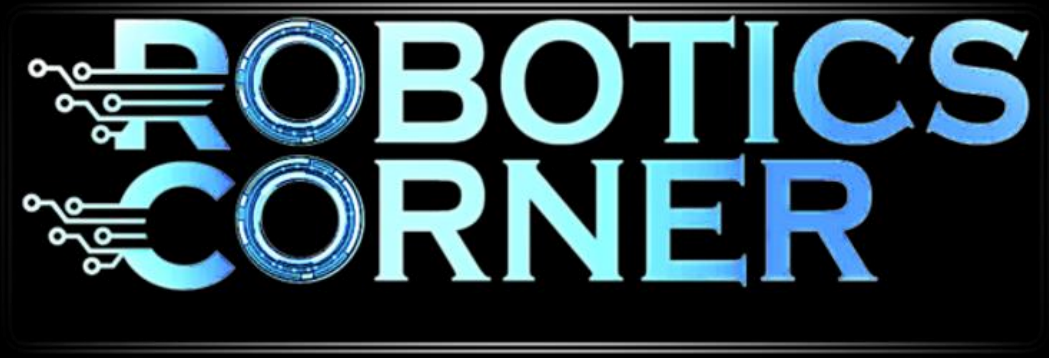


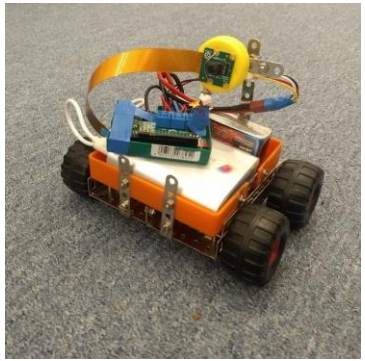
Gazebo simulation for Robocar

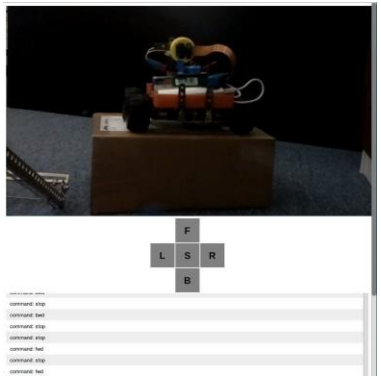
Plan

- Motivation
- Intro to ROS
- Building a simulation for roborace in Gazebo

Motivation

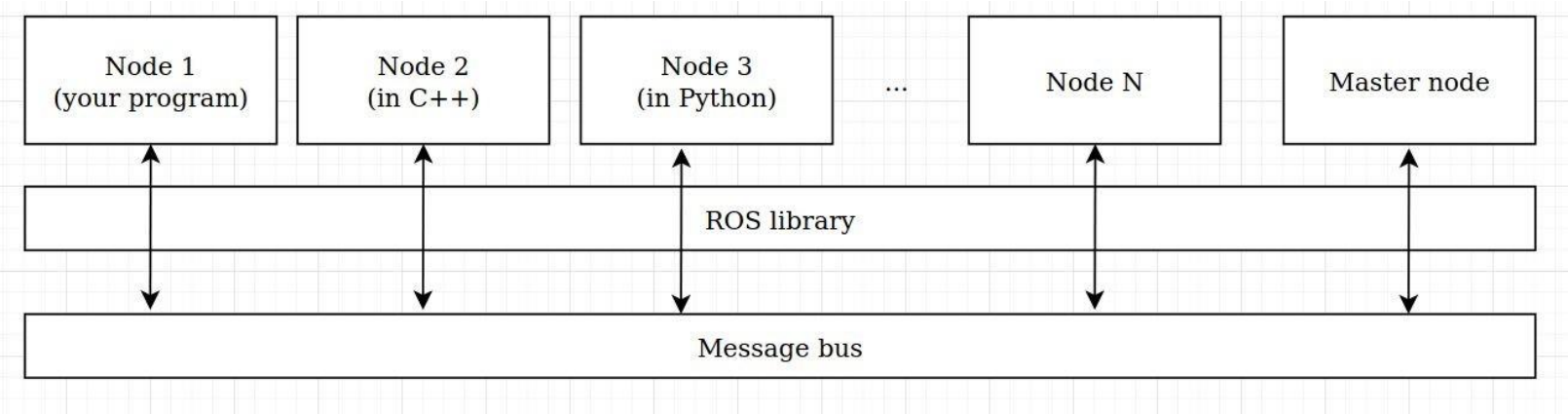
- Preparing for robocar competition 2019
  - Meetup
  - First steps with robotics
  - Low-cost self driving car
- Donkeycar software
  - https://www.donkeycar.com/
  - Easy to start with
  - Active community
  - Specialized software stack
- Robot Operating System (ROS)
  - flexible framework for writing robot software.
  - It is a collection of tools, libraries, and conventions
  - https://www.ros.org/about-ros/

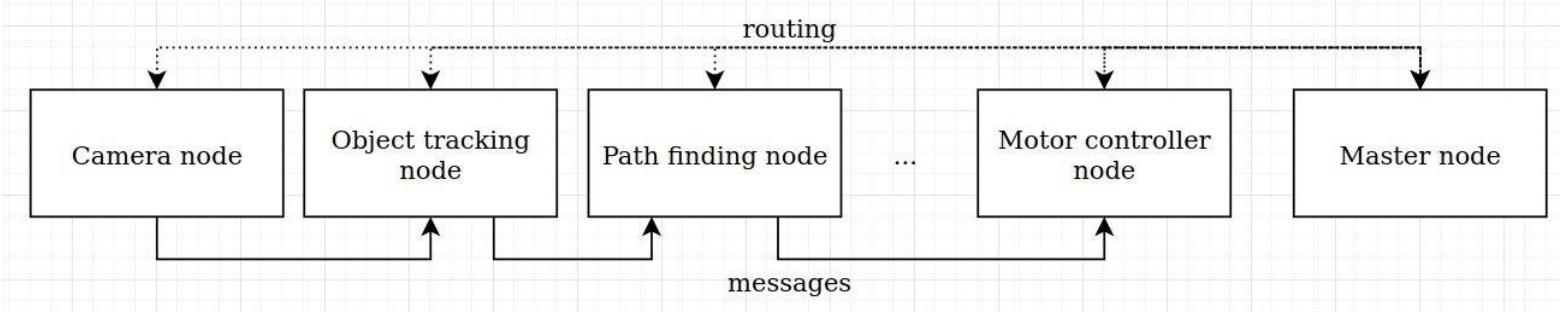




https://www.donkeycar.com/

ROS: Overview





How does ROS look like?

