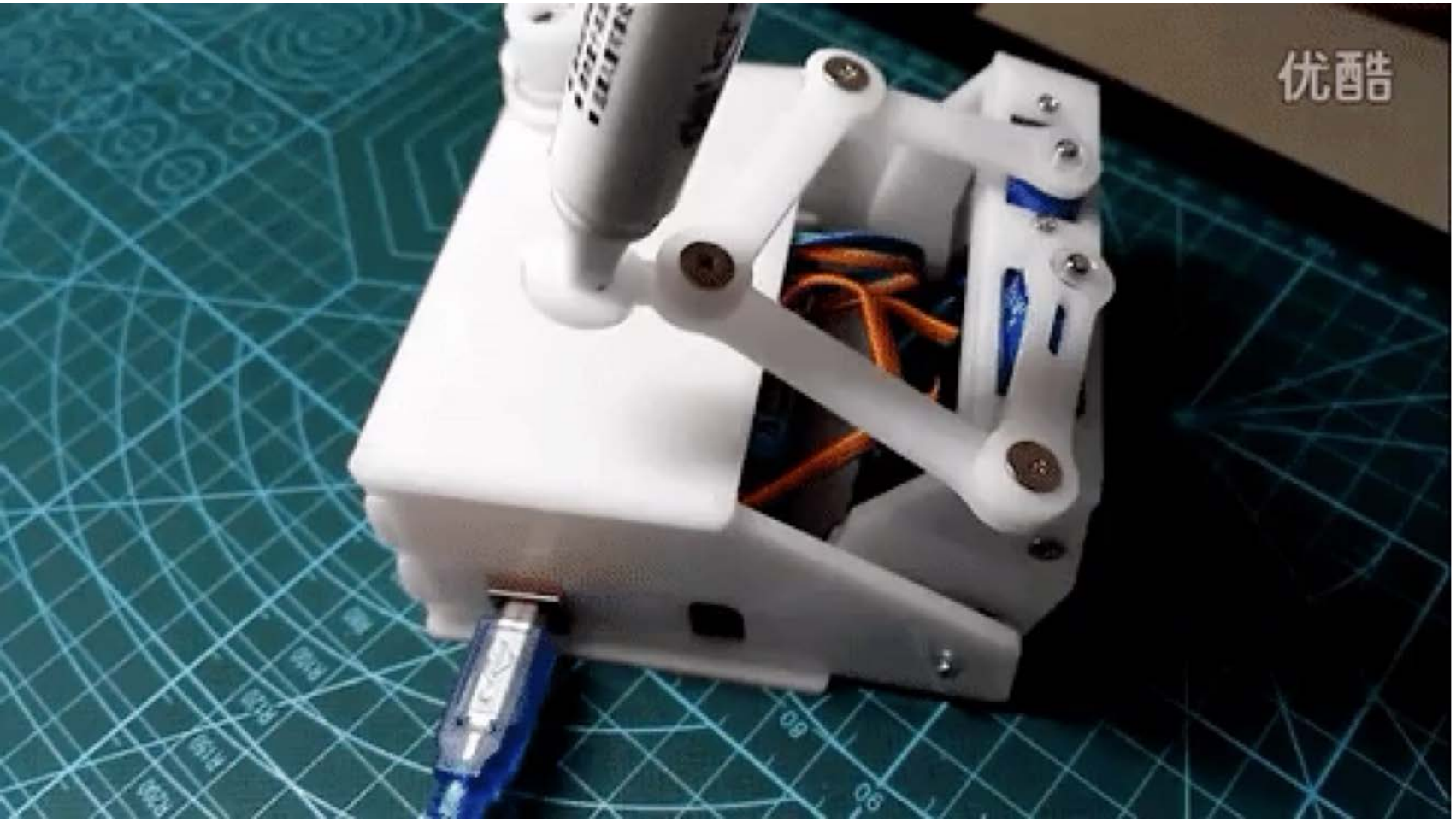


F1/10
Autonomous Racing

rospy.time
Rosbag
F1/10 Simulator

Madhur Behl

CS 4501
Rice Hall 120

Announcements

Assignment 1:

- Due next week

  Tuesday Feb 18th, 2020

  Before the lecture (3:30)pm.

  No extensions.

Demo:

- Submit your code by 3:30pm on Collab.
- Be ready to demo the problems.
  - Each problem describes what you need to demo.
  - Demo order is random – As long as you have submitted your code, the order does not matter.
- Be ready to explain your code.

So far..

- What is ROS.
- ROS workspaces
- ROS packages and build
- ROS nodes
- ROS topics
- ROS messages
- Publishers, and Subscribers
- ROS Services
- ROS Launch files
- rqt_graph
- Turtlesim
- ROS namespaces
- ROS Bags
- Time in ROS
- Gazebo Simulator
- Rviz
- ROS tf
- ROS over Network

ROS Bags

- A *bag* is a format for storing message data
- Binary format with file extension *.bag
- Suited for logging and recording datasets for later visualization and analysis

Record all topics in a bag

```
> rosbag record --all
```

Record given topics

```
> rosbag record topic_1 topic_2 topic_3
```

Stop recording with `Ctrl` + `C`
Bags are saved with start date and time as file name in the current folder (e.g. `2017-02-07-01-27-13.bag`)

rosbags: Recording and playing back data

```
mkdir ~/bagfiles
cd ~/bagfiles
rosbag record -a
```

rosbags: Examining and playing the bag file

```
rosbag info <your bagfile>
```

```
path:        2014-12-10-20-08-34.bag
version:     2.0
duration:    1:38s (98s)
start:       Dec 10 2014 20:08:35.83 (1418270915.83)
end:         Dec 10 2014 20:10:14.38 (1418271014.38)
size:        865.0 KB
messages:    12471
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]
             turtlesim/Color     [353891e354491c51aabe32df673fb446]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /rosout                  4 msgs    : rosgraph_msgs/Log   (2 connections)
             /turtle1/cmd_vel       169 msgs    : geometry_msgs/Twist
             /turtle1/color_sensor 6149 msgs    : turtlesim/Color
             /turtle1/pose         6149 msgs    : turtlesim/Pose
```

year, date, and time and the suffix .bag

rosbags: Playing back data

```
rosbag play <your bagfile>
```

```
[ INFO] [1418271315.162885976]: Opening 2014-12-10-20-08-34.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
```

rosbags: Playing back data

```
rosbag play -r 2 <your bagfile>
```

allows you to change the rate of publishing by a specified factor.

rosbags: Recording specific topics

```
rosbag record -O subset /turtle1/cmd_vel /turtle1/pose
```

-O argument tells **rosbag record** to log to a file named subset.bag

rosbags: Recording specific topics

```
path:        subset.bag
version:     2.0
duration:    12.6s
start:       Dec 10 2014 20:20:49.45 (1418271649.45)
end:         Dec 10 2014 20:21:02.07 (1418271662.07)
size:        68.3 KB
messages:    813
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /turtle1/cmd_vel    23 msgs    : geometry_msgs/Twist
             /turtle1/pose      790 msgs    : turtlesim/Pose
```

Sensors Udacity Lincoln MKZ

Camera 3x Blackfly GigE Camera, 20 Hz

Lidar Velodyne HDL-32E, 9.5 Hz

IMU Xsens, 400 Hz

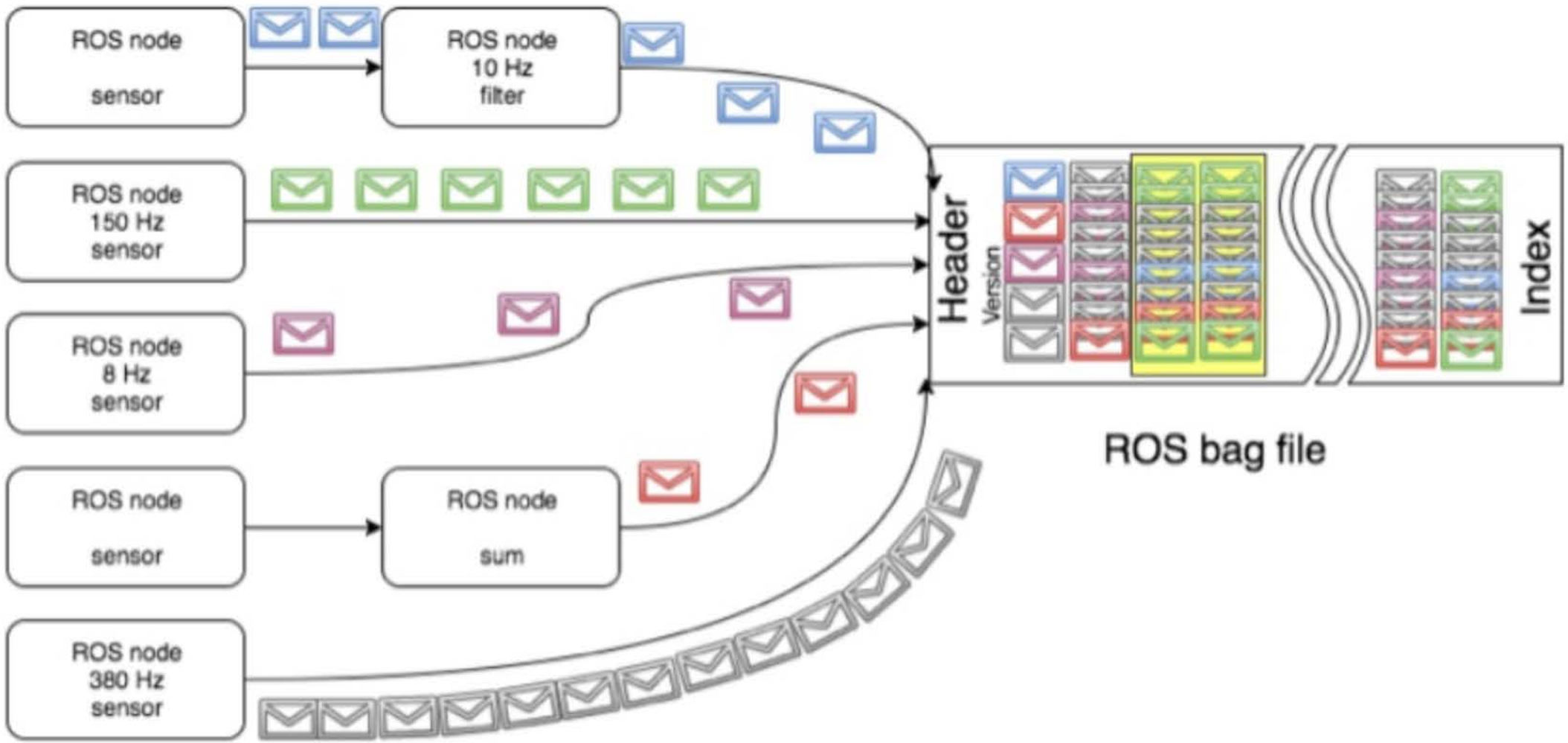
GPS 2x fixed, 1 Hz

CAN bus, 1,1 kHz

Robot Operating System

Data 3 GB per minute

ROS bag data structure



Training & Evaluation

+ Tensorflow ROSRecordDataset
+ Protocol Buffers to serialize records
+ Save time because data conversion not needed
+ Save storage because data duplication not needed

