

Guia para replicar mi progreso en Webots

| | |
|--|----------|
| Requisitos previos | 2 |
| Pasos iniciales | 2 |
| Procedimiento para probar los robots. | 3 |
| Componentes de la simulación. | 5 |
| Guía para migrar tu robot a Webots | 8 |



Requisitos previos

-Instalar ros2_humble: Recomendando hacerlo por debs usando este tutorial <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html>

-Instalar Webots 2023b: Recomendando descargar el deb e instalarlo como un programa más. <https://github.com/cyberbotics/webots/releases>

-Instalar Webots_ros2 desde el paquete oficial de lanzamiento con este comando:

```
sudo apt-get install ros-humble-webots-ros2
```

Pasos iniciales

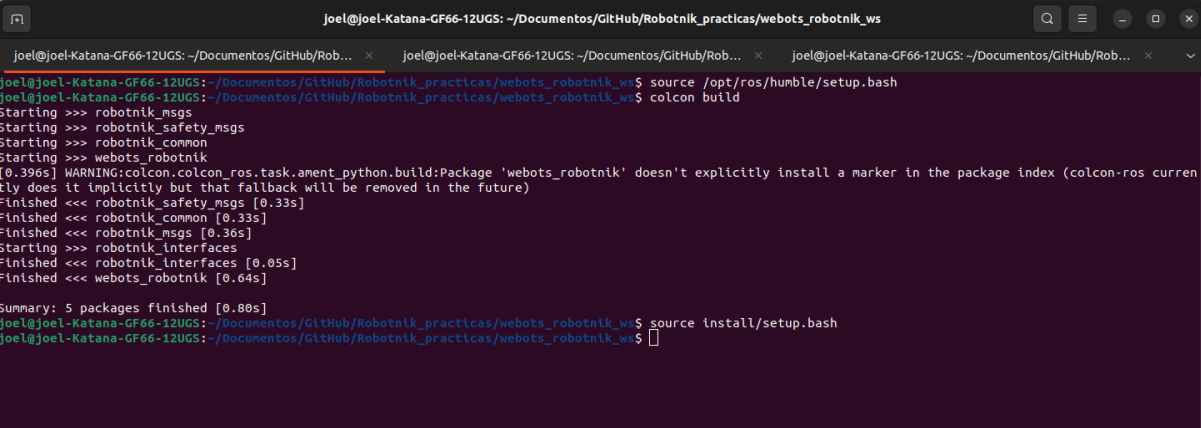
-Clonar mi repositorio de “Robotnik_prácticas” de internet. Este es el enlace: https://github.com/JoelRamosBeltran/Robotnik_practicas

-Abrir una terminal en Robotnik_ws

-Realizar un “`source /opt/ros/humble/setup.bash`”

-Compilar con “`colcon build`”