- Set up environment
- Write code with ROS libraries in C++ or python
    - Use built-in modules as building blocks
    - Define message types
    - Write configs
- Launch
- Use visualization and debug tools

```
import rospy
# imports
        ...

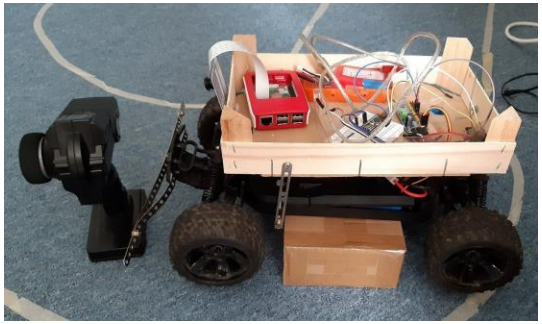
class WheelController(object):
   def __init__(self):
        rospy.init_node("wheel_controller")
        list_ctrlrs = rospy.ServiceProxy(
               "controller_manager/list_controllers",
               ListControllers
        )
        list_ctrlrs.wait_for_service()
        self._sleep_timer = rospy.Rate(3)
        self.axle_pub = rospy.Publisher(
               "axle_ctrlr/command", Float64, queue_size=1
        )

   #...
   def spin(self):
        while not rospy.is_shutdown():
               val = generate_message()
               self.axle_pub.publish(val)
               self._sleep_timer.sleep()

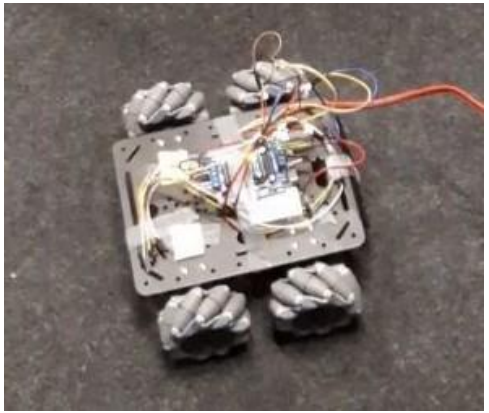
if __name__ == "__main__":
        ctrlr = WheelController()
        ctrlr.spin()
```

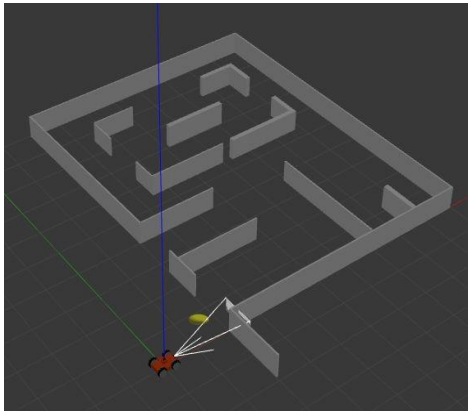
Projects with ROS

- Robocar
  - github V1 - race 2019, wiki
  - github V2 (ongoing)



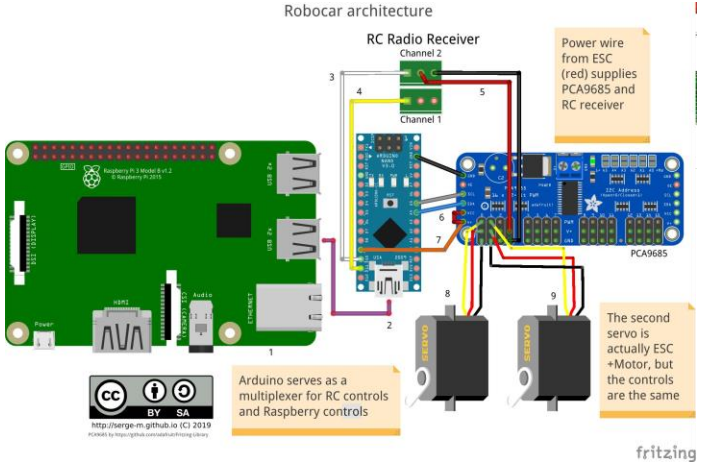
- Project Omicron
  - target: https://www.robocup.org/
  - blog https://project-omicron.github.io/
  - code https://github.com/project-omicron/





Porting Donkeycar to ROS

- Inspiration
  - Donkeycar
  - Project Omicron
  - ….
- Hardware was customized a bit
- ROS for Raspberry Pi
  - Installing ROS Kinetic on the Raspberry Pi
  - ROS melodic on Raspberry Pi 3 B+
- Robocar on github



Why simulation

- I had two projects ongoing, when Corona happened
- How to develop if you don’t have hardware at hand?
- Donkeycar simulator
  - https://github.com/tawnkramer/sdsandbox
  - Learning to Drive Smoothly in Minutes
- Gazebo simulator

What is Gazebo

- Gazebo is an open-source 3D robotics simulator.
  - Physics engine
  - 3D rendering
  - Sensor simulation
  - Integration with ROS
- Let’s launch some simulation (video)
  - $ sudo apt-get install ros-<distro>-husky-simulator
  - $ source /opt/ros/melodic/setup.bash
  - $ export HUSKY_GAZEBO_DESCRIPTION=$(rospack find husky_gazebo)/urdf/description.gazebo.xacro
  - $ roslaunch husky_gazebo husky_playpen.launch kinect_enabled:=true
  - In another terminal to check that the sensors work

    # activate ROS

    $ rqt_image_view
  - In another terminal to control

    # activate ROS

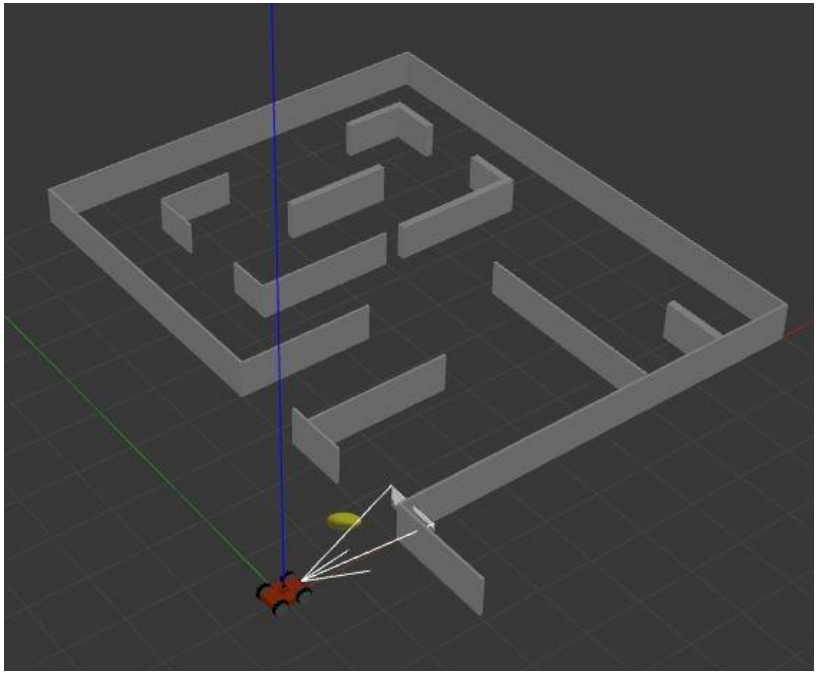
    rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Another example

- Project Omicron https://github.com/project-omicron/gazebo_simulation
  - Check out the README, install the dependencies.
  - ```
    export GAZEBO_MODEL_PATH=/home/s/catkin_ws/src/gazebo_simulation/models/:$GAZEBO_MODEL_PATH
    mkdir -p ~/catkin_ws/src && cd ~/catkin_ws/src
    git clone https://github.com/project-omicron/gazebo_simulation.git
    git clone https://github.com/SyrianSpock/realsense_gazebo_plugin.git
    cd ../
    catkin_make
    source devel/setup.bash
    roslaunch gazebo_simulation world_of_labyrinth.launch
    ```
  - Controlling

    cd ~/catkin_ws/ && source devel/setup.bash
    rosrun teleop_twist_keyboard teleop_twist_keyboard.py
    ```

Another example



Resources for learning

- Tutorials
  - http://wiki.ros.org/urdf_tutorial
  - http://gazebosim.org/tutorials
- Learning by copying
  - turtlebot3 https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/
  - husky http://wiki.ros.org/husky_gazebo/Tutorials/Simulating%20Husky
  - husky + slam https://github.com/incebellipipo/carto_demo
- More advanced examples
  - https://github.com/AutoRally/autorally
  - https://mushr.io/tutorials/intro-to-ros/

Simulation for robocar race