Ros bag

Ros msg

protobuf
Protocol Buffers

TensorFlow

ROS

ROS Dataset

hadoop

Machine Learning

Training Engine

Computer Network Storage

Re-Simulation & Testing

+ Use Spark for preprocessing, transformation, cleansing, aggregation, time window selection before publish to ROS topics
+ Use Re-Simulation framework of choice to subscribe to the ROS topics


Runtime Manager
Quick Start
Setup
Map
Sensing
Computing
Interface
Database
Simulation
Status
Topics
/media/ando/ssd/log/rosbag/rosbag.bag
Ref
Rate:
Start Time (s):
Repeat
Play
Stop
Pause
path: /media/ando/ssd/log/rosbag/rosbag.bag
version: 2.0
duration: 11:35s (695s)
start: Jun 23 2015 10:57:55.84 (1435024675.84)
end: Jun 23 2015 11:09:31.80 (1435025371.80)
size: 26.1 GB
messages: 17393
compression: none [17393/17393 chunks]
types: sensor_msgs/Image [060021388200f6f0f447d0fcd9c64743]
sensor_msgs/PointCloud2 [1158d486dd51d683ce2f1be655c3c181]
topics: /image_raw 10439 msgs : sensor_msgs/Image
/points_raw 6954 msgs : sensor_msgs/PointCloud2
ROSBAG
RViz
RQT
0.0%
0.0%
9.1%
0.0%
0.0%
0.0%
0.0%
0.0%
CPU0
CPU1
CPU2
CPU3
CPU4
CPU5
CPU6
CPU7
compiz (9.2 %CPU)
/sbin/init (0.0 %CPU)
[kthreadd] (0.0 %CPU)
[ksoftirqd/0] (0.0 %CPU)
[kworker/0:0H] (0.0 %CPU)
1GB/31GB(4%)
Memory
AutoWare

Demo with Turtlesim

$ mkdir ~/bagfiles
$ cd ~/bagfiles

$ roslaunch beginner_tutorials turtlemimic.launch

$ rosbag record -a

$ rosbag play <filename.bag>

$ rosbag play –r 2 <filename.bag>

ROS Time

January 1st, 1970 00:00:00

January 1st, 1970 00:00:00

UNIX Epoch time

In ROS messages timestamp is taken from your system, which is unix epoch time

/home/madhur/catkin_ws/src/beginner_tutorials/launch/chat.launch http://localhost:11311

```
[INFO] [1581438700.596499]: hello world 1581438700.6
[INFO] [1581438700.597287]: /listener1I heard hello world 1581438700.6
[INFO] [1581438700.681260]: hello world 1581438700.68
[INFO] [1581438700.682367]: /listener1I heard hello world 1581438700.68
[INFO] [1581438700.696267]: hello world 1581438700.7
[INFO] [1581438700.696858]: /listener1I heard hello world 1581438700.7
[INFO] [1581438700.782215]: hello world 1581438700.78
[INFO] [1581438700.783369]: /listener1I heard hello world 1581438700.78
[INFO] [1581438700.796334]: hello world 1581438700.8
[INFO] [1581438700.797256]: /listener1I heard hello world 1581438700.8
[INFO] [1581438700.881575]: hello world 1581438700.88
[INFO] [1581438700.882349]: /listener1I heard hello world 1581438700.88
[INFO] [1581438700.896454]: hello world 1581438700.9
[INFO] [1581438700.897361]: /listener1I heard hello world 1581438700.9
[INFO] [1581438700.981365]: hello world 1581438700.98
[INFO] [1581438700.982205]: /listener1I heard hello world 1581438700.98
[INFO] [1581438700.997082]: hello world 1581438701.0
[INFO] [1581438700.997705]: /listener1I heard hello world 1581438701.0
```

ROS Time

- Normally, ROS uses the PC's system clock as time source (*wall time*)
- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- To work with a simulated clock:
  - Set the `/use_sim_time` parameter

```
> rosparam set use_sim_time true
```

  - Publish the time on the topic `/clock` from
    - Gazebo (enabled by default)
    - ROS bag (use option `--clock`)

Time and Duration: rospy.Time rospy.Duration

Times and durations have identical representations:

```
int32 secs
int32 nsecs
```

May either represent wall clock time or ROS clock (simulated) time.

Duration can be negative

Time and Duration: rospy.Time rospy.Duration

Times and durations have identical representations:

```
int32 secs
int32 nsecs
```

Sec — Whole seconds
0 (default) | positive integer

Whole seconds, specified as a positive integer.

Note

The maximum and minimum values for `secs` are `[0, 4294967294]`.

Nsec — Nanoseconds
0 (default) | positive integer

Nanoseconds, specified as a positive integer. It this value is greater than or equal to 10^9, then the value is then wrapped and the remainders are added to the value of Sec.

rospy client library : Getting the current time

rospy.Time.now(), rospy.get_rostime() Equivalent

```
1 now = rospy.get_rostime()
2 rospy.loginfo("Current time %i %i", now.secs, now.nsecs)
```

rospy client library : Getting the current time

rospy.Time(secs=0, nsecs=0)

Create a new `Time` instance. `secs` and `nsecs` are optional and default to zero.

```
epoch = rospy.Time() # secs=nsecs=0
t = rospy.Time(10) # t.secs=10
t = rospy.Time(12345, 6789)
```

rospy client library : Sleeping

rospy.sleep(duration)

Toggle line numbers

```
1 # sleep for 10 seconds
2 rospy.sleep(10.)
3
4 # sleep for duration
5 d = rospy.Duration(10, 0)
6 rospy.sleep(d)
```

rospy.ROSInterruptException

rospy client library : Rate

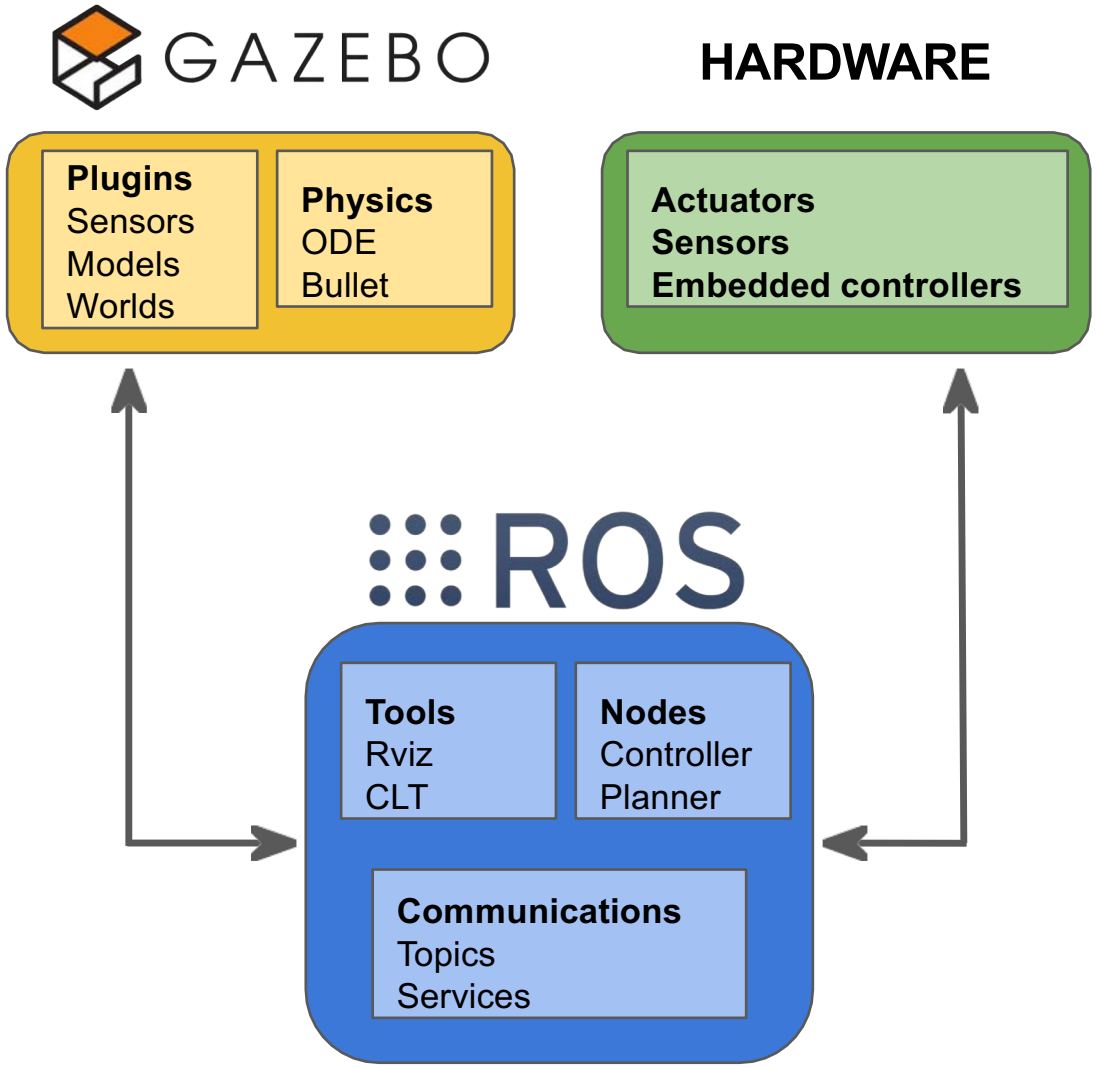
rospy.Rate(hz)