-Realizar un “`source install/local_setup.bash`”



```
Joel@Joel-Katana-GF66-12UGS: ~/Documentos/GitHub/Robotnik_practicas/webots_robotnik_ws
Joel@Joel-Katana-GF66-12UGS: ~/Documentos/GitHub/Robotnik_practicas/webots_robotnik_ws$ source /opt/ros/humble/setup.bash
Joel@Joel-Katana-GF66-12UGS: ~/Documentos/GitHub/Robotnik_practicas/webots_robotnik_ws$ colcon build
Starting >>> robotnik_msgs
Starting >>> robotnik_safety_msgs
Starting >>> robotnik_common
Starting >>> webots_robotnik
[0.396s] WARNING:colcon.colcon_ros.task.ament_python.build:Package 'webots_robotnik' doesn't explicitly install a marker in the package index (colcon-ros currently does it implicitly but that fallback will be removed in the future)
Finished <<< robotnik_safety_msgs [0.33s]
Finished <<< robotnik_common [0.33s]
Finished <<< robotnik_msgs [0.36s]
Starting >>> robotnik_interfaces
Finished <<< robotnik_interfaces [0.05s]
Finished <<< webots_robotnik [0.64s]

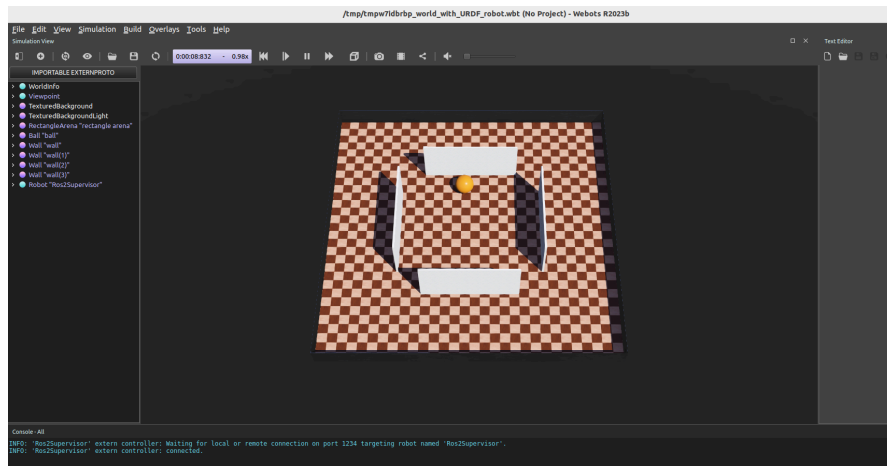
Summary: 5 packages finished [0.80s]
Joel@Joel-Katana-GF66-12UGS: ~/Documentos/GitHub/Robotnik_practicas/webots_robotnik_ws$ source install/setup.bash
Joel@Joel-Katana-GF66-12UGS: ~/Documentos/GitHub/Robotnik_practicas/webots_robotnik_ws$
```

Procedimiento para probar los robots.

-Recomiendo abrir 4 terminales (todas ellas en el directorio Robotnik_ws), y que en las cuatro se hagan los dos sources (los de los pasos iniciales)

-En la primera terminal ejecutaremos el mundo

```
ros2 launch webots_robotnik world_launch.py
```

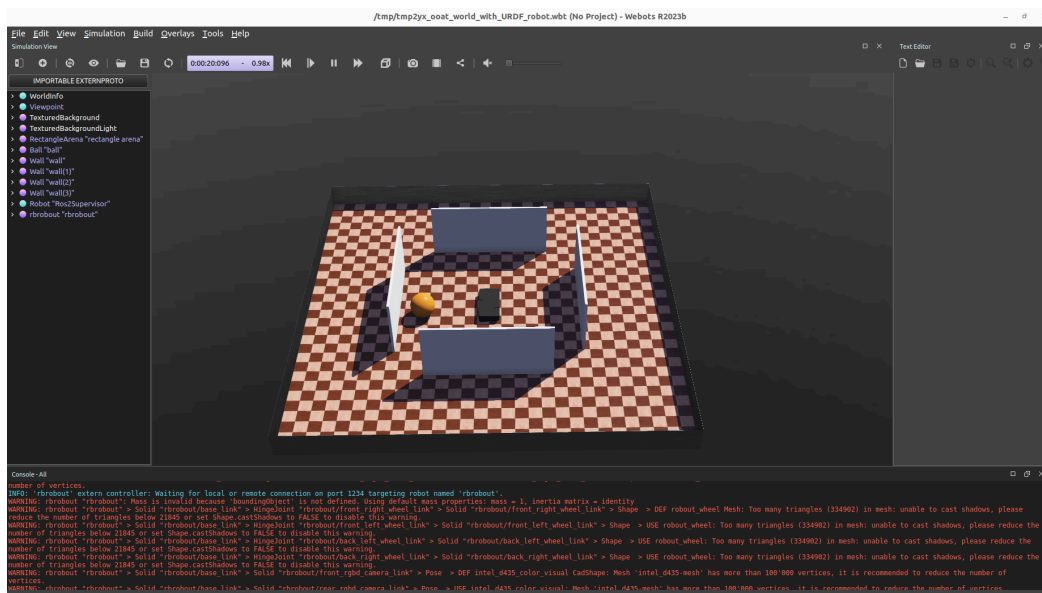


-En la segunda terminal ejecutaremos el spawn del robot

```
ros2 launch webots_robotnik robot_launch_alfa.py robot:=rbrobout namespace:=rbrobout  
x:=0 y:=0 z:=0
```