```
# preparation

# We can either create a new workspace or use an existing one. i'll start with my repo.
mkdir ~/demo
cd ~/demo
git clone https://github.com/serge-m/robocar_v2.git --branch ackermann-demo-start
source /opt/ros/melodic/setup.bash
cd robocar_v2/catkin_ws
rosdep install --from-paths src --ignore-src --rosdistro=melodic -y

# remove unnecessary packages (raspi cam, others) if needed, then make
catkin_make
```

Simulation for robocar race

```
cd catkin_ws/src
catkin_create_pkg robocar_simulation std_msgs rospy roscpp ackermann_msgs

tree ./robocar_simulation/
./robocar_simulation/
├── CMakeLists.txt
├── include
│   └── robocar_simulation
├── package.xml
└── src

catkin_create_pkg robocar_ackermann_description std_msgs rospy
catkin_create_pkg robocar_world std_msgs rospy


cd ~/demo/robocar_v2/catkin_ws/
catkin_make
source devel/setup.bash
```

changing package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>robocar_simulation</name>
  <version>0.0.1</version>
  <description>The robocar_simulation package</description>
  <maintainer email="serge-m@users.noreply.github.com">SergeM</maintainer>
  <license>MIT</license>
  <url type="website">https://github.com/serge-m/robocar_v2/</url>


  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>husky_gazebo</exec_depend>

  <export>
  </export>
</package>
```

launch file

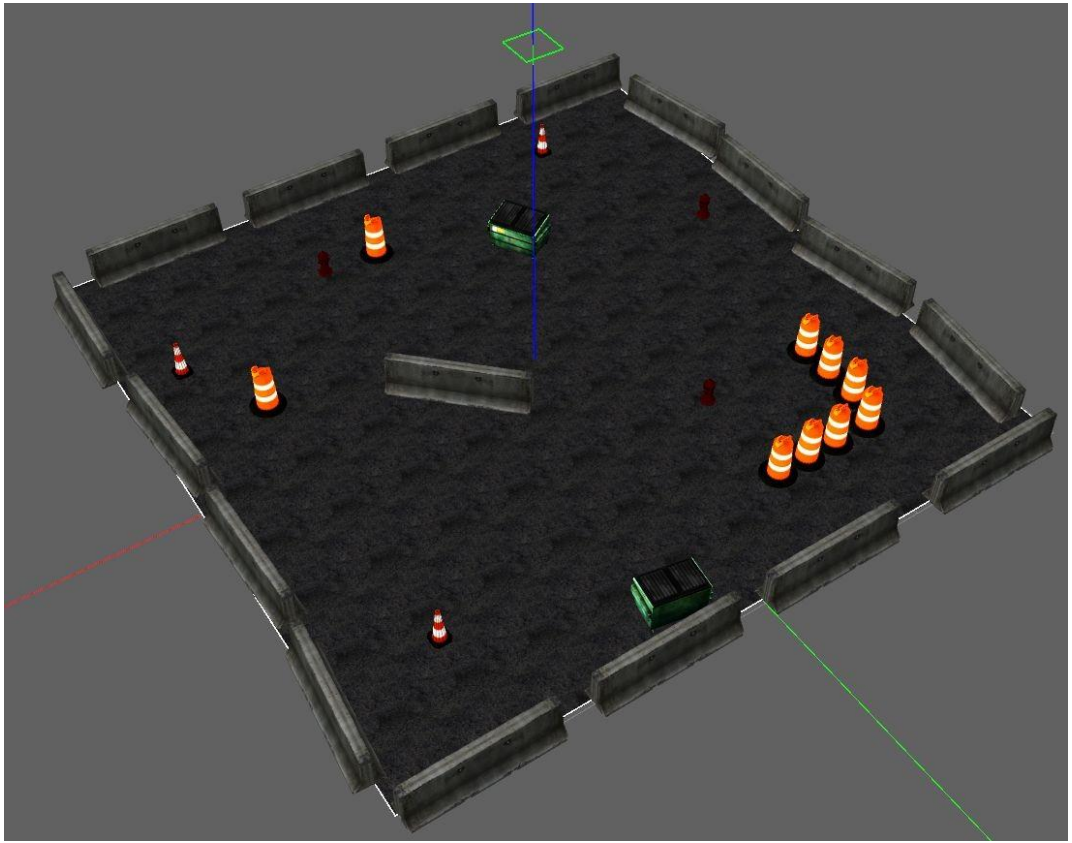
```
<?xml version="1.0"?>
<launch>
  <!-- environment vars. We may need it later -->
  <env name="GAZEBO_MODEL_PATH" value="$(find robocar_ackermann_description)"/>
  <env name="GAZEBO_RESOURCE_PATH" value="$(find robocar_ackermann_description)"/>

  <!-- arguments -->
  <arg name="world_name" default="$(find husky_gazebo)/worlds/clearpath_playpen.world"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- including another standard launch file from gazebo_ros file -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
      <arg name="world_name" value="$(arg world_name)"/>
      <arg name="debug" value="$(arg debug)" />
      <arg name="gui" value="$(arg gui)" />
      <arg name="paused" value="$(arg paused)"/>
      <arg name="use_sim_time" value="$(arg use_sim_time)"/>
      <arg name="headless" value="$(arg headless)"/>
  </include>
</launch>
```