Toggle line numbers

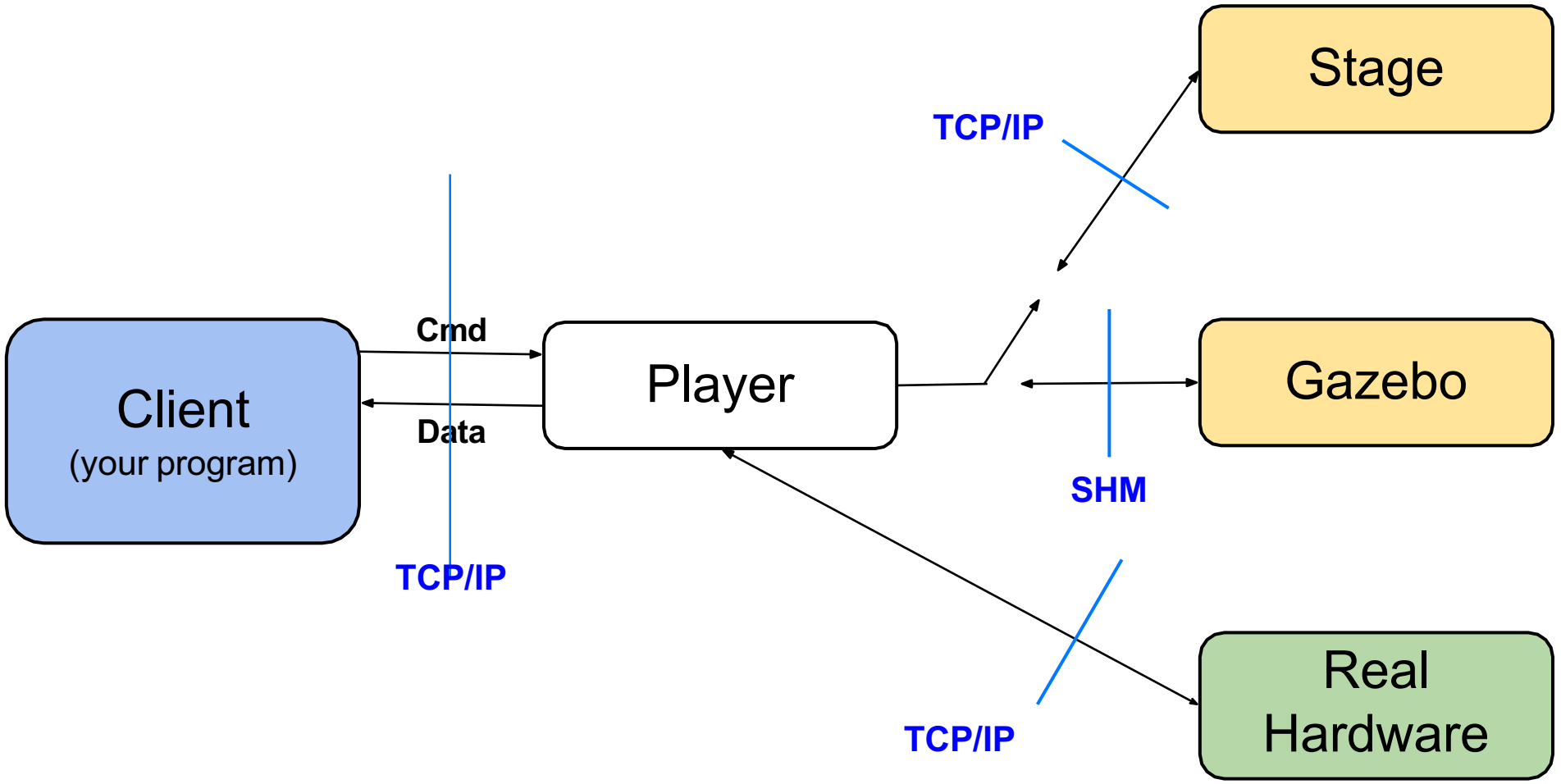
```
1 r = rospy.Rate(10) # 10hz
2 while not rospy.is_shutdown():
3     pub.publish("hello")
4     r.sleep()
```

ROS Simulation

- Simulators mimic the real world, to a certain extent
  - Simulates robots, sensors, and objects in a 3-D dynamic environment
  - Generates realistic sensor feedback and physical interactions between objects
- Why use them?
  - Save time and your sanity
  - Experimentation much less destructive
  - Use hardware you don't have
  - Create really cool videos



Simulation Architecture



Simulation Architecture

Gazebo runs two processes:

- **Server**: Runs the physics loop and generates sensor data.
  - Executable: *gzserver*
  - Libraries: Physics, Sensors, Rendering, Transport
- **Client**: Provides user interaction and visualization of a simulation.
  - Executable: *gzclient*
  - Libraries: Transport, Rendering, GUI

Run Gazebo server and client separately:

```
$ gzserver
$ gzclient
```

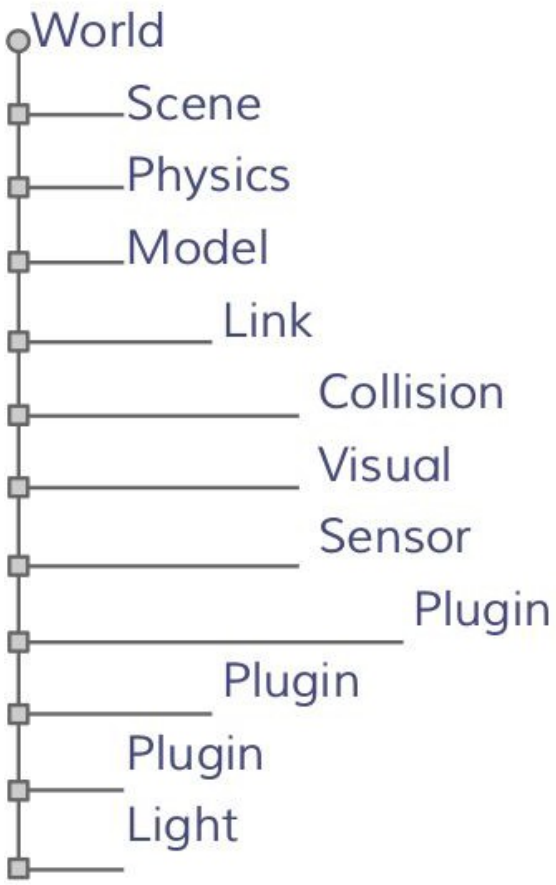
Run Gazebo server and client simultaneously:

```
$ gazebo
```

Elements within Simulation

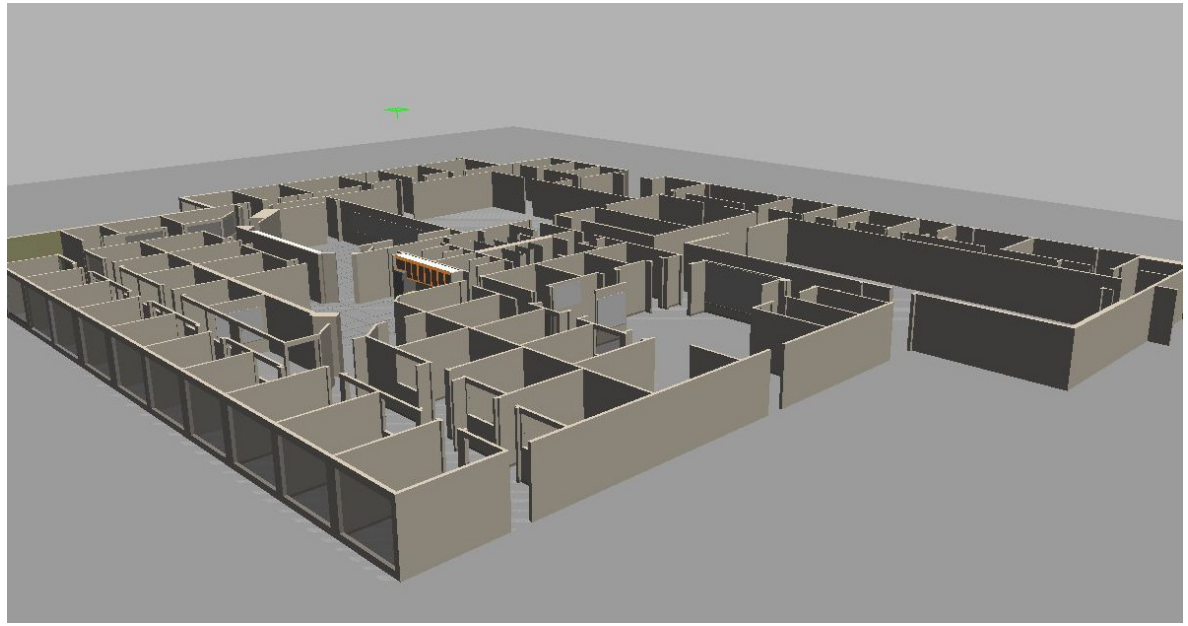
- World
  - Collection of models, lights,plugins and global properties
- Models
  - Collection of links, joints,sensors, and plugins
- Links
  - Collection of collision and visual objects
- Collision Objects
  - Geometry that defines a colliding surface
- Visual Objects
  - Geometry that defines visual representation
- Joints
  - Constraints between links
- Sensors
  - Collect, process, and output data
- Plugins
  - Code attached to a World, Model, Sensor, or the simulator itself

Element Hierarchy



World

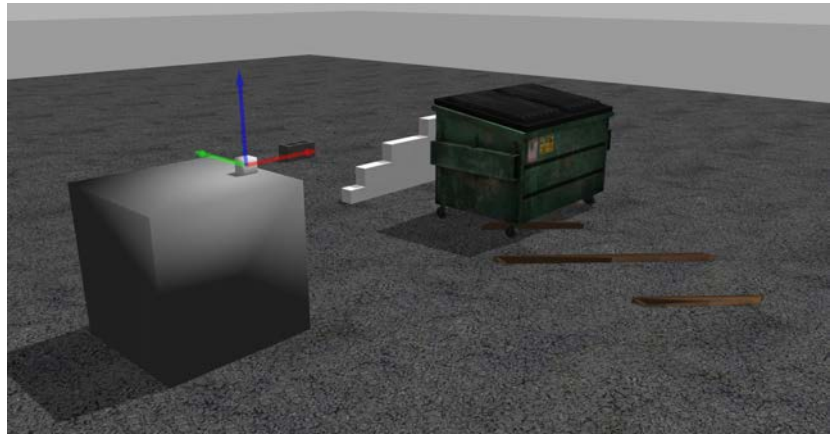
- A world is composed of a model hierarchy
- The Gazebo server (*gzserver*) reads the world file to generate and populate a world
  - This file is formatted using SDF (Simulation Description format) or URDF (Unified Robot Description Format)
  - Has a *".world"* extension
  - Contains all the elements in a simulation, including robots, lights, sensors, and static objects



Willow Garage World

Models

- Each model contains a few key properties:
  - **Physical presence** (optional):
    - Body: sphere, box, composite shapes
    - Kinematics: joints, velocities
    - Dynamics: mass, friction, forces
    - Appearance: color, texture
  - **Interface** (optional):
    - Control and feedback interface (libgazebo)

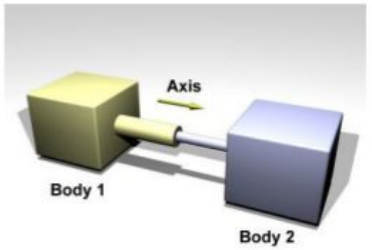


Element Types

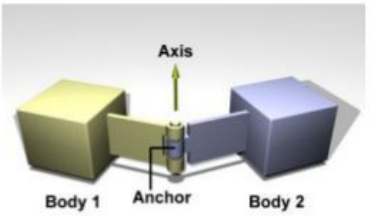
- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes

Element Types

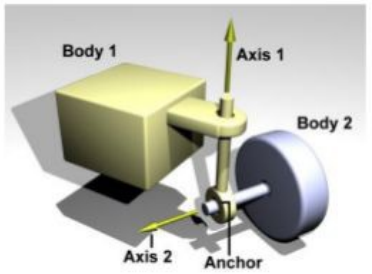
- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes
- Joints
  - Prismatic: 1 DOF translational
  - Revolute: 1 DOF rotational
  - Revolute2: Two revolute joints in series
  - Ball: 3 DOF rotational
  - Universal: 2 DOF rotational
  - Screw: 1 DOF translational, 1 DOF rotational



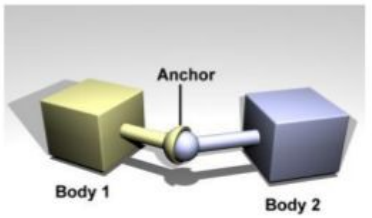
Prismatic



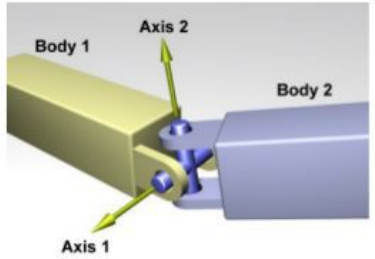
Revolute



Revolute 2



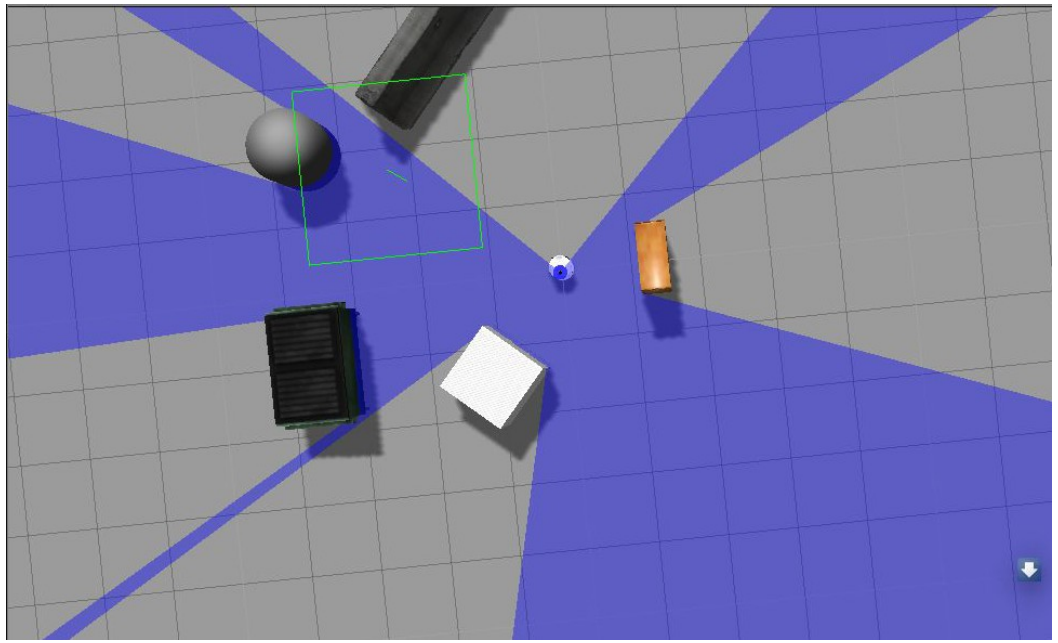
Ball



Universal

Element Types

- Sensors
  - Ray: produces range data
  - Camera (2D and 3D): produces image and/or depth data
  - Contact: produces collision data
  - RFID: detects RFID tags
- Lights
  - Point: omni-directional light source, a light bulb
  - Spot: directional cone light, a spot light
  - Directional: parallel directional light, sun

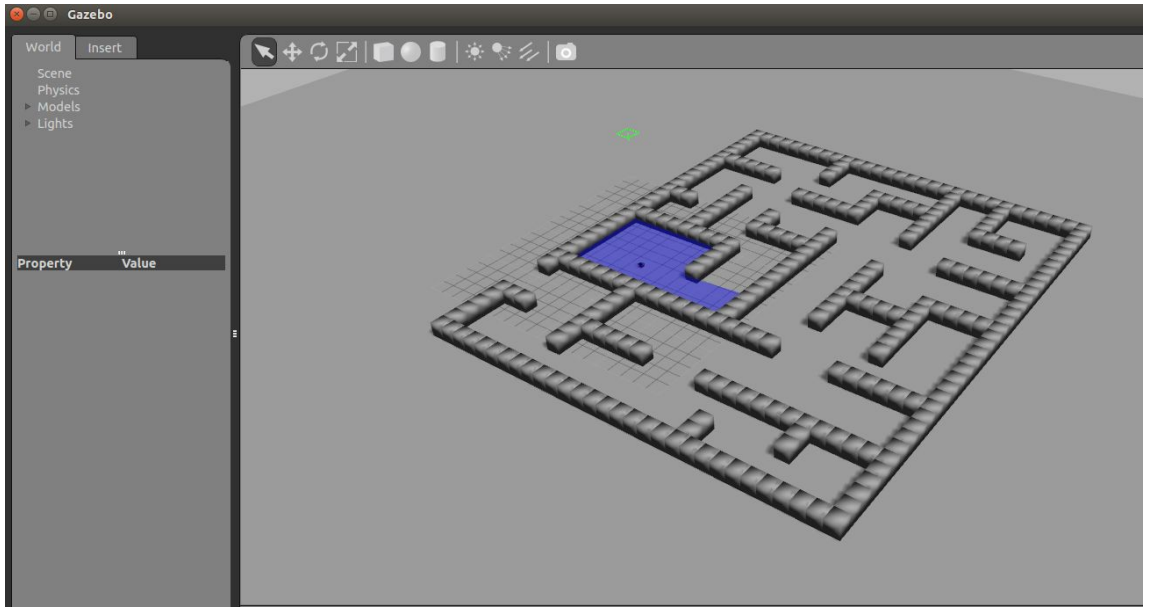


LiDAR sensor in Gazebo

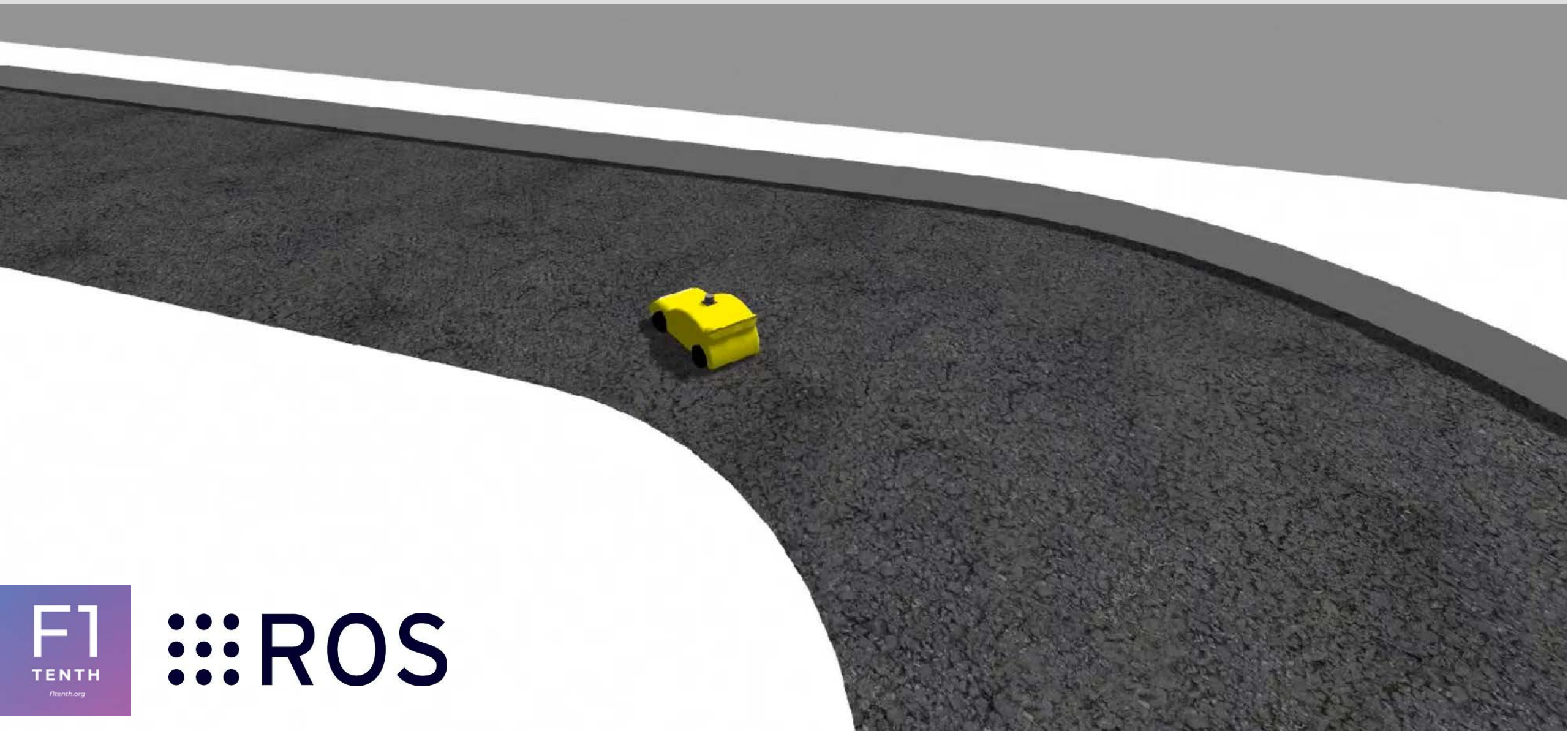
How to use Gazebo to simulate your robot?

