"Control of Mobile Robots" Project

Colli Stefano, Pagani Mattia, Panelli Erica

May 2022

1 Gazebo introduction

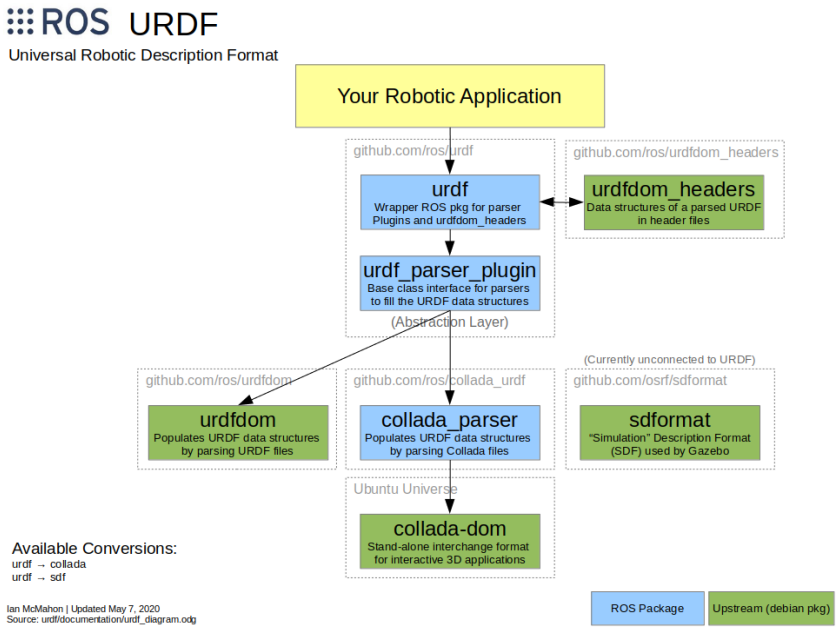
Before entering in the detail of the RACECAR model description we want to give a small introduction about different concepts and terminologies of the Gazebo simulator environment.

1.1 ODE Model Formulation

The ODE (*Open Dynamics Engine*) model formulation is a free library for simulating articulated rigid body dynamics. It is fast, flexible, robust and has built-in collision detection.
An articulated structure is created when rigid bodies of various shapes are connected together with joints of various kinds.
ODE is designed to be used in interactive and real-time simulations. It has hard contacts, this means that a special non-penetration constraint is used whenever two bodies collide. The alternative, is to use virtual springs to represent contacts, this is not always right, and can easily cause errors.

1.2 URDF

URDF (*Unified Robot Description Format*) is an XML format for representing a robot model. A number of different packages and components make up urdf.



URDF can only specify the kinematic and dynamic properties of a single robot in isolation, it cannot specify the pose of the robot itself within a world.

URDF is not a universal description format since it cannot specify joint loops (parallel linkages), and it lacks friction and other properties, and it cannot specify things that are not robots.
The XML structure makes URDF a inflexible model, so it has been developed a new format called SDF (*Simulation Description Format*) which is a complete description for everything from the world level down to the robot level, it is described in XML.
It has been developed also a macro language called *xacro* to make it easier to maintain the robot description files, increase their readability, and to avoid duplication in the robot description files.

