE
POSE DEL ARCHIVO JSON
    R_tmp = calc_R_diff(actual_p, goal);
    tmp_i++;
}

tmp = tmp_i;
return goal;
}

// ... PASA DE FORMATO JSON A geometry_msgs::Pose
.....//

geometry_msgs::Pose get_data_from_file(Json::Value poses_file, int i){
    geometry_msgs::Pose data;
    //POSITION
    data.position.x=poses_file[i]["Pose"]["1
Translation"]["x"].asFloat();
    data.position.y=poses_file[i]["Pose"]["1
Translation"]["y"].asFloat();
    data.position.z=poses_file[i]["Pose"]["1
Translation"]["z"].asFloat();
    //ORIENTATION
    data.orientation.x=poses_file[i]["Pose"]["2
Rotation"]["qx"].asFloat();
    data.orientation.y=poses_file[i]["Pose"]["2
Rotation"]["qy"].asFloat();
    data.orientation.z=poses_file[i]["Pose"]["2
Rotation"]["qz"].asFloat();
    data.orientation.w=poses_file[i]["Pose"]["2
Rotation"]["w"].asFloat();

    return data;
}

// ... PASA DE geometry_msgs::Pose a Json File .....//

void saveJsonArray(geometry_msgs::Pose data, int tmp_x){

```

```
//DA FORMATO AL ARCHIVO JSON Y ASIGNA VALORES
  Json::Value translation(Json::objectValue);
  translation["x"]= data.position.x;
  translation["y"]= data.position.y;
  translation["z"]= data.position.z;

  Json::Value rotation(Json::objectValue);
  rotation["qx"]   = data.orientation.x;
  rotation["qy"]   = data.orientation.y;
  rotation["qz"]   = data.orientation.z;
  rotation["w"]    = data.orientation.w;

  Json::Value transform(Json::objectValue);
  transform["1 Translation"]= translation;
  transform["2 Rotation"]   = rotation;

  Json::Value transforms(Json::objectValue);

  transforms["Pose"]          = transform;

  out_poses[tmp_x]=transforms;
}

// ... CALCULA LA DISTANCIA ENTRE 2 POSES .....//

float calc_R_diff(geometry_msgs::Pose actual, geometry_msgs::Pose
goal) {

  float ax =actual.position.x;
  float ay=actual.position.y;
  float gx=goal.position.x;
  float gy=goal.position.y;
  float R = sqrtf(powf((gx-ax),2)+powf((gy-ay),2));
  return R;
}

```

## Anexo XXIV: *path\_publisher.cpp*

```
#include <ros/ros.h>
#include <jsoncpp/json/json.h>
#include <nav_msgs/Path.h>
#include <geometry_msgs/Pose.h>
#include <iostream>
#include <fstream>

using namespace std;
ifstream json_file("full_poses.json");
Json::Reader reader;
Json::Value obj(Json::arrayValue);
nav_msgs::Path globalpath;

geometry_msgs::PoseStamped tmp_pose;

void getGlobalPathFromJson(Json::Value p);

int main(int argc, char** argv) {

```

```

ros::init(argc, argv, "path_publisher");
ros::NodeHandle node;
ros::Rate rate(10);
ros::Publisher pub=node.advertise<nav_msgs::Path>("globalpath", 1);

globalpath.header.frame_id="map";
reader.parse(json_file, obj);
cout << "CARGANDO EL PATH" << endl;

    for(int tmp=0; tmp<obj.size(); tmp++){

        tmp_pose.header.seq=obj[tmp]["Header"]["seq"].asUInt();

        tmp_pose.header.stamp.sec=obj[tmp]["Header"]["stamp"]["sec"].asU
Int();

        tmp_pose.header.stamp.nsec=obj[tmp]["Header"]["stamp"]["nsec"].a
sFloat();

        tmp_pose.header.frame_id=obj[tmp]["Header"]["frame_id"].asString
();