Steps:

1. load a world
2. load the description of the robot
3. spawn the robot in the world
4. publish joints states
5. publish robot states
6. run rviz

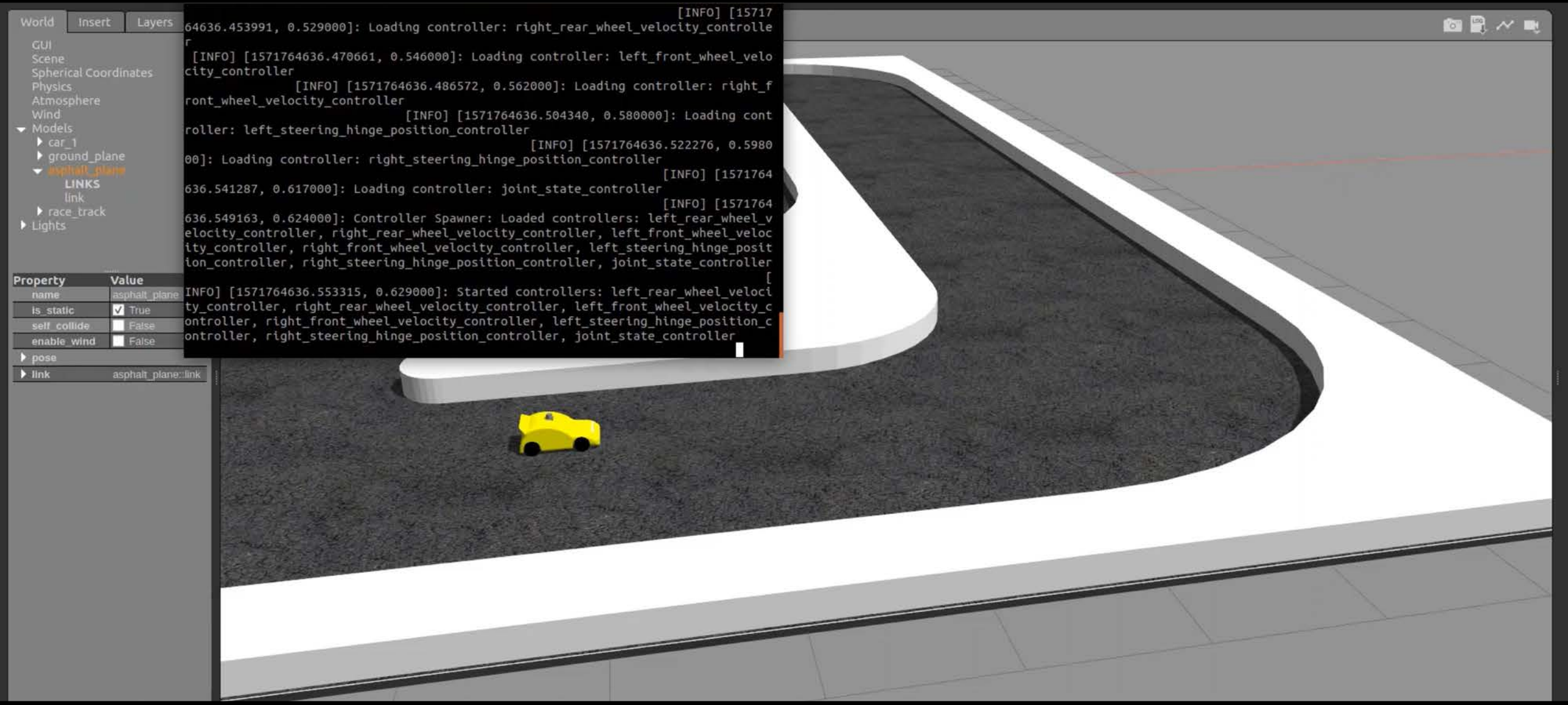


ROS F1/10 Autonomous Racing Simulator



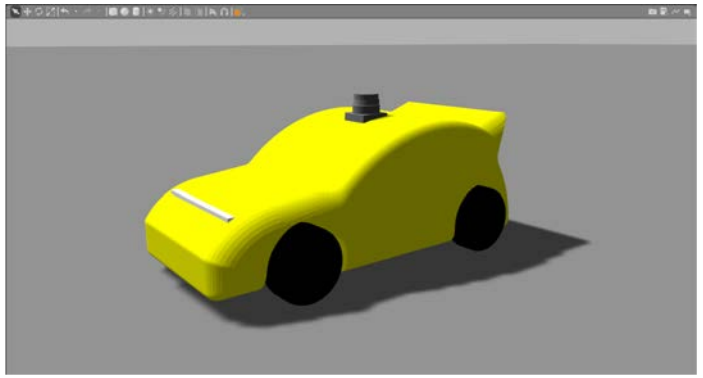
F1 TENTH
f1tenth.org

ROS



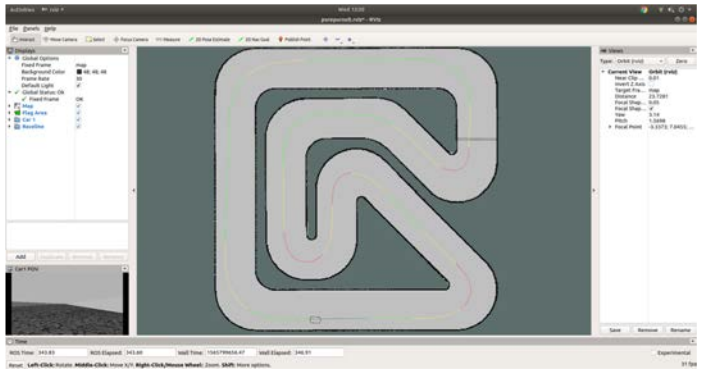
roslaunch f1tenth-sim simulator.launch

Simulator Elements



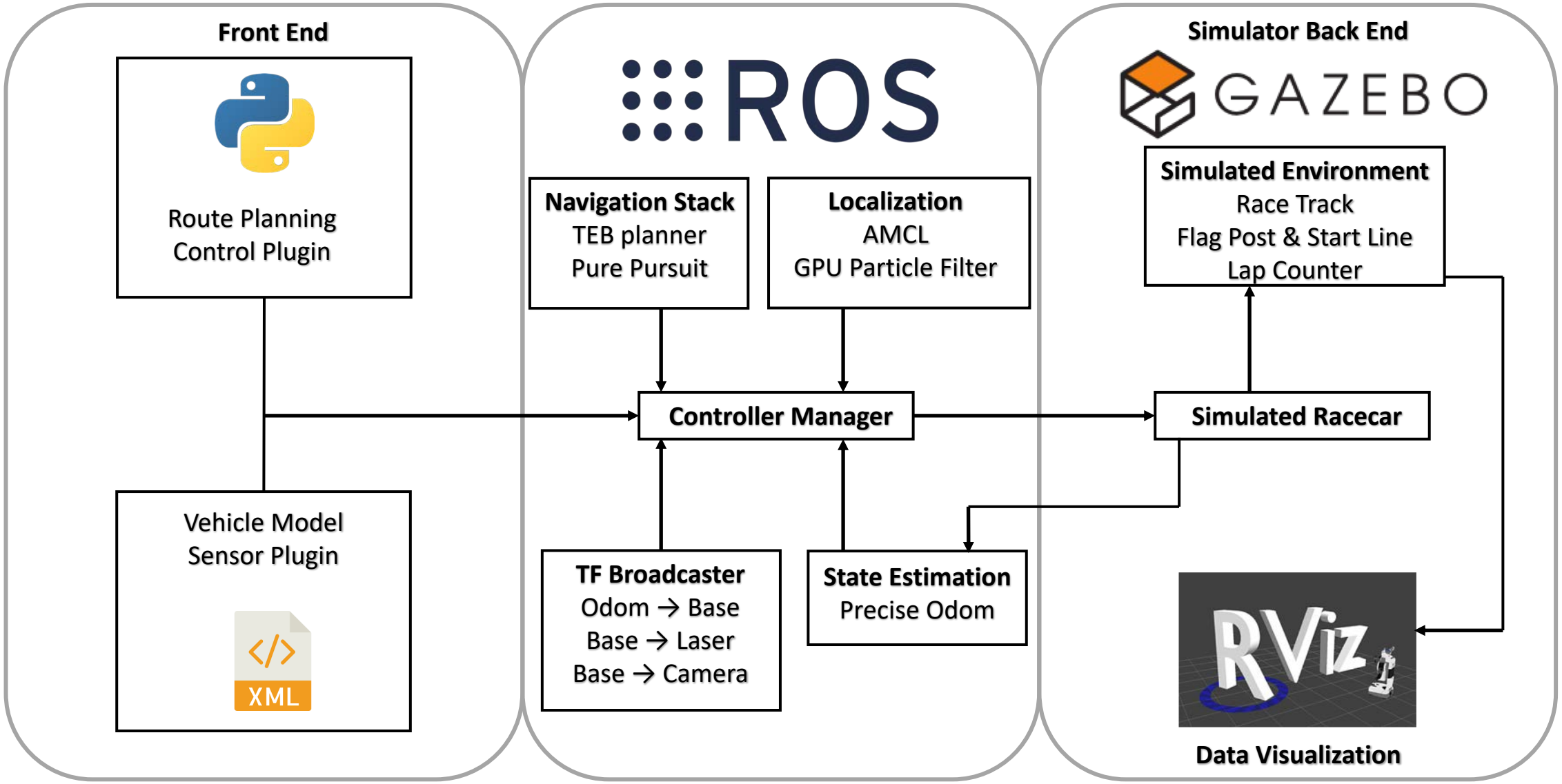
Racecar

Race Track

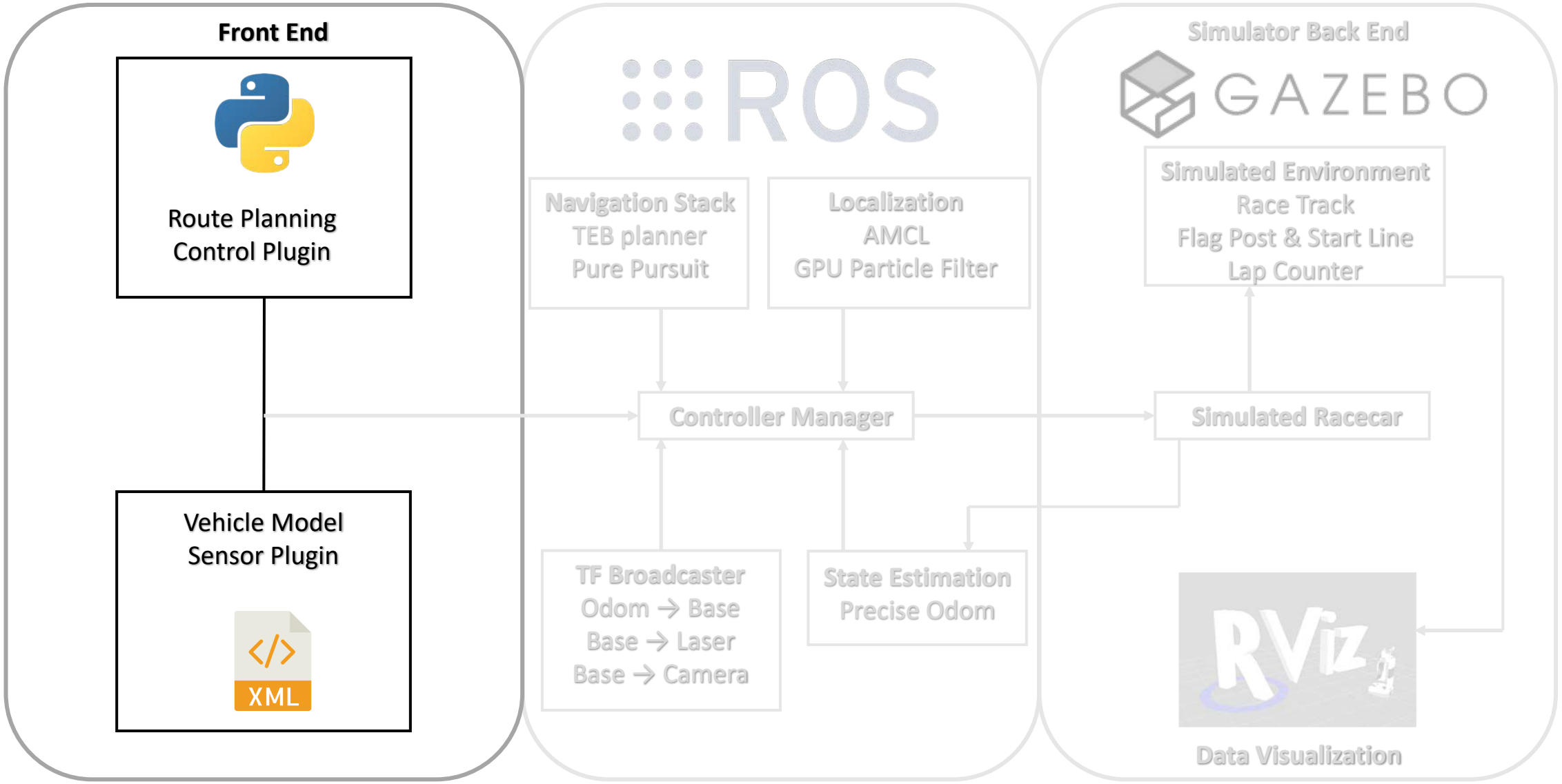