1.3 Links and Joints in URDF formulation

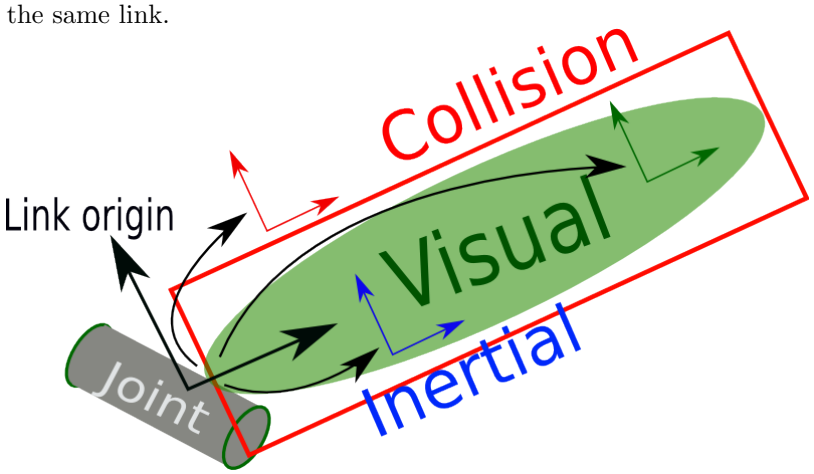
When a body has to be created in a xacro model a *Link* tag has to be specified. This new link will have its own space inside the model, this space can be divided into two different categories, *Visual* and *Collision* spaces. These will have different characteristics and purposes.
In addition when two links are created and they need to be connected a *Joint* tag must be specified.

1.3.1 Visual

The Visual space of a link specifies the shape of the object (e.g. box, cylinder) for visualization purposes. Multiple instances of visuals can exist for the same link.

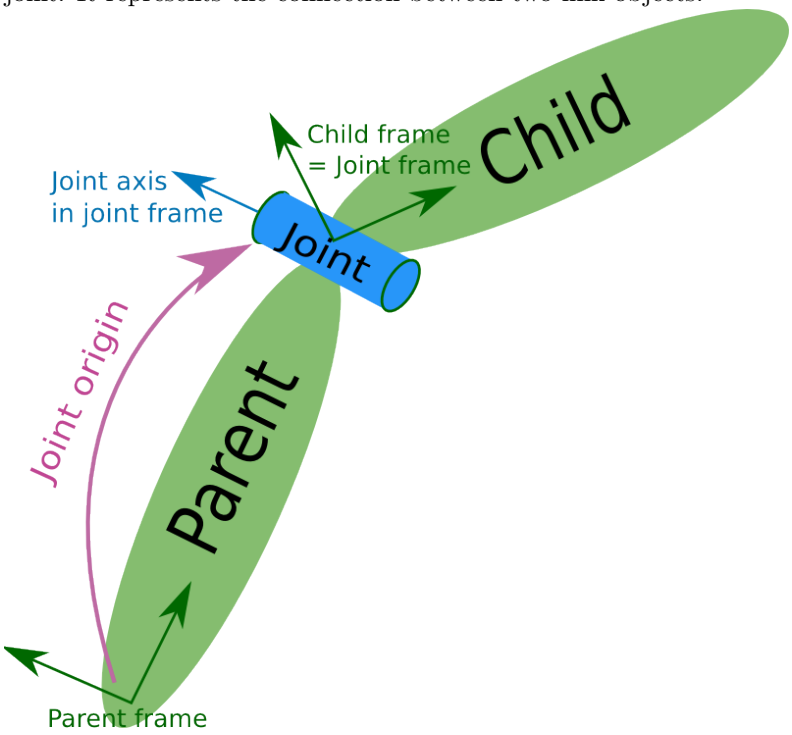
1.3.2 Collision

The Collision space can be different from the visual space of a link. Collision models are often used to reduce computation time, because they can be simpler than the correspondent visual one. Multiple instance of collisions can exist for the same link.



1.3.3 Joint

The joint element describes the kinematics, dynamics and safety limits of the joint. It represents the connection between two link objects.



2 RACECAR model description

In this section we are going to explain the structure of the *RACECAR project* developed by MIT, by focusing on the model the robot and on its simulation made by using Gazebo.
The robot is composed by different bodies: One chassis, four wheels, one Hokuyo laser sensor and one camera.