0

```
ros2 launch webots_robotnik robot_launch_alfa.py robot:=rbwatcher namespace:=rbwatcher  
x:=2 y:=2 z:=0
```



-En la tercera terminal ejecutaremos el cmd_vel

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -r  
/cmd_vel:=rbrout2/diffdrive_controller/cmd_vel_unstamped
```

-Y en la cuarta la utilizaremos de manera auxiliar, como por ejemplo:

1. Revisar los topics: `ros2 topic list`
2. Abrir el rviz2 para revisar el topic de las cámaras y sensores.
3. Spawnear un segundo robot:

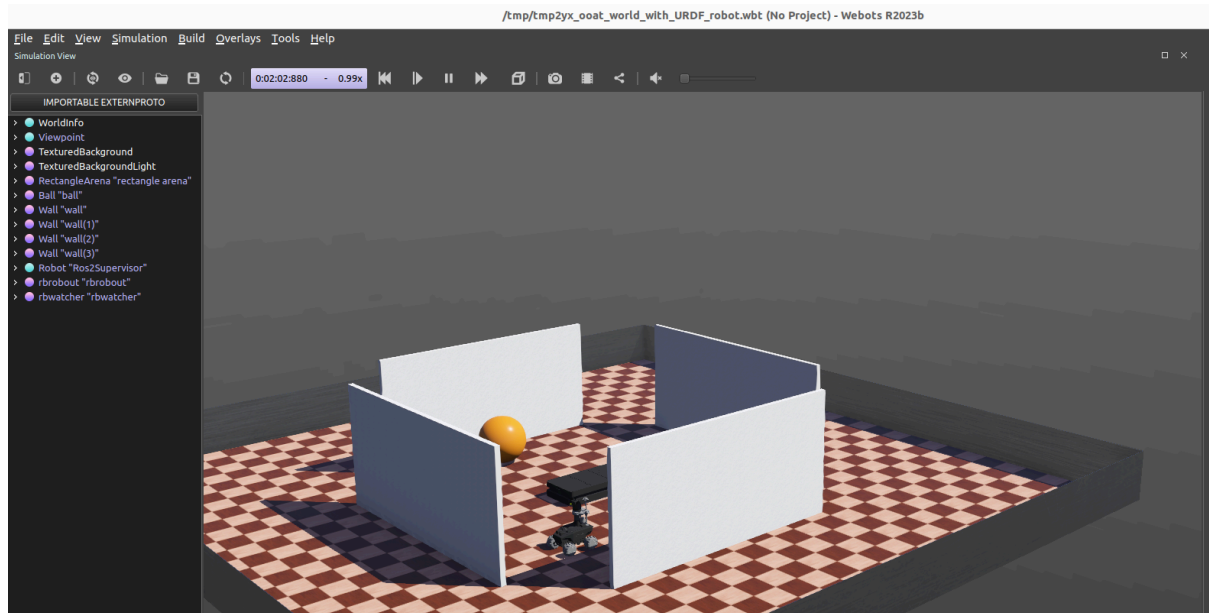
```
ros2 launch webots_robotnik robot_launch_alfa.py robot:=modelo_robot  
namespace:=rbwatcher x:=2 y:=2 z:=0
```

(Es probable que en los dos PROTOs y en el archivo de mundo haya rutas a modelos dae y stl que no funcionen correctamente, habría que arreglarlas si dan error).

Después de esto, deberías poder desplazar correctamente el robot por el escenario, recibiendo los datos de sus sensores en el rviz2.

El comando entero de spawneo es este:

```
ros2 launch webots_robotnik robot_launch_alfa.py robot:=modelo_robot  
namespace:=nombre_unico_del_robot x:=0 y:=0 z:=0
```



Componentes de la simulación.

Antes de nada, voy a explicar un poco como funciona Webots. El simulador Webots se conforma de un archivo wbt, el cual es el mundo, y varios archivos PROTO.

Los archivos PROTO son archivos que siguen el mismo formato que el archivo del mundo, y son los encargados de definir los diferentes objetos de la simulación. Estos pueden ser importados en el mundo, incluso con la interfaz de la simulación, y entre estos, también se incluyen los PROTOs de los robots. Se podría decir que son como módulos importables en el mundo.

Voy a dividir los archivos de la simulación en 3 tipos:

- Launchers
- Elementos de Webots
- Elementos auxiliares de ROS

Launchers

En el repositorio podemos encontrar varios launchers, los cuales son producto de las diferentes pruebas que hice para poder lanzar todos los robots de una manera multirobot y usando controladores correctamente. Beta, avanzado y alfa.

- **Beta:** Este es el primer launcher que creé, y es bastante simple.
- **Avanzado:** Este fue el intento principal durante un largo tiempo, hasta que me topé con problemas usando el Robotnik Controller, culpa de la diferencia entre effort y torque.
- **Alfa:** Este es el actual, el cual funciona con los dos tipos de robots, permite multirobot y usa un controlador diff_drive incluido en Webots.

El launcher alfa es el que funciona correctamente, y es del que voy a hablar a partir de ahora. Este launcher, como todos los launchers de ROS, se compone de 2 partes, la declaración de argumentos y la declaración de acciones que se lanzan.

Los argumentos son los mismos que los incluidos en el simulador de Gazebo, incluyendo el nombre del robot, su namespace y su posición en el mundo.

En cuanto a las acciones, aquí es donde está lo importante del launcher, voy a describir una a una:

- **robot_state_publisher:** Este lanza el nodo encargado de publicar la estructura del robot. Por sí solo no publica nada, pero nos será útil más adelante. Un detalle es el namespace, el cual se usará el mismo para todos los nodos.
- **spawn_robot_service_call:** Este es uno de los más importantes. Se encarga de llamar al servicio de ros2_supervisor que se encarga de spawnear en el mundo un PROTO (en este caso el robot) importado con anterioridad en el archivo del mundo.

A este se le pasa como argumento todos los parámetros que tiene este PROTO con el fin de que sus tfs sean únicas y diferentes a otros robots, además, se le pasa también la posición en el mundo en la que va a aparecer y su nombre único (namespace).

- **footprint_publisher:** Este se encarga de crear el tf footprint
- **worldtf_publisher:** Este se encarga de crear el tf world, al cual se le aplica de transformada relativa con el odom la posición de spawn del robot, con el fin de que World esté en el 0,0,0 del mapa.
- **joint_state_broadcaster:** Activa el nodo encargado de spawnear el controlador "joint_state_broadcaster" el cual publica el estado de las joints móviles del robot.
- **diffdrive_controller_spawner:** Activa el nodo encargado de spawnear el controlador del robot, es decir, lo que hace que sus ruedas se muevan.
- **robotnik_controller:** Este nodo no está siendo activado, pero aún así, su función sería la de activar el robotnik_controller, pero con un problema con la interfaz effort no he logrado hacerlo funcionar correctamente.
- **rbrout_driver:** Este es el encargado de generar, vincular y hacer funcionar todos los controladores de un determinado robot.

Automáticamente busca al robot por su nombre único, y le aplica el controlador declarado en el URDF "controller_avanzado" cuyos parámetros se encuentran en un yml.

Además, activando "set_robot_state_publisher", el robot_state se publica de forma automática para el determinado robot.

El otro launcher necesario para la ejecución de la simulación, y de hecho, el que es necesario lanzar primero, es el encargado de spawnear el mundo, es decir "world_launch". Este simplemente está formado por una sola acción:

- **WebotsLauncher:** Lanza el mundo de Webots indicado. Hay que usar el parámetro del ros2_supervisor para que funcione todo lo referente a ROS.

Elementos de Webots

Mundo: El archivo wbt contiene el mundo de Webots que estoy utilizando para testear los robots. En la parte superior del archivo podemos ver la importación de los PROTOs de los dos robots que van a poder ser spawnados durante la simulación.

PROTOS: Como he dicho arriba, los PROTOs son archivos que contienen elementos de Webots, en este caso, tenemos un PROTO para cada robot, cuya estructura es similar. Lo más destacable son los fields, los cuales son parámetros que personalizan los PROTO, estos pueden ser introducidos al declarar la instancia del PROTO como si fueran argumentos.

Elementos auxiliares de ROS

En la carpeta resource podemos encontrar los archivos que se utilizan para hacer funcionar los controladores de los robots. Los archivos se dividen en dos, el URDF del controlador, y sus parámetros, para cada uno de los modelos de robot.

-URDF: Este es un archivo que incluye los plugins de Webots referentes a los controladores, además, también se declara el hardware de las articulaciones y sus interfaces.

-params.yml: Este archivo incluye los parámetros para todos los controladores que necesita el robot, en este caso, el Joint State Broadcaster y el Differential Drive Controller.

Guía para migrar tu robot a Webots

Voy a separar esta guía en dos partes: De XACRO/URDF a PROTO y ROS2.

De XACRO/URDF a PROTO

Vamos a partir del punto en el que tienes un URDF o XACRO con tu robot. Este es el caso de los robots incluidos en Robot_description.

Primero de todo, revisa el Xacro y mira que todos los argumentos tengan un valor por defecto, y que tal valor te lleve a tener el robot de la forma en la que lo quieres en Webots. Es decir, que pongas como default los argumentos que sueles usar para spawnear el robot. Esto es importante ya que la función tomará los defaults.

Ahora abre una terminal en el directorio principal del repositorio en el que tengas robot_description y robotnik_sensors (en mi caso Robotnik_ws), y luego de hacer "source install/setup.bash" (si ya has compilado), muévete a la carpeta en la que tienes el xacro principal (está en el directorio robots dentro del paquete Robot_description).

Estando en el directorio, realizas este comando:

```
ros2 run webots_ros2_importer xacro2proto --input=rbwatcher.urdf.xacro --output=rbwatcher.proto
```

Sustituyendo rbwatcher por el robot en cuestión. Esto creará automáticamente el PROTO del robot en cuestión. El prompt debería ser algo como esto:

```
joel@joel-Katana-GF66-12UGS:~/Documentos/GitHub/Robotnik_practicas/Robotnik_ws$ ros2 run webots_ros2_importer xacro2proto --input=rbvogul.urdf.xacro --output=rbvogul.proto
Root link: robot_base_footprint
Unsupported "RGB_INT8" image format, using "R8G8B8A8" instead.
Unsupported "L_INT8" image format, using "R8G8B8A8" instead.
Unsupported "L_INT8" image format, using "R8G8B8A8" instead.
Unsupported "RGB_INT8" image format, using "R8G8B8A8" instead.
Unsupported "L_INT8" image format, using "R8G8B8A8" instead.
Unsupported "L_INT8" image format, using "R8G8B8A8" instead.
There are 41 links, 40 joints and 11 sensors
```

Y si todo ha salido bien, tendrás todo el modelo del robot en PROTO.

Aunque hay algunas cosas que es necesario cambiar para conseguir que funcione correctamente todo, porque o si no, es muy probable que el robot aparezca completamente desmontado en la simulación.

1. Primero de todo, siempre que sea posible, cambia los .dae por .stl. Esto garantizará que las piezas del robot aparezcan en la orientación deseada.
2. Segundo, hay que modificar algunos fields (los argumentos de la parte superior) con el fin de garantizar que el robot se conecte a un controlador externo, y por tanto, funcione su manejo.


```

8 PROTO rbrobout [
9   field SFVec3f    translation 0 0 -0.49
10  field SFRotation  rotation    0 0 1 0
11  field SFString    name        "rbrobout" # Is `Robot.name`.
12  field SFString    controller  "<extern>" # Is `Robot.controller`.
13  field MFString    controllerArgs [] # Is `Robot.controllerArgs`.
14  field SFString    customData  "" # Is `Robot.customData`.
15  field SFBool      supervisor  FALSE # Is `Robot.supervisor`.
16  field SFBool      synchronization TRUE # Is `Robot.synchronization`.
17  field SFBool      selfCollision FALSE # Is `Robot.selfCollision`.
18  field MFNode      toolSlot    [] # Extend the robot with new nodes at the end of the arm.
19 ]

```

3. En el caso en el que se quisiera realizar un proceso multirobot, veo necesario añadir nuevos fields referentes a los nombres de los solids y sensors. Esto permitirá añadir prefijos a los tfs de los robots y así evitar que se solapen sus datos con los de otro. Recomiendo revisar mi PROTO del RbRobout y del RbWatcher con el fin de llevar a cabo tal proceso.

Con esto tu PROTO debería estar acabado. En este punto, si tu PROTO está en la carpeta de PROTOs, podrás añadir el robot fácilmente a la simulación desde la interfaz gráfica, aunque este dará error de conexión al controlador. También puede que haya errores referentes al modelo sobrecargado del robot, pero puedes ignorarlos.

Y ahora, verás que el robot se queda estático volando en el lugar de spawn. No te asustes, esto es a causa de que no le hemos añadido la física principal. Se arregla fácilmente poniendo `physics Physics {}`