            tmp_pose.pose.position.x=obj[tmp]["Pose"]["1
Translation"]["x"].asFloat();
            tmp_pose.pose.position.y=obj[tmp]["Pose"]["1
Translation"]["y"].asFloat();
            tmp_pose.pose.position.z=obj[tmp]["Pose"]["1
Translation"]["z"].asFloat();
            tmp_pose.pose.orientation.x=obj[tmp]["Pose"]["2
Rotation"]["qx"].asFloat();
            tmp_pose.pose.orientation.y=obj[tmp]["Pose"]["2
Rotation"]["qy"].asFloat();
            tmp_pose.pose.orientation.z=obj[tmp]["Pose"]["2
Rotation"]["qz"].asFloat();
            tmp_pose.pose.orientation.w=obj[tmp]["Pose"]["2
Rotation"]["w"].asFloat();
            globalpath.poses.push_back(tmp_pose);
            cout << (tmp*100)/(obj.size()) << "%" << endl;
        }
        cout << "PATH CARGADO" << endl;

        while(node.ok()){
            pub.publish(globalpath);
            ros::spinOnce();
            rate.sleep();
        }
    return 0;
}

```

## Anexo XXV: *point\_follower.cpp*

```

#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/Pose.h>
#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <iostream>
#include <jsoncpp/json/json.h>
#include <fstream>

```



```

#include <math.h>

using namespace std;

// ... DECLARACION DE VARIABLES
.....//

//ARCHIVO CON LOS DATOS
ifstream targets("resampled_poses.json"); //ARCHIVO DE DONDE SE
OBTIENEN LOS GOALS

//VARIABLES PARA TRABAJAR CON JSON FILE
Json::Reader      reader;
Json::Value       target_array(Json::arrayValue);

//VARIABLES PARA ALMACENAR DATOS
geometry_msgs::PoseWithCovarianceStamped amcl_pose;
geometry_msgs::Pose actual_pose, goal_pose;
geometry_msgs::Twist cmd_vel;

//VARIABLES DE TRABAJO
float R_diff, angle_diff;
int tmp = 0;
int target_n=0;
bool goal_reached = false;
bool get_new_goal = true;

// ... DECLARACION DE FUNCIONES
.....//

//FUNCIONES INTERNAS
geometry_msgs::Pose get_data_from_file(Json::Value poses_file, int i);
geometry_msgs::Pose
get_actual_pose(geometry_msgs::PoseWithCovarianceStamped posecova);
geometry_msgs::PoseStamped goal_to_rviz(geometry_msgs::Pose goal);

float calc_R_diff(geometry_msgs::Pose actual, geometry_msgs::Pose
goal);
float calc_angle_diff(geometry_msgs::Pose actual, geometry_msgs::Pose
goal);
float compute_w(float angle_diff);
float compute_v(float pose_diff);

//CALLBACK DE LA SUSCRIPCION
void saveAmclPose(const
geometry_msgs::PoseWithCovarianceStamped::ConstPtr& amclpose){
    actual_pose=get_actual_pose(*amclpose);
}

// ... COMIENZO FUNCION MAIN
.....//
.//

int main(int argc, char** argv)
{
    ros::init(argc, argv, "point_follower");
    ros::NodeHandle node;

```

```

    ros::Rate rate(10.0);
    ros::Publisher
cmd_vel_pub=node.advertise<geometry_msgs::Twist>("/sim/cmd_vel", 100);
    ros::Publisher
goal_pub=node.advertise<geometry_msgs::PoseStamped>("/next_goal", 1);
    ros::Subscriber amcl_sub =node.subscribe("/amcl_pose", 1,
saveAmclPose);
    reader.parse(targets, target_array);

    while(node.ok()){

        R_diff = calc_R_diff(actual_pose, goal_pose);
//      cout << "R_diff: " << R_diff << endl;
        angle_diff = calc_angle_diff(actual_pose, goal_pose);

        if(get_new_goal==true){
            goal_pose = get_data_from_file(target_array,
target_n);
            cout << "GOAL_POSE N°: " << target_n+1 << endl;
            target_n++;
            get_new_goal = false;
        }

        if(R_diff <= 0.5)
            get_new_goal = true;