2.1 Chassis

The robot's chassis weights 4 $[kg]$ and since it has to be attached to the other bodies, its' visual origin, or in other words where the center of the visual element should be, was set without any rotation (rpy parameters equal to 0) and with an offset on the x axis (about 0.15 $[m]$). This will allow the car to be immediately aligned frontally with respect to the driving direction. The chassis has its own inertial matrix:

$$\begin{bmatrix} 0.010609 & 0 & 0 \\ 0 & 0.050409 & 0 \\ 0 & 0 & 0.05865 \end{bmatrix}$$

It is a frictionless contact, but has spring constant equals to $10 * 10^6 [N/m]$ and a damping constant equals to $1.0 [kg/s]$

2.2 Wheels

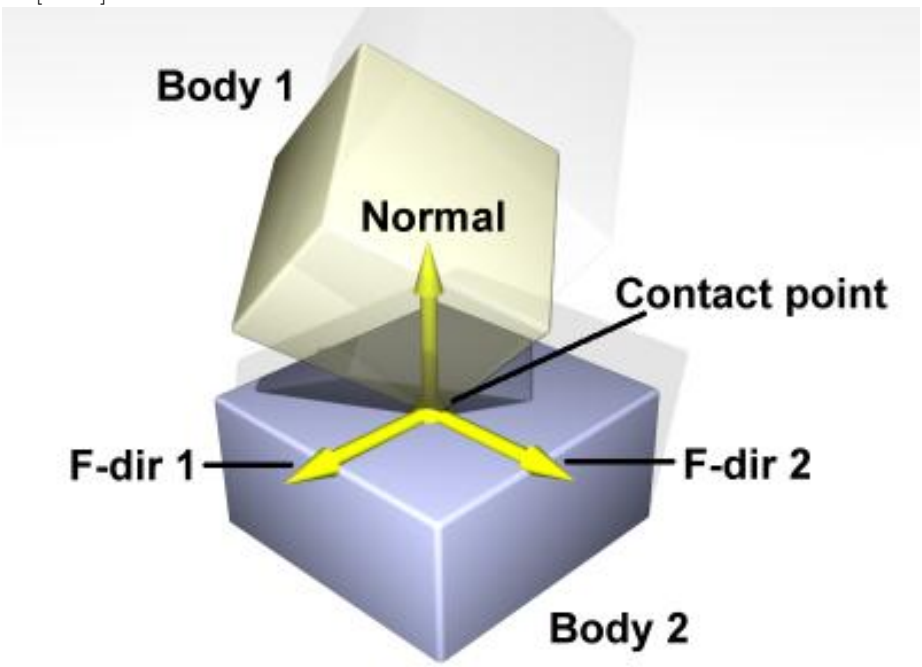
Racecar is a four-wheeled vehicle. Each wheel has its own inertial parameters which include initial positions, weight and inertial matrix. They all have in common the weight, which is equal to $0.34 [kg]$ and the inertial matrix, which is reported here:

$$\begin{bmatrix} 0.00026046 & 0 & 0 \\ 0 & 0.00026046 & 0 \\ 0 & 0 & 0.00041226 \end{bmatrix}$$

A difference is made in the origin of the wheels, right wheels are shifted about 0.0225 $[m]$ on z axis, and left wheels are shifted about -0.0225 $[m]$ on the z axis. The xyz and rpy parameters are all set to 0. This allow to make the wheels aligned with respect to the car's chassis.
Each wheel has also two friction coefficient both set to 1. These coefficients are named as μ and $\mu 2$, and are referring to the principal contact directions along the contact surface. The ODE (Open Dynamics Engine) which is used by Gazebo to model dynamics in simulation, uses the notion of *Coulomb Friction* which is the most commonly used friction model. The simplified friction's law is reported:

$$F_c = \begin{cases} \mu N * sign(v) \ if \ |v| > 0 \\ min(|F_{app}|, \mu N) * sign(F_{app}) \ if \ v = 0 \end{cases} \tag{1}$$

Where F_c is the Coulomb friction force, v is the sliding speed, μ is the coefficient of friction, N is the normal contact force and F_{app} is the applied force on the body.
Each wheel has also specified the spring constant ($10 * 10^6$ $[N/m]$) and the damping constant (1.0 $[kg/s]$).
Due to the frictional property of wheels it is necessary also to specify the first friction direction in the local reference frame along which frictional force is applied. It must be of unit length and perpendicular to the contact normal, it is typically tangential to the contact surface: rear wheels have a friction direction vector equal to [1 0 0], instead front wheels have a friction direction vector equal to [0 0 1].