```

Robot {
  translation IS translation
  rotation IS rotation
  controller IS controller
  physics Physics {
  }
  children [
    Shape {
      appearance PBRAppearance {
        baseColor 0.5 0.5 0.5
      }
    }
  ]
}

```

Con esto, el robot ya caerá contra el suelo.

Es probable que, más adelante, tengas que editar el robot para recolocar las cámaras y otros sensores, pero por ahora el PROTO está terminado.

Para finalizar esta parte, edita el archivo del mundo añadiendo la dirección a tu nuevo robot en la parte superior. Esto servirá más adelante para que podamos spawnear varias instancias de este en nuestro mundo.

```

IMPORTABLE EXTERNPROTO "../protos/rbrobout.proto"
IMPORTABLE EXTERNPROTO "../protos/rbwatcher.proto"

```

ROS2 y controladores

Bien, tenemos el modelo del robot, pero hay que hacer que funcione, porque un robot que no se mueve no es un robot.

Primero vamos al launcher que, aunque no tengas que cambiar nada de él por ahora, nos servirá para guiarnos con los archivos que necesitamos crear. El launcher ya lo he descrito arriba, así que aquí me meteré solo en los archivos que tienes que crear.

1. **URDF de control.** Este URDF, con nombre “robot_controller_avanzado.urdf” es un URDF ubicado en resources, y en él se encuentra todo lo referente al controlador y el hardware del robot.

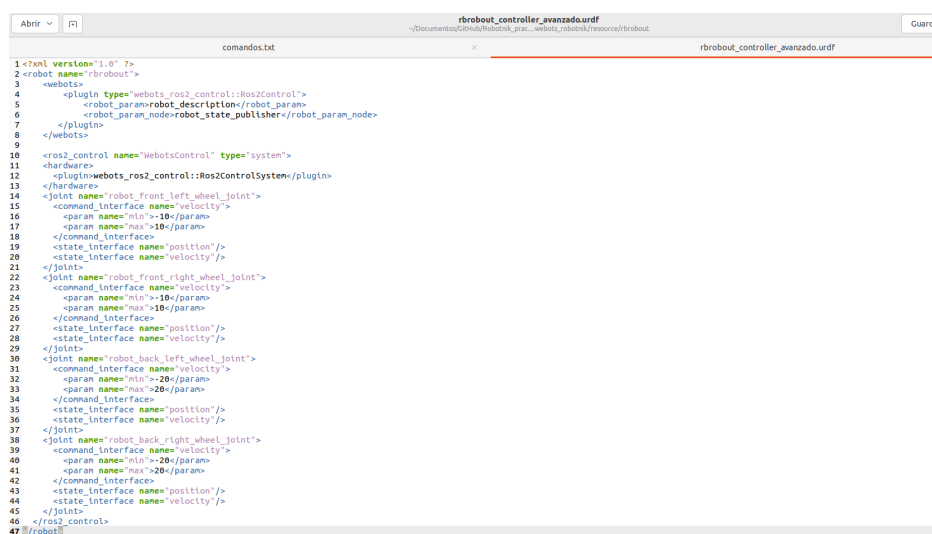
Para editar este archivo, recomiendo usar como base el del RbRobout y el XACRO del controlador del robot elegido, que se encuentra en robot_description, dentro de la carpeta simulation.

2. **Parámetros de control.** Estos parámetros son exactamente iguales que los que se usan en Ignition, aunque el robotnik_controller, por el problema del effort, no me iba, así que usé el diff_drive que hay de base en el simulador.

Para editarlo recomiendo lo mismo que en el anterior, tomar los parámetros en Webots del RbRobout y los parámetros del robot elegido en robot_description.

Y con esto sería todo. Si has puesto las cosas en las carpetas que toca, con los nombres que toca, debería funcionar. El robot debería aparecer con controlador en la simulación usando los dos launchers de los que hablé anteriormente.

Una vez el robot aparezca y se mueva, revisa los topics. Si todo ha ido bien, deberían estar publicándose los topics de todos los sensores. También comprueba las cámaras, ya que sus frames puede que se hayan girado, y será necesario modificar el PROTO para recolocarlas al sitio.