Algorithms

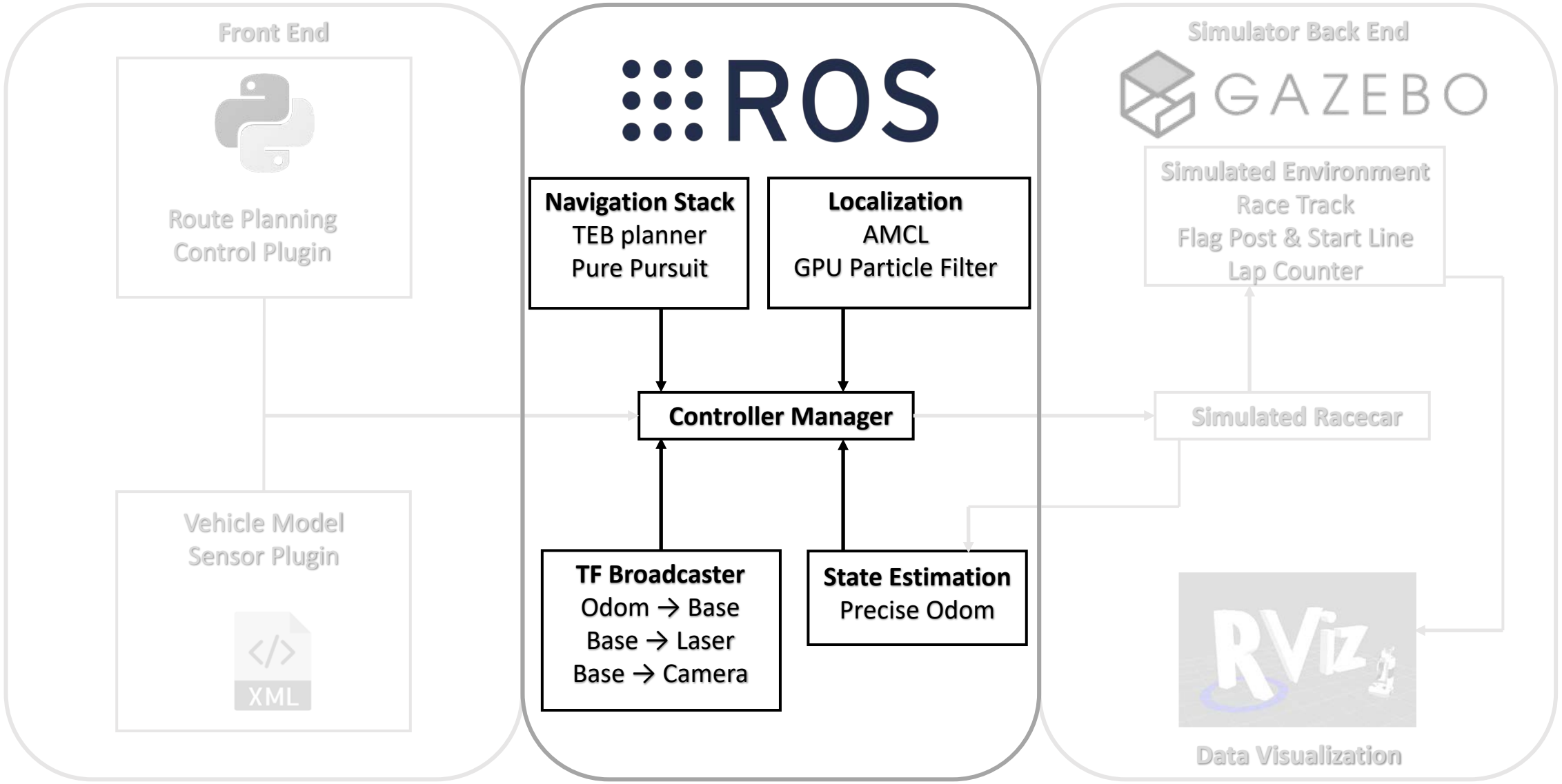
Simulator Architecture



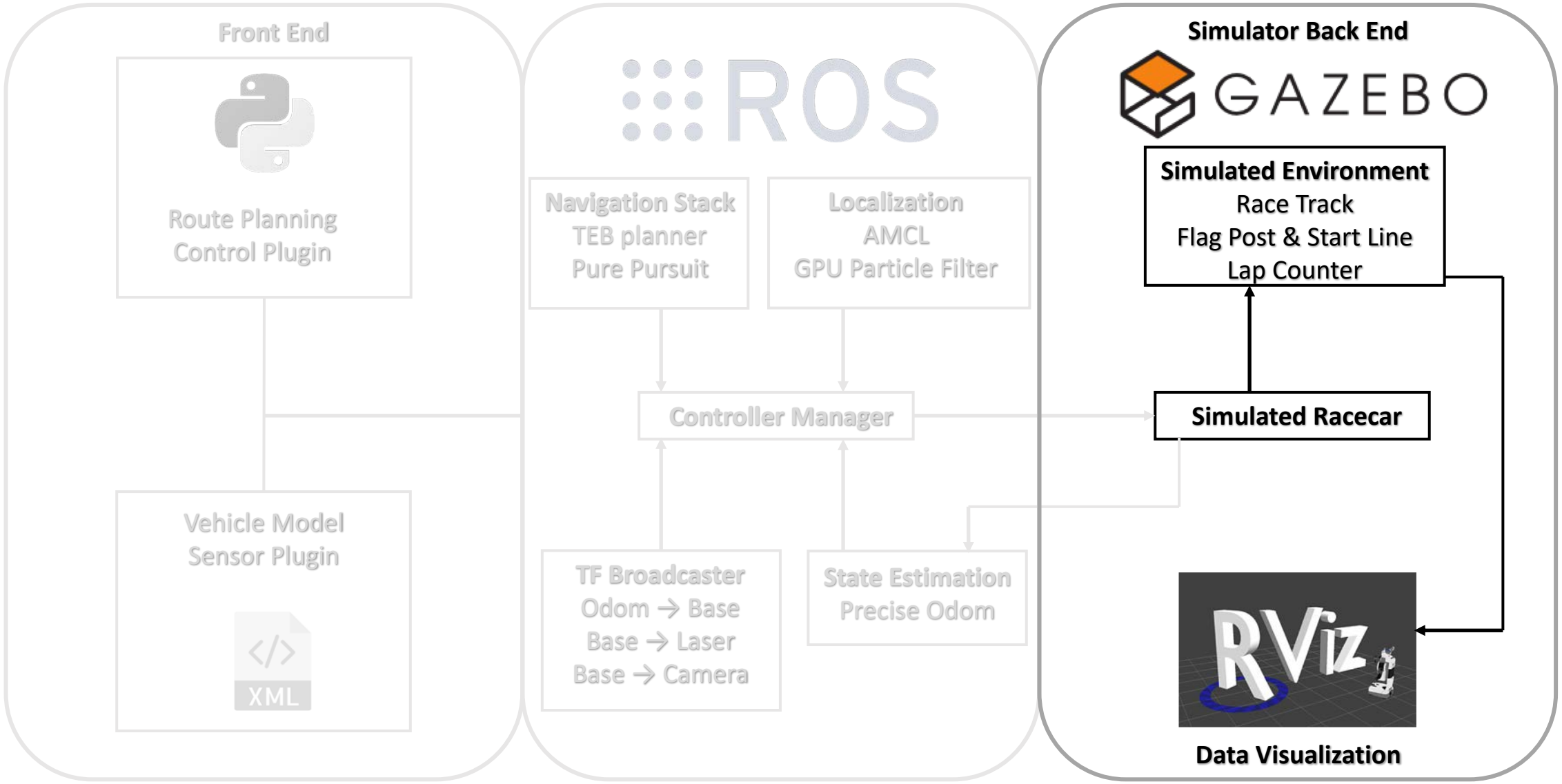
Front End


Front End
Route Planning
Control Plugin
Vehicle Model
Sensor Plugin
XML
ROS
Navigation Stack
TEB planner
Pure Pursuit
Localization
AMCL
GPU Particle Filter
Controller Manager
TF Broadcaster
Odom → Base
Base → Laser
Base → Camera
State Estimation
Precise Odom
Simulator Back End
GAZEBO
Simulated Environment
Race Track
Flag Post & Start Line
Lap Counter
Simulated Racecar
RViz
Data Visualization