launch file - results

```
roslaunch robocar_simulation robocar_sim.launch
```



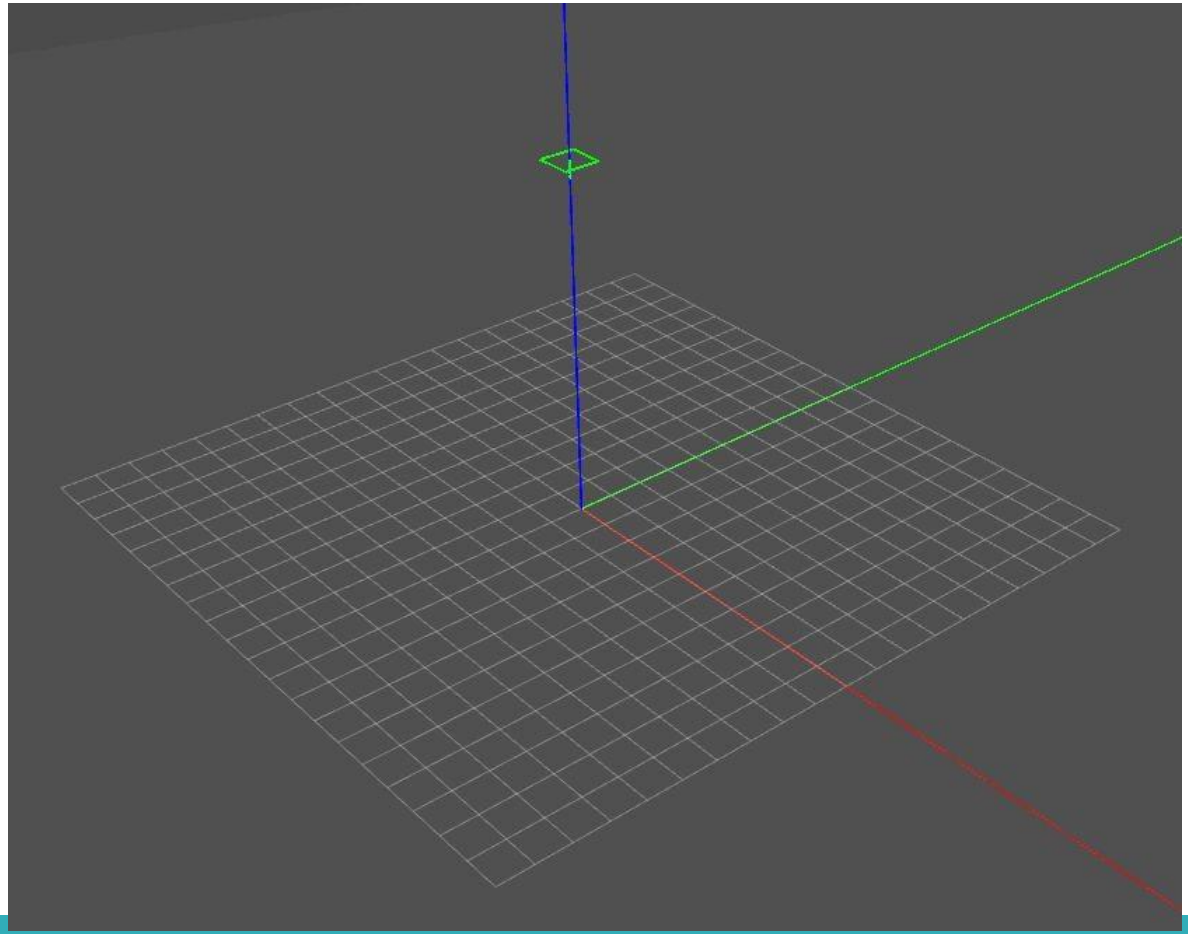
Defining the (empty) world

```
# SDF: Simulation Description Format

<sdf version="1.4">
  <world name="default">
      <scene>
      <ambient>0.6 0.6 0.6 1</ambient>
      <background>0.7 0.7 0.7 1</background>
      <shadows>true</shadows>
      </scene>
      <!-- global light -->
      <include>
      <uri>model://sun</uri>
      </include>
      <!-- ground plane -->
      <include>
      <uri>model://ground_plane</uri>
      </include>
  </world>
</sdf>


# Running:
roslaunch robocar_simulation robocar_sim.launch \
    world_name:=$(rospack find robocar_world)/world/empty.world
```

Defining the (empty) world

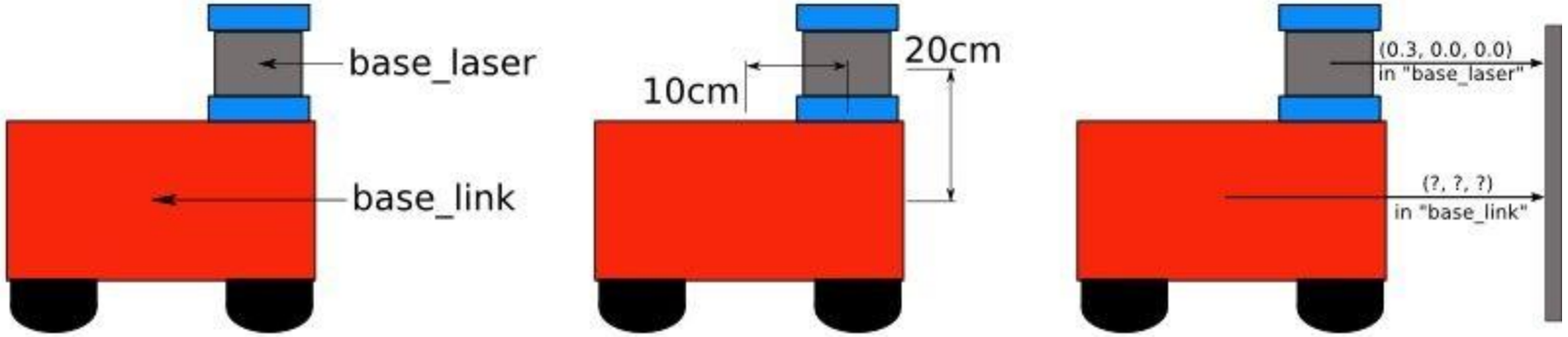


Defining a robot

To define a robot we have to describe all the parts (aka links) and connections between them (aka joints).

Ros and Gazebo take care of

- geometrical transformations (tf module),
- physics simulation and rendering.



http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

Defining a robot - URDF/XACRO (not working)

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="zaza">
  <xacro:property name='width' value='2.2' />
  <xacro:property name='length' value='2.4' />
  <xacro:property name='height' value='1.1' />

  <material name="blue"><color rgba="0 0 0.8 1"/></material>

  <link name="base">
      <visual>
      <geometry><box size="${length} ${width} ${height}"/></geometry>
      <material name="blue"/>
      <origin xyz="0 0 0"/>
      </visual>
      <collision>
      <origin xyz="0 0 0"/>
      <geometry><box size="${length} ${width} ${height}"/></geometry>
      </collision>
  </link>
</robot>
```

Defining a model - XACRO for Gazebo

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="zaza">
  <xacro:property name='width' value='2.2' />
  <xacro:property name='length' value='2.4' />
  <xacro:property name='height' value='1.1' />

  <xacro:macro name="box_inertial" params="mass length width height">
  <inertial>
       <mass value="${mass}" />
       <inertia ixx="${mass*(height*height+width*width)/12}" ixy="0.0" ixz="0.0" iyy="${mass*(length*length+width*width)/12}"
iyz="0.0" izz="${mass*(length*length+height*height)/12}" />
  </inertial>
  </xacro:macro>

  <material name="blue"><color rgba="0 0 0.8 1"/></material>

  <link name="base">
       <visual>
       <geometry><box size="${length} ${width} ${height}"/></geometry>
       <material name="blue"/>
       <origin xyz="0 0 0"/>
       </visual>
       <collision>
       <origin xyz="0 0 0"/>
       <geometry><box size="${length} ${width} ${height}"/></geometry>
       </collision>
       <xacro:box_inertial mass="2" length="${length}" width="${width}" height="${height}"/>
  </link>

  <gazebo reference="base"><material>Gazebo/Orange</material></gazebo>
</robot>
```