        cmd_vel.linear.x = 0.05; //VELOCIDAD
CONSTANTE O "compute_v(R_diff)"
        cmd_vel.angular.z = compute_w(angle_diff);
        goal_pub.publish(goal_to_rviz(goal_pose)); //PUBLICA EL
GOAL PARA VISUALIZARLO EN RVIZ
        cmd_vel_pub.publish(cmd_vel); //PUBLICA LOS
COMANDOS DE VELOCIDAD

        ros::spinOnce();
        rate.sleep();
    }
    return 0;
}

// ... PASA DE FORMATO JSON A geometry_msgs::Pose
.....//

geometry_msgs::Pose get_data_from_file(Json::Value poses_file, int i){
    geometry_msgs::Pose data;
    //POSITION
    data.position.x=poses_file[i]["Pose"]["1
Translation"]["x"].asFloat();
    data.position.y=poses_file[i]["Pose"]["1
Translation"]["y"].asFloat();
    data.position.z=poses_file[i]["Pose"]["1
Translation"]["z"].asFloat();
    //ORIENTATION
    data.orientation.x=poses_file[i]["Pose"]["2
Rotation"]["qx"].asFloat();
    data.orientation.y=poses_file[i]["Pose"]["2
Rotation"]["qy"].asFloat();

```

```

    data.orientation.z=poses_file[i]["Pose"]["2
Rotation"]["qz"].asFloat();
    data.orientation.w=poses_file[i]["Pose"]["2
Rotation"]["w"].asFloat();

    return data;
}

// ... CONVIERTE GOAL_POSE PARA MOSTRARLA EN RVIZ
.....//

geometry_msgs::PoseStamped goal_to_rviz(geometry_msgs::Pose goal){
    geometry_msgs::PoseStamped rviz_goal;

    rviz_goal.header.frame_id="map";
    rviz_goal.pose=goal;
    return rviz_goal;
}

// ... CONVIERTE AMCL_POSE A geometry_msgs::Pose
.....//

geometry_msgs::Pose
get_actual_pose(geometry_msgs::PoseWithCovarianceStamped posecova){
    geometry_msgs::Pose pose;

    //POSITION
    pose.position.x=posecova.pose.pose.position.x;
    pose.position.y=posecova.pose.pose.position.y;
    pose.position.z=posecova.pose.pose.position.z;
    //ORIENTATION
    pose.orientation.x=posecova.pose.pose.orientation.x;
    pose.orientation.y=posecova.pose.pose.orientation.y;
    pose.orientation.z=posecova.pose.pose.orientation.z;
    pose.orientation.w=posecova.pose.pose.orientation.w;

    return pose;
}

// ... CALCULA LA DISTANCIA ENTRE 2 POSES
.....//

float calc_R_diff(geometry_msgs::Pose actual, geometry_msgs::Pose
goal){
    float ax =actual.position.x;
    float ay=actual.position.y;
    float gx=goal.position.x;
    float gy=goal.position.y;
    float R = sqrtf(powf((gx-ax),2)+powf((gy-ay),2));

    return R;
}

```

```
// ... CALCULA EL ÁNGULO ENTRE 2 POSES
.....//

float calc_angle_diff(geometry_msgs::Pose actual, geometry_msgs::Pose
goal){
    double actual_angle, goal_angle, a_diff;
    //QUATERNION TO YAW
    double siny_actual = 2 * (actual.orientation.w *
actual.orientation.z);
    double cosy_actual = 1 - 2 * (actual.orientation.z *
actual.orientation.z);
    actual_angle = std::atan2(siny_actual, cosy_actual);

    //QUATERNION TO YAW
    double siny_goal = 2 * (goal.orientation.w *
goal.orientation.z);
    double cosy_goal = 1 - 2 * (goal.orientation.z *
goal.orientation.z);
    goal_angle = std::atan2(siny_goal, cosy_goal);

    a_diff= goal_angle - actual_angle;

    return a_diff;
}

// ... CONTROL_P DE VELOCIDAD ANGULAR
.....//

float compute_w(float angle_diff){
    float control_w=0;
    float K=0.5;

    control_w=K * angle_diff;

    if(control_w > 0.2)
        control_w=0.2;
    else if(control_w < -0.2)
        control_w=-0.2;