Each wheel has its own collision space, it is delimited by a cylinder of $length = 0.045$ $[m]$ and of $radius = 0.05$ $[m]$. The origin of the collision spaces for right wheels correspond to $xyz = [0\ 0\ 0.0225]$, instead the origin of the collision spaces for the left wheels correspond to $xyz = [0\ 0\ -0.0225]$. All the collision spaces are not rotated with respect to the ground plane ($rpy = [0\ 0\ 0]$).

2.2.1 Wheels' mechanics

Racecar is a front wheels steering vehicle, which means that only the front wheels are attached to the chassis via a steering hinge, instead the rear wheels are attached with the chassis directly via common joints.

Rear wheels joints Both left and right rear wheels are attached to the chassis through direct joints. The dynamics of the joint is not specified and their safety limits concern effort and velocity only due to the *continuous* nature of these joints (they are rotating around the axis and have no upper and lower limits).
The effort on one joint has to be no more than 10 $[N]$, the controller commands the effort on the joint, if the controller tries to command an effort beyond the effort limit, the magnitude of the effort is truncated.
The velocity has to be no more than 100 $[m/s]$, this limit specifies the bounds on the magnitude of the joint velocity and it can be extended to the concept of effort, because it is not possible to apply an effort in such a way that it can push the joint beyond the velocity limits.

Steering hinge The steering hinges are programmed on left and right front wheels, both have same inertial parameters and geometry.
The steering hinges have both origin in the centre of xyz plane ($xyz = [0\ 0\ 0]$) and no rotations are applied to them ($rpy = [0\ 0\ 0]$). The weight is equal to 0.100 $[kg]$ and the inertial matrices are:

$$\begin{bmatrix} 4E-06 & 0 & 0 \\ 0 & 4E-06 & 0 \\ 0 & 0 & 4E-06 \end{bmatrix}$$

Steering hinge joints Right and left steering hinges are connected to the chassis through two separate joints.
Both steering hinge joints are rotated by design with a pitch of 1.5708 around y axis. Their safety limits concerns effort, velocity and the upper joint and lower joint limits. The limits for upper joints and lower joints are respectively 1.0 $[rad]$ and -1.0 $[rad]$, these limits are specified due to the *revolute* nature of the joints. A revolute joint is an hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
The effort limit is 10 $[N]$ and the velocity limit is 100 $[N]$.

Front wheels joints Front wheels are the ones that need the connection to the steering hinge, due to this reason they are not attached to the chassis, but to the right and left steering hinges themselves. The joints have both origin in the centre of xyz plane ($xyz = [0\ 0\ 0]$), but they are rolled around the x axis with a value of 1.5708.
The limits about effort and velocity are the same as the other continuous joints, effort limit is 10 $[N]$ and velocity limit is 100 $[m/s]$.

2.3 Hokuyo Laser

The RACECAR model is also equipped with a Hokuyo laser sensor.
The laser is programmed to have an update rate of $40[Hz]$, it has a ray which operates horizontally, the number of simulated rays to generate per complete laser sweep cycle is 1081 and it operates within the angle's interval: $[-2.356\ ;\ 2.356]$.

It has a noise distribution with $mean = 0$ and $standard\ deviation = 0.01$.
It has its own inertial parameters: its origin position is in the centre of the xyz plane ($xyz = [0\ 0\ 0]$) and it is not subject to initial rotations ($rpy = [0\ 0\ 0]$); its weight is equal to $0.130[kg]$ and its inertial matrix is :

$$\begin{bmatrix} 4E-06 & 0 & 0 \\ 0 & 4E-06 & 0 \\ 0 & 0 & 4E-06 \end{bmatrix}$$

This component has its own collision space delimited by a box of dimensions [0.1 0.1 0.1] $[m]$. The origins of the visual and collision spaces coincide to the origin of the ground plane ($xyz = [0\ 0\ 0]$), without any rotation ($rpy = [0\ 0\ 0]$).