```
1 <?xml version="1.0" ?>
2 <robot name="rrobout">
3   <webots>
4     <plugin type="webots_ros2_control::Ros2Control">
5       <robot_param robot_description="/robot_param">
6         <robot_param_node robot_state_publisher="/robot_param_node">
7       </robot_param_node>
8     </plugin>
9   </webots>
10   <ros2_control name="WebotsControl" type="system">
11     <hardware>
12       <plugin>webots_ros2_control::Ros2ControlSystem</plugin>
13     </hardware>
14     <joint name="robot_front_left_wheel_joint">
15       <command_interface name="velocity">
16         <param name="min">-10</param>
17         <param name="max">10</param>
18       </command_interface>
19       <state_interface name="position"/>
20       <state_interface name="velocity"/>
21     </joint>
22     <joint name="robot_front_right_wheel_joint">
23       <command_interface name="velocity">
24         <param name="min">-10</param>
25         <param name="max">10</param>
26       </command_interface>
27       <state_interface name="position"/>
28       <state_interface name="velocity"/>
29     </joint>
30     <joint name="robot_back_left_wheel_joint">
31       <command_interface name="velocity">
32         <param name="min">-20</param>
33         <param name="max">20</param>
34       </command_interface>
35       <state_interface name="position"/>
36       <state_interface name="velocity"/>
37     </joint>
38     <joint name="robot_back_right_wheel_joint">
39       <command_interface name="velocity">
40         <param name="min">-20</param>
41         <param name="max">20</param>
42       </command_interface>
43       <state_interface name="position"/>
44       <state_interface name="velocity"/>
45     </joint>
46   </ros2_control>
47 </robot>
```

Posibles errores

Estos son posibles errores que pueden haberte ocurrido al migrar un nuevo robot.

1. **El robot spawnea con falta de modelos o directamente invisible.** Como he dicho arriba, esto ocurre por no haberse compilado `robot_description` y `robotnik_sensors`. Por eso recomiendo clonar todo mi repositorio de prácticas, para así evitar que se te pierda nada. Aún así, el error debería marcarlo en la terminal inferior de la interfaz de Webots.
2. **Al spawnear varios robots se implosionan.** Esto es debido a que alguna de las mallas de colisiones es incorrecta. Revisa el mundo entero, ya que, por ejemplo, en mi caso un elemento del `rbwatcher` tenía una malla más grande que el escenario, y al spawnear dos de estos, colisionaban entre sí. Si la pieza no es muy relevante, puedes quitar su colisión eliminando su `BoundingBox` en el PROTO
3. **Error con el controlador.** Hay varios motivos por los que el controlador no podría funcionar:
 - Al crear los fields para los Tfs únicos en el PROTO, hiciste también para el motor de las ruedas, lo que causó que el nombre de este motor no coincide con el del controlador.
 - Hay algún error con el URDF o yaml del controlador. Este error debería verse en la terminal.
 - Se te pasó modificar el field "controller" en el PROTO del robot a "<extern>"
4. **Error al importar el PROTO desde XACRO/URDF.** Este error me ocurrió al intentar pasar el `rbvogui` al `rbrobout`. Parece ser que el importador no toma en cuenta los links sin inercia, y en la body del robot, el `base_link` estaba declarado sin inercia. Esto causó que todo lo que era hijo de `base_link` se obviara, y solo tomara una de las cámaras.

Otro problema que puede ser causado es que las piezas te salgan desmontadas en el Webots, esto se arregla cambiando los `.dae` por los `.stl`

Para evitar otro tipo de problemas, procura hacer `source install/setup.bash` al paquete de `robot_description` y `sensors` para que el comando encuentre todos los modelos y xacros.

Eso sería todo. Buena suerte.