    return control_w;
}

// ... CONTROL_P DE VELOCIDAD LINEAL
.....//

float compute_v(float pose_diff){
    float control_v=0;
    float K=0.1;

    control_v= K * pose_diff;

    if(control_v > 0.1)
        control_v=0.1;
    // else if(control_v < -0.1)
    //     control_v=-0.1;

    return control_v;
}

```

## Anexo XXVI: *hector\_params.yaml*

```

base_frame: base_link
map_frame: map
odom_frame: odom
map_resolution: 0.05
map_size: 1024
map_start_x: 0.5
map_start_y: 0.5
map_update_distance_thresh: 0.4
map_update_angle_thresh: 0.9
map_pub_period: 2.0
map_multi_res_levels: 3
update_factor_free: 0.4
update_factor_occupied: 0.9
laser_min_dist: 0.4
laser_max_dist: 30.0
laser_z_min_value: -1.0
laser_z_max_value: 1.0
pub_map_odom_transform: true
output_timing: false
scan_subscriber_queue_size: 50
pub_map_scanmatch_transform: true
tf_map_scanmatch_transform_frame_name: scanmatcher_frame

```

## Anexo XXVII: *hector\_mapping.launch*

```

<launch>

<!--INITIALIZE hector_mapping-->
  <node name="hector_mapping" pkg="hector_mapping"
    type="hector_mapping">

    <remap from="/scan" to="/sim/scan"/>
    <rosparam file="$(find
p3at_simulation)/params/hector_params.yaml" command="load" />

  </node>

</launch>

```

## Anexo XXVIII: *test\_point\_follower.launch*

```

<launch>

  <!-- run GAZEBO SIMULATION -->
  <include file="$(find
p3at_simulation)/launch/robot_p3at_point_follower_world_simulate
d.launch"/>

```

```

<!-- run MAP SERVER -->
<node name="map_server" pkg="map_server" type="map_server"
args="$(find
p3at_simulation)/models/maps/point_follower_map.yaml">
</node>

<!-- run AMCL -->
<include file="$(find
p3at_simulation)/launch/amcl_p3at_SIM.launch"/>

<!-- run RViz -->
<node name="rviz" pkg="rviz" type="rviz" args="-d
/home/fran/.rviz/hector_gmapping.rviz"/>

</launch>

```

### Anexo XXIX: *robot\_p3at\_point\_follower\_world\_simulated.launch*

```

<launch>

  <!-- INDICAMOS DONDE ESTA LA CARPETA CON NUESTROS MODELOS -->
  <env name="GAZEBO_MODEL_PATH"
value="$GAZEBO_MODEL_PATH:$(find p3at_simulation)/models/urdf"
/>
  <env name="GAZEBO_RESOURCE_PATH"
value="$GAZEBO_RESOURCE_PATH:$(find
p3at_simulation)/models/urdf" />

  <!-- CREAMOS ARGUMENTOS PARA CONFIGURAR LA SIMULACIÓN -->
  <arg name="paused"          default="false" />
  <arg name="use_sim_time"    default="true" />
  <arg name="extra_gzbo_args" default="" />
  <arg name="gui"             default="true" />
  <arg name="headless"        default="false" />
  <arg name="debug"           default="false" />
  <arg name="verbose"         default="true" />
  <arg name="output"          default="screen" />
  <arg name="physics"         default="ode" />
  <arg name="respawn_gazebo"  default="false" />
  <arg name="server_required" default="false" />
  <arg name="gui_required"    default="false" />
  <arg name="world"           default="$(find
p3at_simulation)/models/worlds/point_follower.world" />

  <!-- INDICAMOS MODELO DEL ROBOT -->
  <arg name="urdf"             default="$(find
p3at_simulation)/models/urdf/my_p3at.urdf" />

  <!-- SETEAMOS EL PARÁMETRO ROBOT_DESCRIPTION -->
  <param name="robot_description" command="$(find xacro)/xacro -i
$(arg urdf)"/>

  <!-- POSICION INICIAL DEL ROBOT EN ENTORNO SIMULACION -->
  <arg name="x_off" value="5" /> <!--Desplazamiento en x-->
  <arg name="y_off" value="0" /> <!--Desplazamiento en y-->
  <arg name="Y_off" value="3.1416"/> <!--Rotación respecto z-->

```