Hokuyo Laser Joints The laser sensor is attached to the model by a joint directly connected to the chassis. This connection is fixed, this is not really a joint because it cannot move, because of this all degrees of freedom are locked. This type of joint does not require to specify any additional information apart from the position and the connected elements.

2.4 ZED Camera

The last sensor attached to the RACECAR model is a ZED RGB camera.
This camera has an update rate of 30 $[Hz]$, the width of the camera is 640 $[px]$ and the height of the camera is 480 $[px]$. The image format is $B8G8R8$ and the images taken by this camera has to at a distance between 0.02 $[m]$ and 300 $[m]$. The noise presented by the instrument is gaussian and has $noise = 0.0$ and $standard\ deviation = 0.007$.
Its inertial parameters are composed by $weight = 1e{-}5\ [kg]$, its origin position which is in the centre of the ground plane and it has not been subjected to any rotation ($xyz = [0\ 0\ 0]$, $rpy = [0\ 0\ 0]$) and its inertial matrix is:

$$\begin{bmatrix} 1E-6 & 0 & 0 \\ 0 & 1E-06 & 0 \\ 0 & 0 & 1E-06 \end{bmatrix}$$

Its collision and visual spaces coincide, they are both centered in the same position of the origin of the camera component and are not rotated, the delimitation is made by a box of dimensions [0.033 0.175 0.030].

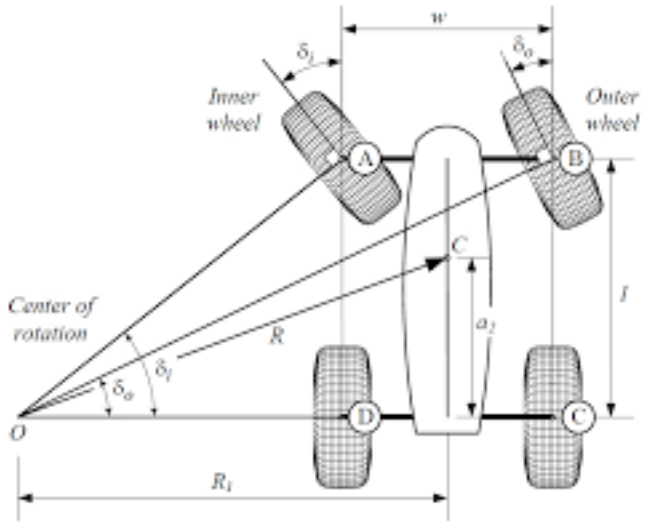
ZED Camera Joint This component is connected to the chassis through a fixed joint, in a similar way as the laser sensor: only the position, the parent and child connections are specified.

2.5 Ackermann steering model

The RACECAR model follows the Ackermann front-wheel steering model.
This model is helpful to solve the problem of independence between the rotations

described by the wheels: Ackermann steering ensures that all four tires have a common point around which they rotate when the car is turning. This ensures that none of the tires are required to slip in order to complete a turn.
The angle of the inside front tier will be greater than the angle of the outside front tier, this allows the car to rotate successfully around a single point, without slipping. If the angles of front tiers were equal the car won't be as stable as in the previous case, in particular it won't be stable with higher speed.
Each tier path has different radius around the point of rotation, this means that each tier is rotating at a different rate and the outside tiers will rotate at a higher speed than inside tiers. In addition when the car is rotating the front tiers will move faster than the rear tiers.
For a given *turn radius* R, *wheelbase* L and *track width* T it is possible to define two expressions for the front steering angles of the inner wheel ($\delta_{f,in}$) and of the outer wheel($\delta_{f,out}$).

$$\delta_{f,in} = \tan^{-1}(\frac{L}{R - \frac{T}{2}}) \qquad \delta_{f,out} = \tan^{-1}(\frac{L}{R + \frac{T}{2}}) \tag{2}$$