Defining a model - Updating launch file

```
…
  <arg name="model"/>

  <!-- vehicle pose -->
  <arg name="x" default="0"/>
  <arg name="y" default="0.5"/>
  <arg name="z" default="0.5"/>
  <arg name="roll" default="0.0"/>
  <arg name="pitch" default="0.0"/>
  <arg name="yaw" default="3.14"/>
…

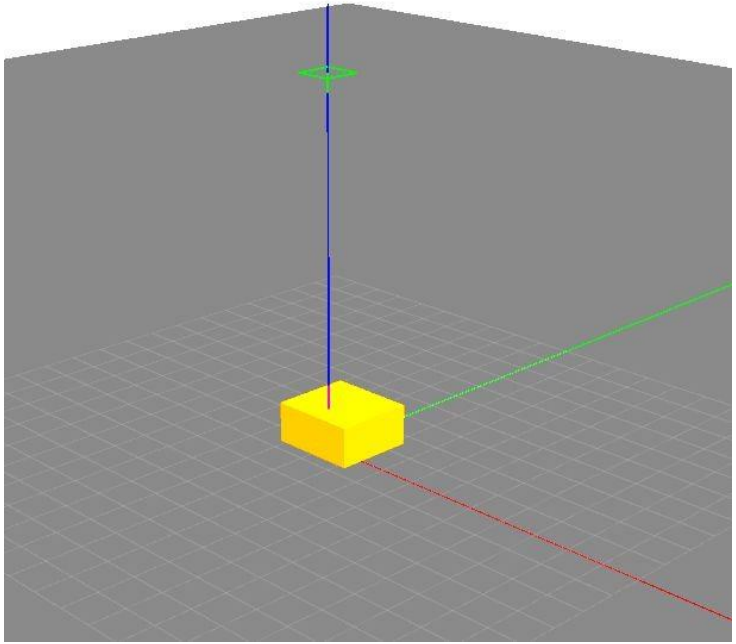

  <param name="robot_description" command="$(find xacro)/xacro $(arg model)"/>

  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
       args="-param robot_description -urdf -model robocar_robot
             -x $(arg x) -y $(arg y) -z $(arg z)
             -R $(arg roll) -P $(arg pitch) -Y $(arg yaw)"/>

…
```

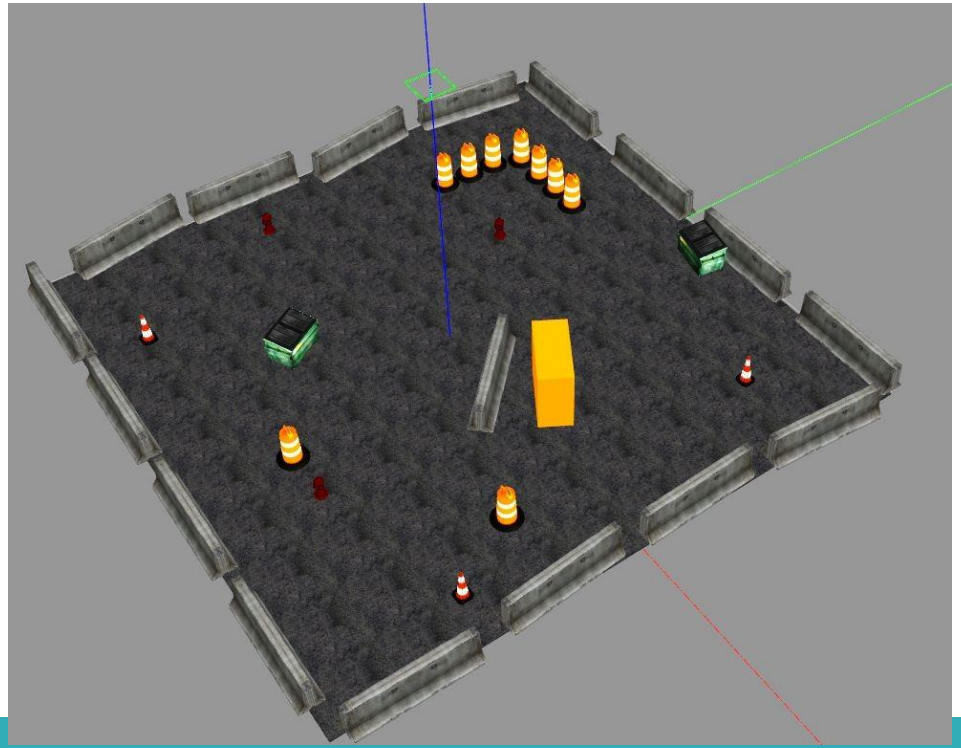
Defining a model - Results

```
# with optional parameters
# roslaunch robocar_simulation robocar_sim.launch model:=$(rospack find
robocar_ackermann_description)/urdf/dummy_model.xacro x:=3 z:=3 roll:=1.5
```



Defining a model - Results

```
# we still can run with another world
# with optional parameters
roslaunch robocar_simulation robocar_sim.launch model:=$(rospack find
robocar_ackermann_description)/urdf/dummy_model.xacro world_name:=$(rospack find
husky_gazebo)/worlds/clearpath_playpen.world x:=2 roll:=1.57
```



Let’s add some wheels - XACRO update

```
  <link name="axle_carrier"><xacro:null_inertial/></link>

  <joint name="axle_suspension" type="fixed">
       <parent link="base"/>
       <child link="axle_carrier"/>
       <origin rpy="0 0 0"/>
       <axis xyz="1 1 0"/>
  </joint>

  <joint name="axle" type="continuous">
       <parent link="axle_carrier"/>
       <child link="wheel"/>
       <origin rpy="${pi / 2} 0 0"/>
       <axis xyz="0 1 0"/>
       <limit effort="10" velocity="10"/>
  </joint>

  <link name="wheel">
       <visual>
       <origin xyz="1 1.5 0"/>
       <geometry><cylinder radius="0.3" length="0.1"/>
       </geometry>
       <material name="tire_mat"/>
       </visual>

       <collision>
       <origin xyz="0 1.5 0"/>
       <geometry><cylinder radius="0.3" length="0.1"/>
       </geometry>
       </collision>
       <xacro:box_inertial mass="0.2" length="0.2" width="0.2"
height="0.2"/>
  </link>
```

```
  <transmission name="axle_trans">
    <type>transmission_interface/SimpleTransmission</type>

    <joint name="axle">
       <hardwareInterface>hardware_interface/EffortJointInterface
</hardwareInterface>
    </joint>

    <actuator name="axle_act">
       <hardwareInterface>hardware_interface/EffortJointInterface
</hardwareInterface>
       <mechanicalReduction>1</mechanicalReduction>
    </actuator>
  </transmission>

  <gazebo reference="wheel">
       <mu1 value="200.0"/>
       <mu2 value="100.0"/>
       <kp value="10000000.0" />
       <kd value="1.0" />
       <material>Gazebo/Wood</material>
  </gazebo>

…

  <gazebo>
       <plugin name="gazebo_ros_control"
filename="libgazebo_ros_control.so">
       <robotNamespace>/</robotNamespace>
       </plugin>
  </gazebo>

…
```

Let’s add some wheels - launch.xml

```
# launch.xml

<?xml version="1.0"?>
<launch>

  ……

  <node
     name="controller_spawner"
     pkg="controller_manager"
     type="spawner"
     args="$(find
robocar_simulation)/config/wheel.yaml"
output="screen"/>

  <node
     name="wheel_controller"
     pkg="robocar_simulation"
     type="wheel_controller.py"
     required="true"
     args="" output="screen"/>

</launch>
```

```
# robocar_simulation/config/wheel.yaml

axle_ctrlr:
  joint: axle
  type: effort_controllers/JointPositionController
  pid: {p: 4.0, i: 0.0, d: 1.0}
```