```

<!-- SETEAMOS EL PARAMETRO /use_sim_time A TRUE -->
<param name="/use_sim_time" value="$(arg use_sim_time)" />

<!-- INSERTAMOS NUESTRO ROBOT EN GAZEBO -->
<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
args="-param robot_description -urdf -x $(arg x_off) -y $(arg
y_off) -Y $(arg Y_off) -model Pioneer-3AT" />

<!-- INICIAMOS NODO EL CUAL PUBLICA NUESTRAS TRANSFORMADAS -->
<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />

->
<!-- INICIAMOS NODO PUBLICA EL ESTADO DE LAS UNIONES DEL ROBOT -
->
<node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />

<!-- SET COMMAND ARGUMENTS -->
<arg unless="$(arg paused)" name="command_arg1" value="" />
<arg if="$(arg paused)" name="command_arg1" value="-u" />
<arg unless="$(arg headless)" name="command_arg2" value="" />
<arg if="$(arg headless)" name="command_arg2" value="-r" />
<arg unless="$(arg verbose)" name="command_arg3" value="" />
<arg if="$(arg verbose)" name="command_arg3" value="--verbose" />
<arg unless="$(arg debug)" name="script_type" value="gzserver" />
<arg if="$(arg debug)" name="script_type" value="debug" />

<!-- INICIAMOS GAZEBO SERVER -->
<node name="gazebo"
  pkg="gazebo_ros"
  type="$(arg script_type)"
  respawn="$(arg respawn_gazebo)"
  output="$(arg output)"
  args="$(arg command_arg1) $(arg command_arg2) $(arg
command_arg3) -e $(arg physics) $(arg extra_gzbo_args)
$(arg world)"
  required="$(arg server_required)" />

<!-- INICIAMOS GAZEBO CLIENT -->
<group if="$(arg gui)">
  <node name="gazebo_gui"
    pkg="gazebo_ros"
    type="gzclient"
    respawn="false"
    output="$(arg output)"
    args="$(arg command_arg3)"
    required="$(arg gui_required)"/>
</group>