3 Plugins

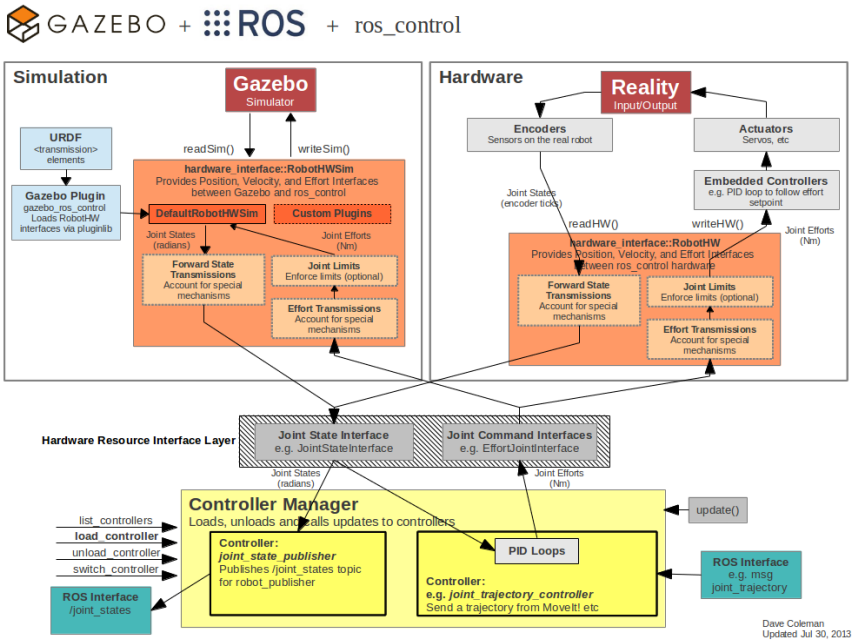
RACECAR model is enriched with functionalities that can control the several components specified before in the model description section.
These functionalities are developed as Plugins, which are equivalent to chunks of code, compiled as shared libraries and inserted into the simulation. A plugin has direct access to Gazebo's functionalities through C++ classes.
Plugins are used because of several positive aspects: they let developers control almost any aspect of Gazebo, they are self-contained routines that are easily shared and they can be inserted and removed easily from a running system.
There are six different types of plugins and each of them is managed by different Gazebo's components, these types are:

- World
- Model
- Sensor
- System
- Visual
- GUI

A plugin type should be chosen based on the desired functionality.
In this section we will analyze each model's plugin by specifying characteristics, functionalities and connections between components.

3.1 gazebo_ros_control plugin

The *gazebo_ros_control* plugin is a ros_control based controller out of URDF.
The *ros_control* packages take as input the joint's set data from the robot's actuators and an input set point. Its purpose is to control the output which is typically the effort sent to the robot's actuators. To perform its task the ros_control package uses a generic (typically PID) control loop feedback mechanism, it gets more complicated when there is not a one-to-one mappings of joint positions, efforts, etc.
The gazebo_ros_control plugin provides default ros_control interfaces and it also provides a pluginlib-based interface to implement custom interfaces between Gazebo and ros_control for simulating more complex mechanisms.
Hardware interfaces are used by ros_control to send and receive commands to the hardware, in the case of RACECAR project only one of the available interfaces is used, that is "Effort Joint Interface", for commanding effort, which is part of the derived class from the parent class named "Joint Command Interfaces", it is an hardware interface to permit the handling of an array of joints.



The used gazebo_ros_control library is *libgazebo_ros_control.so*. It is inserted in the model by specifying the *RobotSimType* tag with the value *DefaultRobotHWSim*: without this specification the gazebo_ros_control plugin will attempt to get all of the information it needs to interface with a ros_control based controller out of URDF, instead the model can work correctly only by take into account actuator's information presented in the URDF specification. DefaultRobotHWSim implements a simulated ros_control hardware interface which is *RobotHW*, it provides API-level access to read and command joints' properties in gazebo simulation.
For every joint that Gazebo has to actuate it is needed to add a transmission block, it is used to describe the relationship between an actuator and a joint. In the RACECAR model there are two kinds of transmission, concerning wheels and the steering hinge.