Let’s add some wheels - controller

```
# imports ...

def value_generator():
       while True:
       for i in range(0, 100, 1):
              yield 4 * pi * i / 100.

       for i in range(100, 0, -1):
              yield 4 * pi * i / 100.

class WheelController(object):
   def __init__(self):
       rospy.init_node("wheel_controller")
       list_ctrlrs = rospy.ServiceProxy(
        "controller_manager/list_controllers", ListControllers
       )
       list_ctrlrs.wait_for_service()
       self._sleep_timer = rospy.Rate(3)
       wait_for_controller(list_ctrlrs, "axle_ctrlr")
       self.axle_pub = rospy.Publisher("axle_ctrlr/command",
Float64, queue_size=1)


   def spin(self):
       values = value_generator()
       while not rospy.is_shutdown():
              val = next(values)
              print("wheel_controller: publishing position
{}".format(val))
              self.axle_pub.publish(val)
              self._sleep_timer.sleep()
```

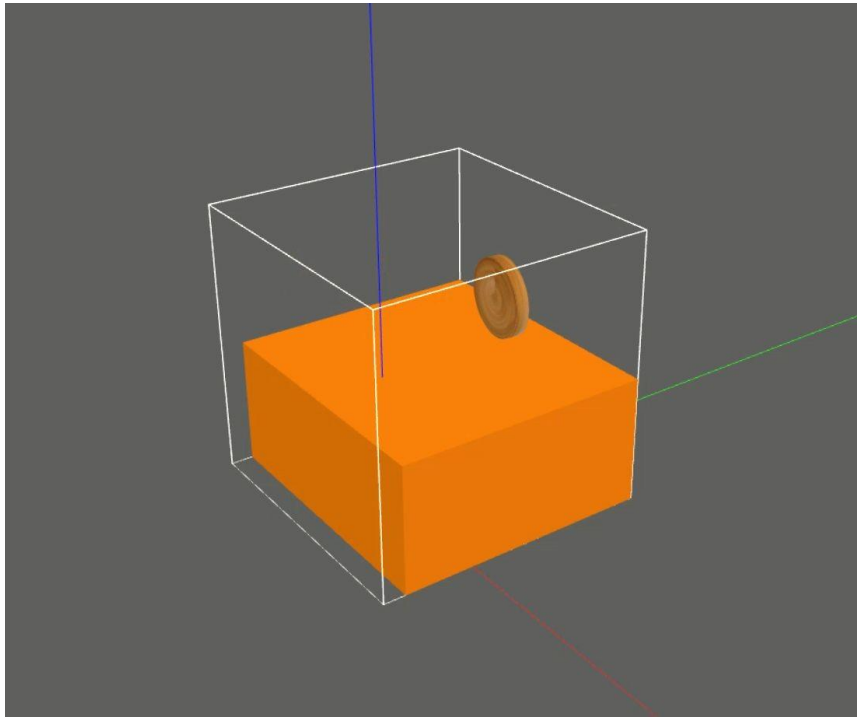
```
def wait_for_controller(list_ctrlrs, ctrlr_name):
    # not important


if __name__ == "__main__":
       ctrlr = WheelController()
       ctrlr.spin()
```

Launching

```
roslaunch robocar_simulation robocar_sim.launch model:=$(rospack find robocar_ackermann_description)/urdf/wheels.xacro
```

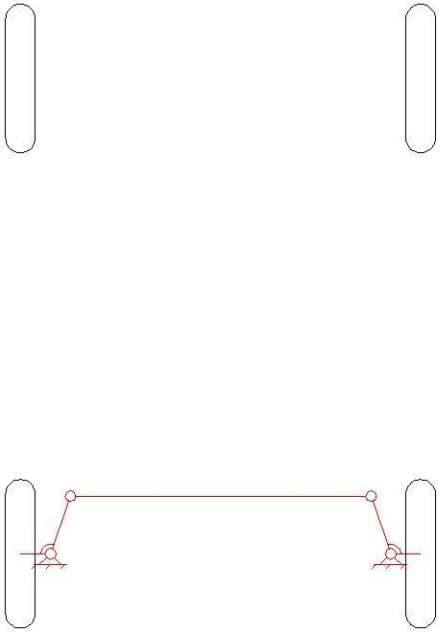
(yes, it rotates along the wrong axis, but you can fix it)
https://www.youtube.com/watch?v=P5NPJYlAosA

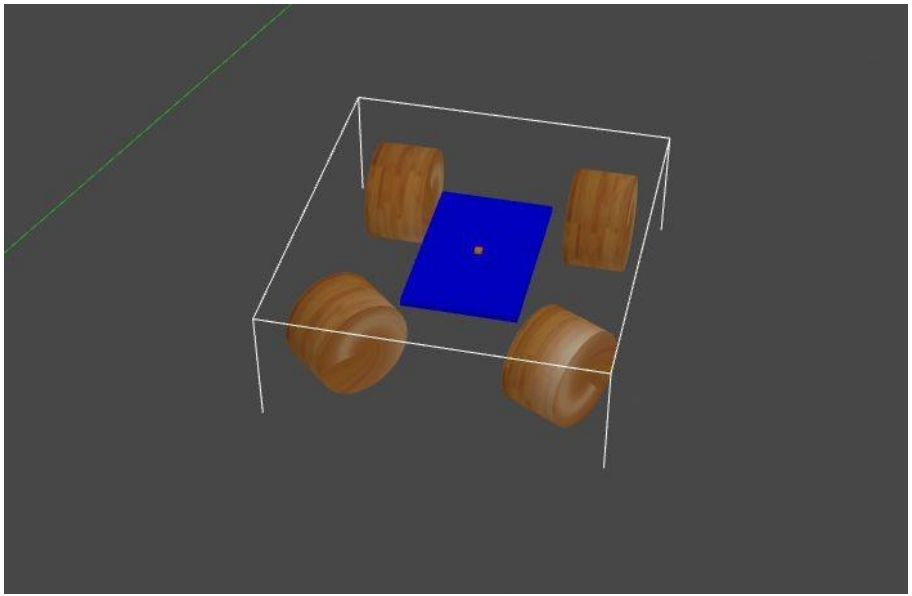


A realistic car with Ackermann steering

https://en.wikipedia.org/wiki/Ackermann_steering_geometry

https://github.com/trainman419/ackermann_vehicle-1, blog post





A realistic car with Ackermann steering

Changes in launch file:

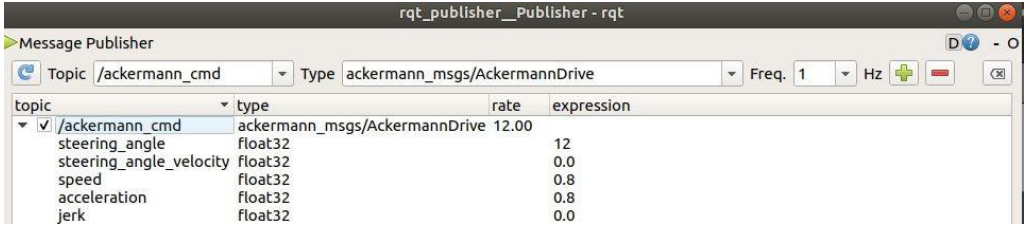
```
<node name="controller_spawner" pkg="controller_manager" type="spawner"
      args="$(arg joint_params)" output="screen"/>

<!-- Control the steering, axle, and shock absorber joints. -->
<node name="ackermann_controller" pkg="robocar_simulation"
      type="ackermann_controller.py"
      output="screen">
      <param name="cmd_timeout" value="$(arg cmd_timeout)"/>
      <rosparam file="$(find robocar_simulation)/config/em_3905_ackermann_ctrlr_params.yaml"
command="load"/>
</node>

<!-- required to get geometrical transformations from ackermann_controller -->
<node name="vehicle_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher">
      <param name="publish_frequency" value="30.0"/>
</node>
```