</launch>

```

## 9. BIBLIOGRAFÍA

- [1] hiosur.com, «Robot Móvil,» [En línea]. Disponible en:  
<https://www.hiosur.com/es/mobile-robot-42899/>.
- [2] Adept MobileRobots, «Pioneer 3-AT Datasheet,» [En línea]. Disponible en: <https://www.generationrobots.com/media/Pioneer3AT-P3AT-RevA-datasheet.pdf>.
- [3] SICK, «Hoja de datos en línea,» [En línea]. Disponible en:  
[https://cdn.sick.com/media/pdf/3/43/843/dataSheet\\_LMS200-30106\\_1015850\\_es.pdf](https://cdn.sick.com/media/pdf/3/43/843/dataSheet_LMS200-30106_1015850_es.pdf).
- [4] SICK, «Laser Measurement Systems,» [En línea]. Disponible en:  
<http://sicktoolbox.sourceforge.net/docs/sick-lms-technical-description.pdf>.
- [5] Ubuntu, «Robotics,» [En línea]. Disponible en:  
<https://ubuntu.com/robotics#:~:text=Tech%20visionaries%20choose%20Ubuntu%20to,vision%20of%20a%20robotic%20future.>
- [6] Ubuntu, «Ubuntu 16.04.7 LTS,» [En línea]. Disponible en:  
<https://releases.ubuntu.com/16.04/>.
- [7] Ubuntu, «The Linux command line for beginners,» [En línea]. Disponible en: <https://ubuntu.com/tutorials/command-line-for-beginners#1-overview>.
- [8] Guia Ubuntu, «Comandos - Guía Ubuntu,» [En línea]. Disponible en:  
<https://www.guia-ubuntu.com/index.php/Comandos>.
- [9] ROS Wiki, «Documentation,» [En línea]. Disponible en:  
<http://wiki.ros.org/>.
- [10] ROS Wiki, «ROS/Concepts,» [En línea]. Disponible en:  
<http://wiki.ros.org/ROS/Concepts>.
- [11] ROS Wiki, «Instalation/Ubuntu,» [En línea]. Disponible en:  
<http://wiki.ros.org/Installation/Ubuntu>.



- [12] ROS Wiki, «catkin/workspaces,» [En línea]. Disponible en:  
[http://wiki.ros.org/catkin/workspaces#Catkin\\_Workspaces](http://wiki.ros.org/catkin/workspaces#Catkin_Workspaces).
- [13] ROS Wiki, «Creating a ROS Package,» [En línea]. Disponible en:  
<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>.
- [14] ROS Wiki, «ROS Tutorials,» [En línea]. Disponible en:  
<http://wiki.ros.org/ROS/Tutorials>.
- [15] ROS Wiki, «roscatkin/rosrun,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin#rosrun>.
- [16] ROS Wiki, «roscatkin,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin>.
- [17] ROS Wiki, «roscatkin\_graph,» [En línea]. Disponible en:  
[http://wiki.ros.org/roscatkin\\_graph](http://wiki.ros.org/roscatkin_graph).
- [18] ROS Wiki, «Creating a ROS msg and srv,» [En línea]. Disponible en:  
<http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv>.
- [19] ROS Wiki, «roscatkin,» [En línea]. Disponible en:  
<http://wiki.ros.org/action/show/roscatkin?action=show&redirect=rossrv>.
- [20] ROS Wiki, «roscatkin,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin>.
- [21] ROS Wiki, «Writing a Simple Service and Client (C++),» [En línea].  
Disponible en:  
<http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29>.
- [22] ROS Wiki, «rossrv,» [En línea]. Disponible en:  
<http://wiki.ros.org/rossrv>.
- [23] ROS Wiki, «roscatkin,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin>.
- [24] ROS Wiki, «roscatkin/XML,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin/XML>.
- [25] ROS Wiki, «roscatkin/Commandline Tools,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin/Commandline%20Tools>.
- [26] ROS Wiki, «roscatkin/Commandline,» [En línea]. Disponible en:  
<http://wiki.ros.org/roscatkin/Commandline>.

- [27] ROS Wiki, «geometry\_msgs/Twist,» [En línea]. Disponible en: [http://docs.ros.org/en/api/geometry\\_msgs/html/msg/Twist.html](http://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html).
- [28] ROS Wiki, «nav\_msgs/Odometry,» [En línea]. Disponible en: [http://docs.ros.org/en/api/nav\\_msgs/html/msg/Odometry.html](http://docs.ros.org/en/api/nav_msgs/html/msg/Odometry.html).
- [29] Wikipedia, «Odometría,» [En línea]. Disponible en: <https://es.wikipedia.org/wiki/Odometr%C3%ADa>.
- [30] ROS Wiki, «RosAria/Tutorials,» [En línea]. Disponible en: <http://wiki.ros.org/ROSARIA/Tutorials/How%20to%20use%20ROSARIA>.