3.1.1 Wheel transmission

The wheel transmission is specified as a macro, this permits to describe the transmission one time and make it available for each one of the four wheels.
The type is specified as a simple transmission,
The hardware interface for the joint and the actuator connected through the transmission is *EffortJointInterface*. The actuator has a mechanical reduction of 1.

3.1.2 Steering hinge transmission

The steering hinge transmission is specified as a macro and it is used only for the joints between the chassis and the right and left steering hinges.
This is a simple transmission and the used hardware interface is equal both for the joint and the actuator connected, it is *EffortJointInterface*. The actuator has also a mechanical reduction of 1.

3.2 Laser plugin

The Hokuyo laser plugin is specified in a *Sensor* tag. It refers to the library *libgazebo_ros_laser.so* which takes the role of controller of the laser (named *gazebo_ros_hokuyo_controller*).
This controller gathers range data from a simulated ray sensor, publishes range data through *sensor_msgs::LaserScan* ROS topic, the specified topic name is */scan* and the corresponding frame name is *laser*.

3.3 Camera plugin

Similarly as the laser scan plugin the camera plugin is specified in a *Sensor* tag. It refers to the library *libgazebo_ros_camera.so* which permits to control this sensor (the controller is named *camera_controller*).
This plugin has been set to be always on and with an update rate of 30 $[Hz]$ which matches the update rate of the camera.
Inside the specification of the plugin it is defined the *rostopic* in which the camera sensor will publish for both the image topic and the camera information topic. The main camera name is *camera/zed* but for the image topic it will publish on *camera/zed/rgb/image_rect_color* and for the camera info topic it will publish on *camera/zed/rgb/camera_info*.
The coordinate frame where the image is published (tf tree) is *camera_link*.

4 racecar_gazebo code description

In the previous section we have illustrated how the model has been made and its characteristics. In this section we want to explain how the code is organized and how the model works in a more technical way.
We will focus only on the *racecar_gazebo* package inside the *mit_racecar* repository.

4.1 racecar_description

The first folder that we want to take into consideration contains all specifications and files useful to describe the car model and the graphic of the elements which will compose various worlds in the simulation.
In the *URDF* folder there are four different files:

- *macros.xacro*
  In this file there are macros definitions for inertial parameters, geometries and transmissions for all the elements composing the car model. It is useful to have this file with all the same kinds parameters listed and which permits a more readable and organized code.

- *materials.xacro*
  This file contains all the matches between color names and RGB codes.

- *racecar.xacro*
  In this file are specified all links, joints and transmission that involve all the car's components.
  Inside the links are specified all the inertial and geometry parameters, concerning also colors of the simulated shapes.

- *racecar.gazebo*
  This file contains for each car's component all the friction parameters and in addition for the camera and the laser sensors contains also the setup parameters (e.g. update rate, resolution, noise distribution).
  It also contains plugins references to controllers of the model.

In the *models* folder there are the *.config* and *.sdf* files that are used to setup the visual elements during the simulation.
The last folder is named *meshes*, it contains *.STL* and *.dae* files which are the images used by Gazebo to show the environment and car's components.

4.2 racecar_gazebo

In this folder there are files used to launch the Gazebo simulation.
There is a *worlds* folder in which all *.worlds* files are present: These files contain the geometries and also the friction parameters of the environment in which the model will be simulated. The second folder is the *script* one, it contains one file named *gazebo_odometry.py*. It is a python file in which there is the description of the *Odometry Node*, this node is in charge of read and update the *pose* of the robot by following a specified update rate.
The OdomNode calls a timer callback with an update rate of 20 $[Hz]$, it reads from the topic */gazebo/link_states* the robot's pose and twist, then update them and publishes these new values in the topic */vesc/odom*.
The last folder is named *launch*and contains one launch file for each kind of world that can be specified.
A launch file contains several calls to other files to permit an easier start of the simulation:

- World file: the name of the world file that will be simulated and its path are reported.

- racecar.xacro file: it contains the robot's description.