A realistic car with Ackermann steering

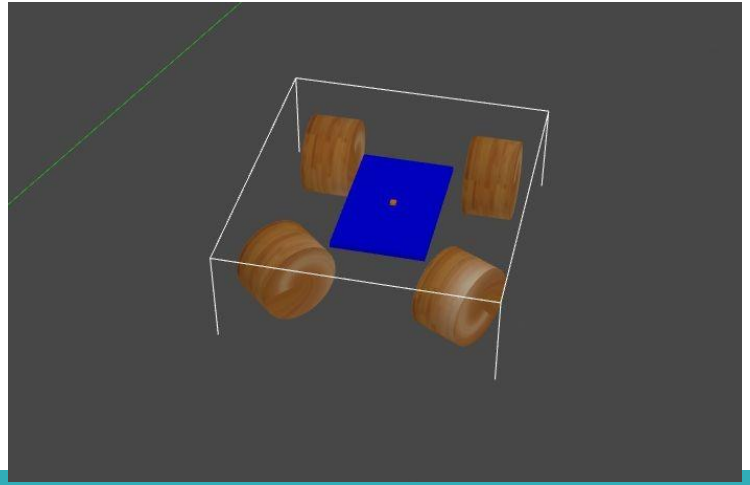
Publishing ackermann_msgs:



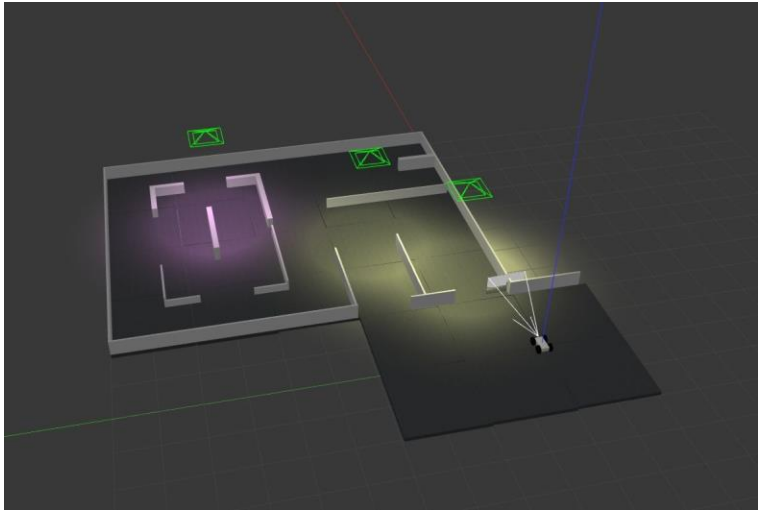
roslaunch robocar_simulation robocar_sim.launch model:=$(rospack find robocar_ackermann_description)/urdf/car.xacro joint_params:=$(rospack find robocar_simulation)/config/em_3905_joint_ctrlr_params.yaml

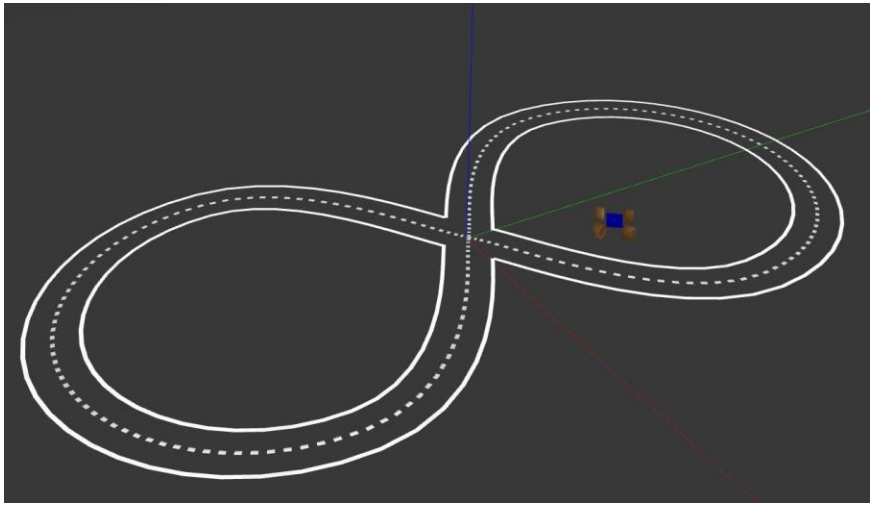
Results:

https://www.youtube.com/watch?v=oXl0L4rbr3o



Defining the track





https://github.com/project-omicron/gazebo_simulation

Defining the track

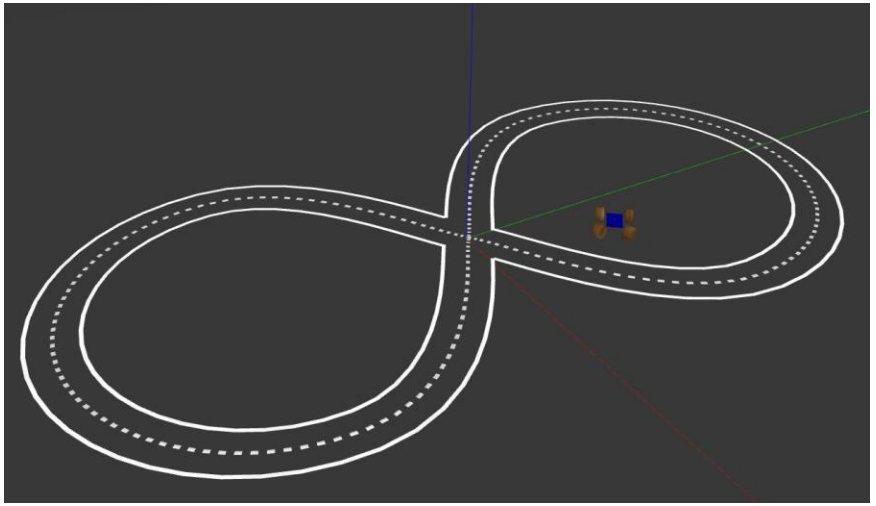
```
# in launch file:
<env name="GAZEBO_MODEL_PATH" value="$(find
robocar_world)/models"/>
```

```
# world (robocar_world/worlds/road.world)
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
        <scene>
        <grid>false</grid>
        </scene>
        <!-- A global light source -->
        <include>
        <uri>model://sun</uri>
        </include>

        <include>
        <uri>model://ground_plane</uri>
        </include>

        <include>
        <uri>model://lane_marking</uri>
        <pose>0.0 0.05 0.01 0 0 0</pose>
        </include>

        <include>
        <uri>model://lane_marking</uri>
        <pose>0.0 0.0 0.01 0 0 3.1415</pose>
        </include>
  </world>
</sdf>
```