Navigation and Control Algorithms



Gazebo and Visualization

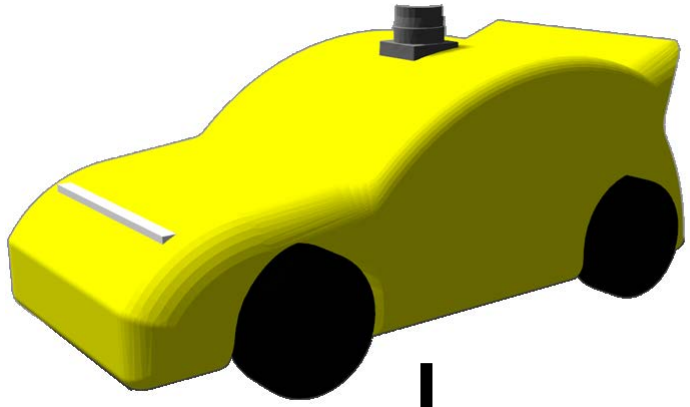


Racecar Model

F1/10 Autonomous
Racecar Platform

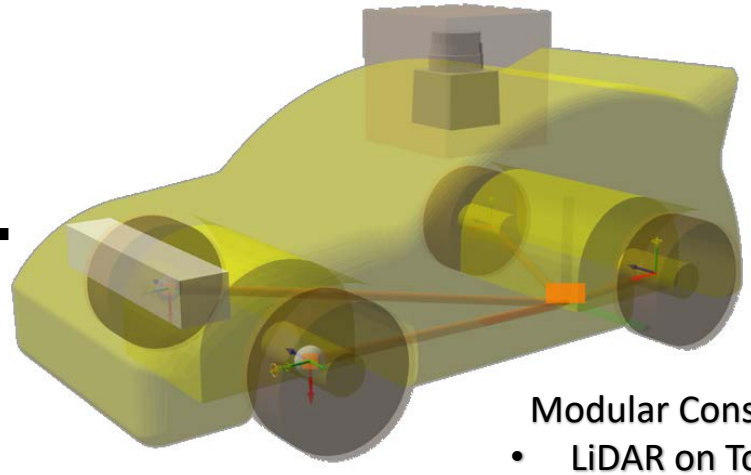
Size & Performance
Replicated

CAD, xacro, urdf



Simulated F1/10 Racecar
- Customizable color
- Multiple sensors

Gazebo
Sensor Plugins

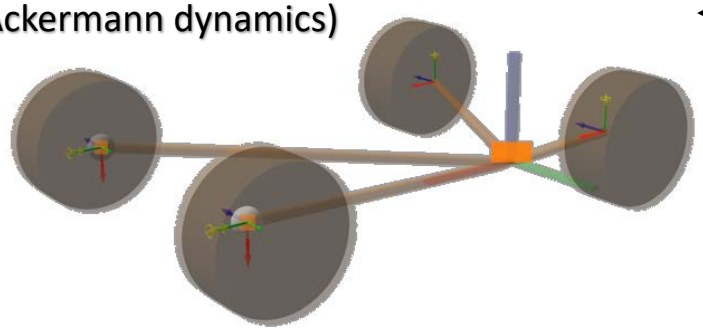


Modular Construction
- LiDAR on Top
- Camera on the hood

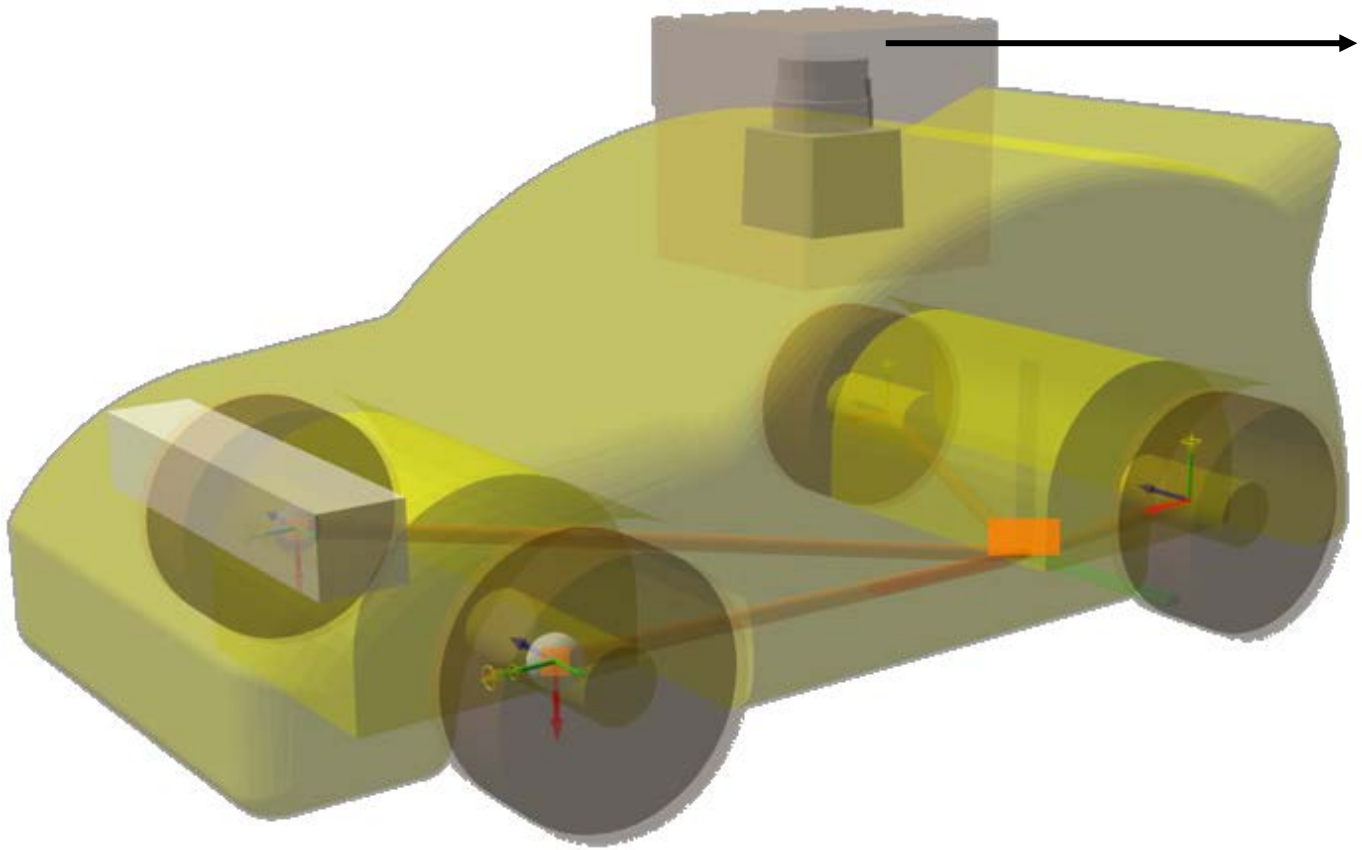
4 Wheel Independent Drive

(with hard differential constraints
enforced by control plugin)

Powertrain
(with Ackermann dynamics)



Modular Design



Primary Navigation Sensor

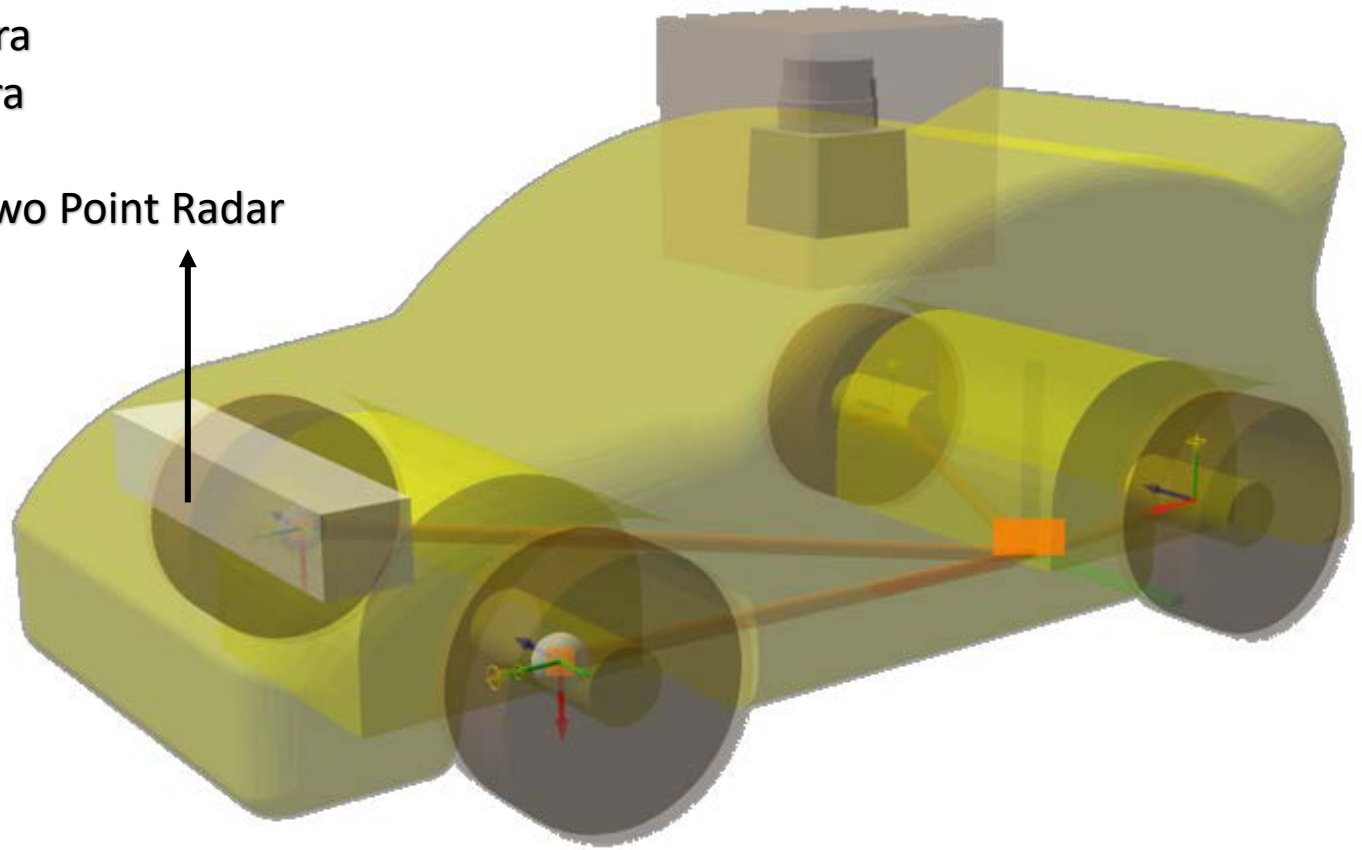
2D Scanning LiDAR

- Hokuyo sensor plugin
- *LaserScan.msg* message type
- Range (4.0m to 30.0m)
- Scan Angle (180 to 360 degrees)
- Position changed in URDF