- [31] Gazebo, «Which combination of ROS/Gazebo versions to use,» [En línea]. Disponible en: [http://gazebosim.org/tutorials/?tut=ros\\_wrapper\\_versions](http://gazebosim.org/tutorials/?tut=ros_wrapper_versions).
- [32] Gazebo, «URDF in Gazebo,» [En línea]. Disponible en: [http://gazebosim.org/tutorials?tut=ros\\_urdf&cat=connect\\_ros](http://gazebosim.org/tutorials?tut=ros_urdf&cat=connect_ros).
- [33] Gazebo, «SDF format Specification,» [En línea]. Disponible en: <http://sdformat.org/spec>.
- [34] ROS Wiki, «urdf,» [En línea]. Disponible en: <http://wiki.ros.org/urdf>.
- [35] ROS Wiki, «urdf/XML,» [En línea]. Disponible en: <http://wiki.ros.org/urdf/XML>.
- [36] ROS Wiki, «rviz/UserGuide,» [En línea]. Disponible en: <http://wiki.ros.org/rviz/UserGuide>.
- [37] ROS Wiki, «Setting up your robot using tf,» [En línea]. Disponible en: <http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF>.
- [38] ROS Wiki, «tf,» [En línea]. Disponible en: <http://wiki.ros.org/tf>.
- [39] V. F. M. Martínez, «Tesis Doctoral. - Universidad de Málaga,» 1995. [En línea]. Disponible en: <http://webpersonal.uma.es/~VFMM/tesis.html#CONTRIBUCIONES>. [Último acceso: 2021].
- [40] ROS Wiki, «Navigation,» [En línea]. Disponible en: <http://wiki.ros.org/navigation>.
- [41] ROS Wiki, «REP 105 - Coordinate Frames for Mobile Platforms,» [En línea]. Disponible en: <https://www.ros.org/reps/rep-0105.html>.

- [42] ROS Wiki, «gmapping,» [En línea]. Disponible en:  
<http://wiki.ros.org/gmapping>.
- [43] S.-A. G. Pujals, «key\_teleop,» [En línea]. Disponible en:  
[http://wiki.ros.org/key\\_teleop](http://wiki.ros.org/key_teleop).
- [44] S. THRUN, W. BURGARD y D. FOX, Probabilistic Robotics, 1999.
- [45] ROS Wiki, «amcl,» [En línea]. Disponible en: <http://wiki.ros.org/amcl>.
- [46] ROS Wiki, «constmap\_2d,» [En línea]. Disponible en:  
[http://wiki.ros.org/costmap\\_2d](http://wiki.ros.org/costmap_2d).
- [47] ROS Wiki, «clear\_costmap\_recovery,» [En línea]. Disponible en:  
[http://wiki.ros.org/clear\\_costmap\\_recovery](http://wiki.ros.org/clear_costmap_recovery).
- [48] ROS Wiki, «rotate\_recovery,» [En línea]. Disponible en:  
[http://wiki.ros.org/rotate\\_recovery](http://wiki.ros.org/rotate_recovery).
- [49] MobileRobots, «MobileRobots/amr-ros-config,» [En línea]. Disponible en:  
<https://github.com/MobileRobots/amr-ros-config>.
- [50] Gazebo, «Gazebo plugins in ROS,» [En línea]. Disponible en:  
[http://gazebo.org/tutorials?category=ros\\_gzplugins](http://gazebo.org/tutorials?category=ros_gzplugins).
- [51] K. Zheng, «ROS Navigation Tuning Guide,» [En línea]. Disponible en:  
<https://kaiyuzheng.me/documents/navguide.pdf>.
- [52] WikiBooks, «JsonCpp,» [En línea]. Disponible en:  
<https://en.wikibooks.org/wiki/JsonCpp>.
- [53] G. Shuang, «Skid Steering in 4-Wheel-Drive Electric Vehicle,» 2007. [En línea]. Disponible en: <https://www.semanticscholar.org/paper/Skid-Steering-in-4-Wheel-Drive-Electric-Vehicle-Shuang-Cheung/5ca8d2b5b49532ad385d95d961c5e863f1d666b8#paper-header>.
- [54] Wikipedia, «Ubuntu,» [En línea]. Disponible en:  
<https://es.wikipedia.org/wiki/Ubuntu>.
- [55] GitHub, «GitHub,» [En línea]. Disponible en: <https://github.com/>.
- [56] ROS Wiki, «Setup and Configuration of the Navigation Stack on a Robot,» [En línea]. Disponible en:  
<http://wiki.ros.org/navigation/Tutorials/RobotSetup>.

- [57] ROS Wiki, «nav\_msgs/Odometry,» [En línea]. Disponible en:  
[http://docs.ros.org/en/noetic/api/nav\\_msgs/html/msg/Odometry.html](http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html).
- [58] ROS Wiki, «global\_planner,» [En línea]. Disponible en:  
[http://wiki.ros.org/global\\_planner](http://wiki.ros.org/global_planner).
- [59] ROS Wiki, «base\_local\_planner,» [En línea]. Disponible en:  
[http://wiki.ros.org/base\\_local\\_planner](http://wiki.ros.org/base_local_planner).
- [60] ROS Wiki, «move\_base,» [En línea]. Disponible en:  
[http://wiki.ros.org/move\\_base](http://wiki.ros.org/move_base).