Defining the track

```
$ tree models
models
└── lane_marking
    ├── materials
    │   ├── scripts
    │   │   ├── marking_dashed.material
    │   │   └── marking_solid.material
    │   └── textures
    │       ├── lane_markings.kra
    │       ├── marking_dashed.png
    │       └── marking_solid.png
    ├── model.config
    ├── model.sdf
    ├── README.md
    ├── track3.blend
    ├── track_solid.dae
    └── track_middle.dae

4 directories, 11 files
```

```
# model.config

<?xml version="1.0"?>
<model>
  <name>lane_marking</name>
  <version>1.0</version>
  <sdf version="1.5">model.sdf</sdf>
  <author>
       <name>Sergey M</name>
       <email></email>
  </author>
  <description>
       lane marking for a part of 8-shaped track
  </description>
</model>
```

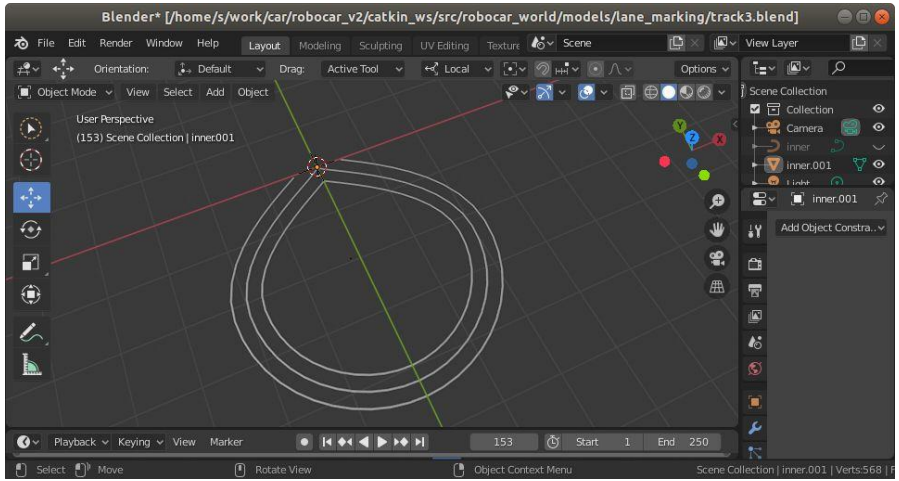
Defining the track (model.sdf)

```xml
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="lane_solid">
    <pose>0 0 0.01  0 0 0</pose>
    <static>true</static>
    <link name="solid">
      <inertial>
        <mass>1.0</mass>
        <inertia>
          <ixx>1</ixx><ixy>0</ixy><ixz>0</ixz>
          <iyy>1</iyy><iyz>0</iyz><izz>1</izz>
        </inertia>
      </inertial>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://lane_marking/track_solid.dae</uri>
          </mesh>
        </geometry>
        <material>
          <script>
            <uri>model://lane_marking/materials/scripts</uri>
            <uri>model://lane_marking/materials/textures</uri>
            <name>marking_solid</name>
          </script>
        </material>
      </visual>
    </link>
```

```xml
    <link name="dashed">
      <inertial>
        <mass>1.0</mass>
        <inertia>
          <ixx>1</ixx><ixy>0</ixy><ixz>0</ixz>
          <iyy>1</iyy><iyz>0</iyz><izz>1</izz>
        </inertia>
      </inertial>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://lane_marking/track_middle.dae</uri>
          </mesh>
        </geometry>
        <material>
          <script>
            <uri>model://lane_marking/materials/scripts</uri>
            <uri>model://lane_marking/materials/textures</uri>
            <name>marking_dashed</name>
          </script>
        </material>
      </visual>
    </link>
  </model>
</sdf>
```

Defining the track (3d model and materials)

3d models are designed in blender and exported to Colada format (*.dae)



marking_solid.material

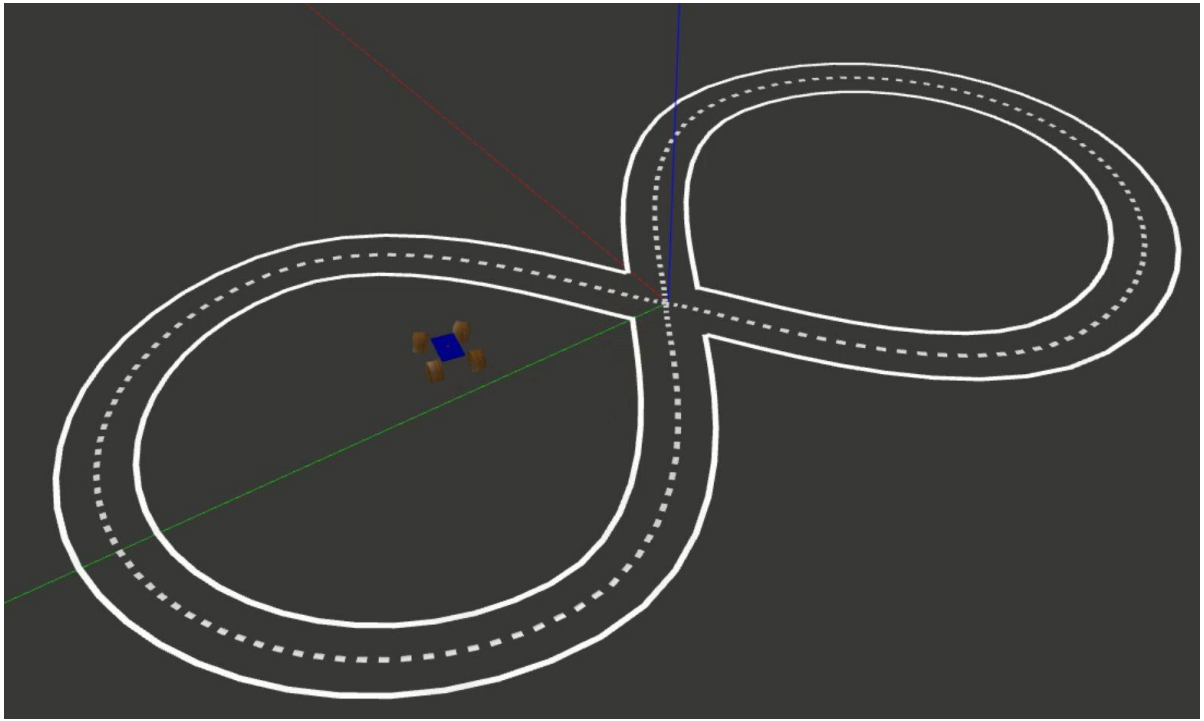
```
material marking_solid
{
    technique
    {
        pass
        {
            lighting off

            ambient 1 1 1 1
            diffuse 1 1 1 1
            specular 0 0 0 0
            emissive 0 0 0

            scene_blend alpha_blend
            depth_write off

            texture_unit
            {
                texture marking_solid.png
                tex_coord_set 0
                colour_op modulate
                scale 1.0 5.0
            }
        }
    }
}
```

marking_dashed.png

marking_solid.png

Results

```
rosrun rqt_publisher rqt_publisher &
roslaunch robocar_simulation robocar_sim.launch model:=$(rospack find robocar_ackermann_description)/urdf/car.xacro
joint_params:=$(rospack find robocar_simulation)/config/em_3905_joint_ctrlr_params.yaml world_name:=$(rospack find
robocar_world)/worlds/road.world
```



https://youtu.be/vllWGL7J_i8

Thank you