- racecar_spawn: a spawner permits to automatically load and start a set of controllers at once, in this case it loads the gazebo_ros package.
- racecar_control.launch: it is a file that contains all the information to start all necessary controllers of car's components.
- racecar_v2 mux.launch: it refers to the racecar base folder of the pure ros version of the RACECAR model, it contains information to load the Ackermann steering commands and references.
- topic for the odometry handling: this reference sets up the two topic used by the odometry node.

4.3 racecar_control

This folder contains the logic of the car's controller.
In the first folder named *scripts* there are two files:

- *keyboard_teleop.py*
  The implementations of commands used to move the car with keyboard keys is written here. Who wants to move the machine has to open this file and follow the *W, A, S, D* pattern, by choosing another key different from these ones will stop the car and by making the combination *CTRL-C* the controller will be stopped.

```
Reading from the keyboard  and Publishing to AckermannDriveStamped!
---------------------------
Moving around:
        w
   a    s    d
anything else : stop
CTRL-C to quit
"""

keyBindings = {
  'w':(1,0),
  'd':(1,-1),
  'a':(1,1),
  's':(-1,0),
}
```

  Each key corresponds to a row of the *key bindings* matrix, the columns of this matrix correspond to the transformation applied to the speed and the steering angle of the car.
  The main function of this file is to implement a publisher of the main car's movement information (normalized by pressed key factors) in a specific timestamp, which will publish into the *AckermannDriveStamped* topic the speed, acceleration, jerk, steering_angle and steering_angle_velocity.

- *servo_commands.py*
  The main purpose of this file is to read the information published into the *AckermannDriveStamped* topic and redirect these information inside each specific component's controller topic of the car. There are four controllers, one for each wheel, in which will be published the $throttle = speed / 0.1$ and two controllers, one for the left and one for the right steering hinge, in which will be published the $steer = steeringangle$.

The second folder is named *config* and it contains only one file which is *racecar_control.yaml*. A YAML file loads Node configuration parameters in the ROS Parameter server, it contains the configuration about type, joint and pid (proportional-integral-derivative) parameters of three controller topic type:

- Joint State
- Velocity (it concerns the four controllers of the wheels)
- Position (it concerns the two controllers of the steering hinge)

The last folder is named *launch* and it contains three launch files regarding controller functionalities.

- *teleop.launch*
  It makes a simple include of a xml launch file which is necessary to make the user control the model.
- *gazebo_sim_joy.launch*
  It launches the file regarding the tunnel's setup and the teleop.launch file.
- *racecar_control.launch*
  This file is structured similarly as the launch files previously discussed. It contains several references to other file to easily call each process and quickly start the simulation. It has the reference about *Controller Manager*, this manager contains the information to load, unload and call updates to controllers, it provides a loop to control the robot mechanism represented by the hardware_interface::RobotHW instance. When loading a controller the controller manager will use the load controller name as the root for all its parameters (these will identify which plugin to load). The *spawner* is set to automatically load and start a set of controllers at once (it can be used also to automatically stop and unload them). In addition to the controller manager's node, this file includes also the remapping of the main topic used to publish and subscribe by the robot state publisher (joint states), the robot's movements controller (servo_commands.py) and the odometry publisher.

4.3.1 ackermann_msgs Package

The RACECAR model uses Ackermann steering as a model for the wheels' movement. To implement this kind of model has been used the library provided

by the ROS Ackermann steering group, it defines ROS messages for vehicles using front-wheel Ackermann steering.
The ackermann_msgs package includes the AckermannDriveStamped, which is the class used by the publisher inside the file *keyboard_teleop.py*, it will publish inside the $vesc/ackermann_cmd_mux/input/teleop$ topic the information specified by the Ackermann class:

- Steering angle
- Steering angle velocity
- Speed
- Jerk
- Acceleration

Then the main process handled by *servo_commands.py* will read this topic and then publish separately the data on six different topics, four for wheels and two for steering hinges, as explained in the previous paragraph. These six topics are instantiated by the racecar_control.launch file which can do this procedure by referencing to the *racecar* package present in the MIT RACECAR repository.
In the racecar package there is the complete handling of the steering mechanisms and information to let the car to follow the Ackermann steering model.