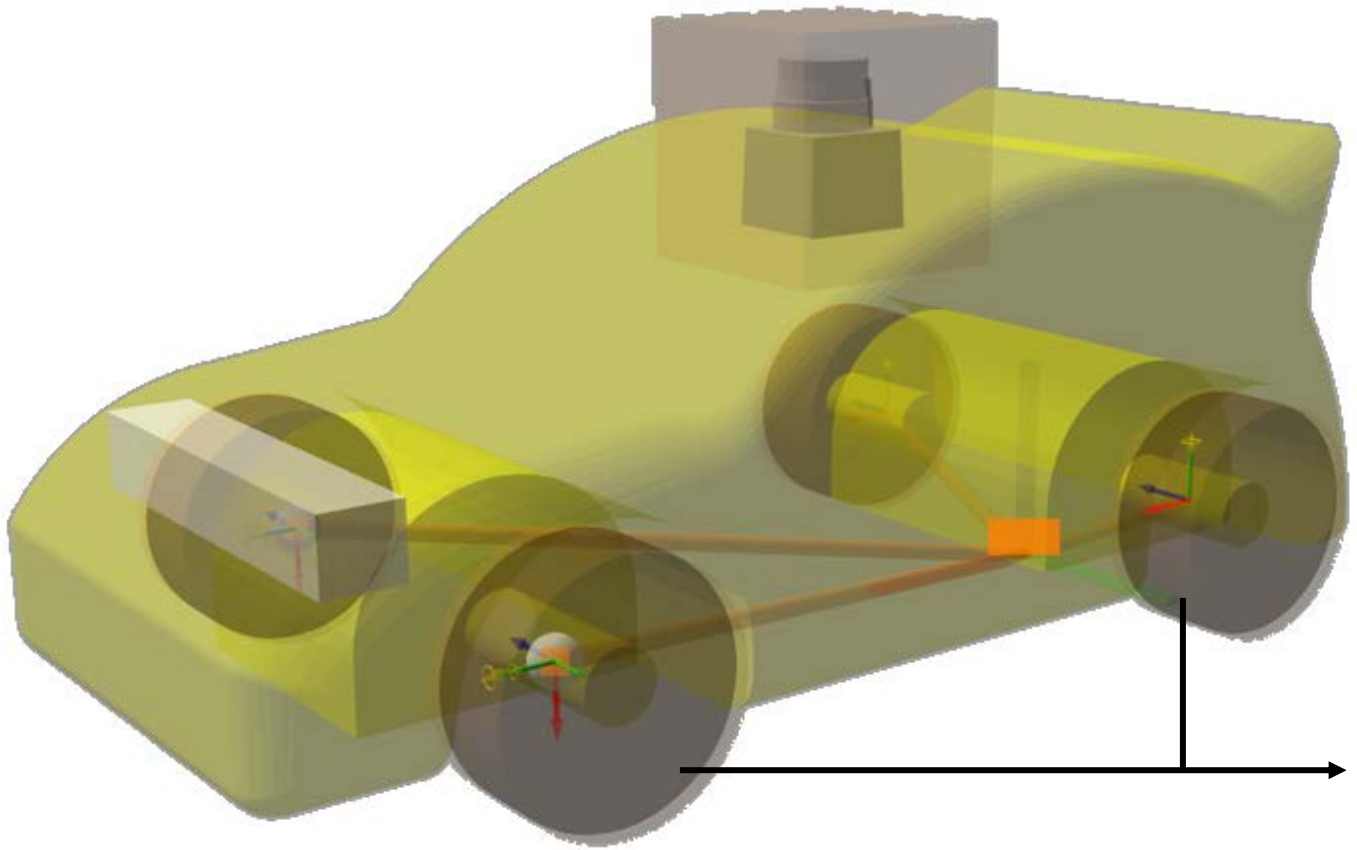
Modular Design

Secondary Navigation Sensors

- Stereo Camera
- Depth Camera
- IMU
- Front/Rear Two Point Radar



Modular Design

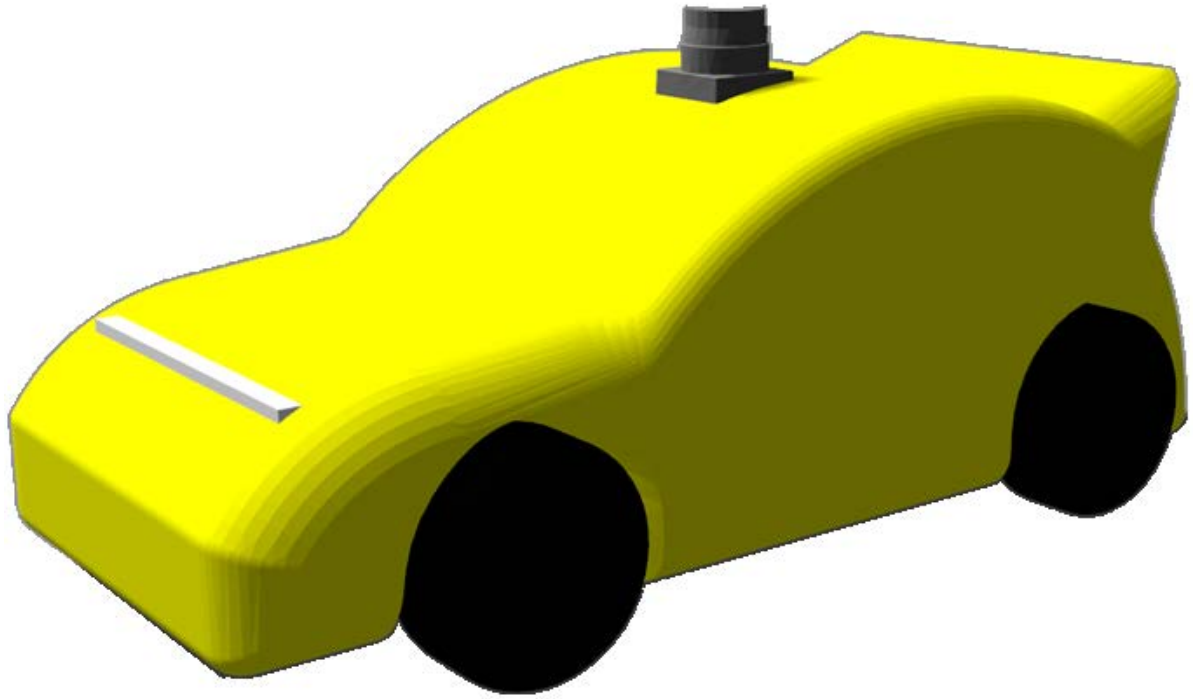


Racecar Chassis
(Shell, powertrain,
Collision sensor, controller)

- Collision sensor
- Rear Wheel/ All wheel drive
- Speed set as RPM or m/s
- Differential enable/disable
- Energy meter and bank with boost

Control Plugin

Command & Control



Message Type:

`AckermannDrive.msg`

ackermann_msgs/AckermannDrive Message

File: `ackermann_msgs/AckermannDrive.msg`

Raw Message Definition

Compact Message Definition

```
float32 steering_angle
float32 steering_angle_velocity
float32 speed
float32 acceleration
float32 jerk
```

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular

```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

autogenerated on Sun, 09 Feb 2020 03:18:27

Executable File | 32 lines (27 sloc) | 891 Bytes

Raw | Blame | History

You're using code navigation to jump to definitions or references. Learn more or give us feedback

/ src / f1tenth / platform / nodes / twist_ackermann.py

```python
#!/usr/bin/env python
'''
2018 Varundev Suresh Babu (University of Virginia)
                MIT License
'''

import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64

angle_pub = rospy.Publisher('/commands/servo/position', Float64, queue_size = 1)
speed_pub = rospy.Publisher('/commands/motor/speed', Float64, queue_size = 1)

VESC_SPEED_REF = 10000.0
VESC_ANGLE_MIN = 0.0
VESC_ANGLE_MAX = 1.0

def twist_ackermann(data):
    angle_ref      = Float64()
    speed_ref      = Float64()
    angle_ref.data = float(data.angular.z/2.0 + 0.5)
    speed_ref.data = float(data.linear.x  * VESC_SPEED_REF)
    angle_pub.publish(angle_ref)
    speed_pub.publish(speed_ref)

if __name__ == '__main__':
    try:
        rospy.init_node('twist_ackermann')
        rospy.Subscriber('cmd_vel', Twist, twist_ackermann)
        rospy.spin()
    except rospy.ROSException:
        pass
```

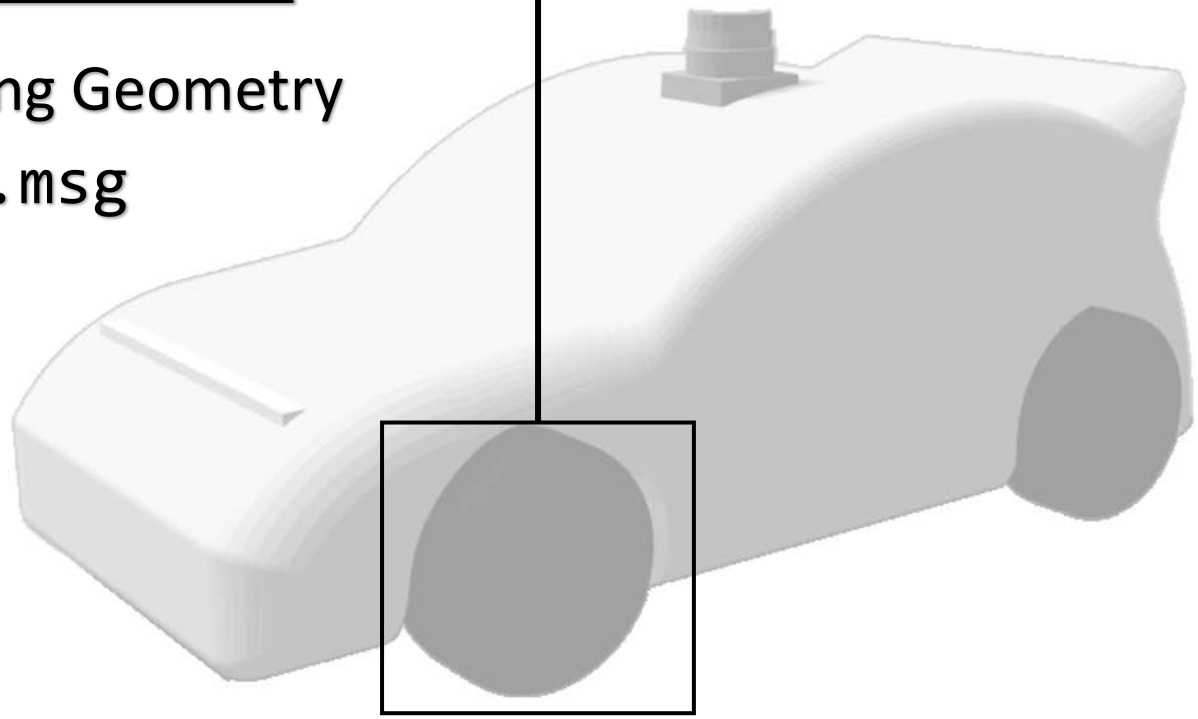
Control Plugin

Command & Control

Steering Column Controller

Ackermann Steering Geometry

`Float64.msg`

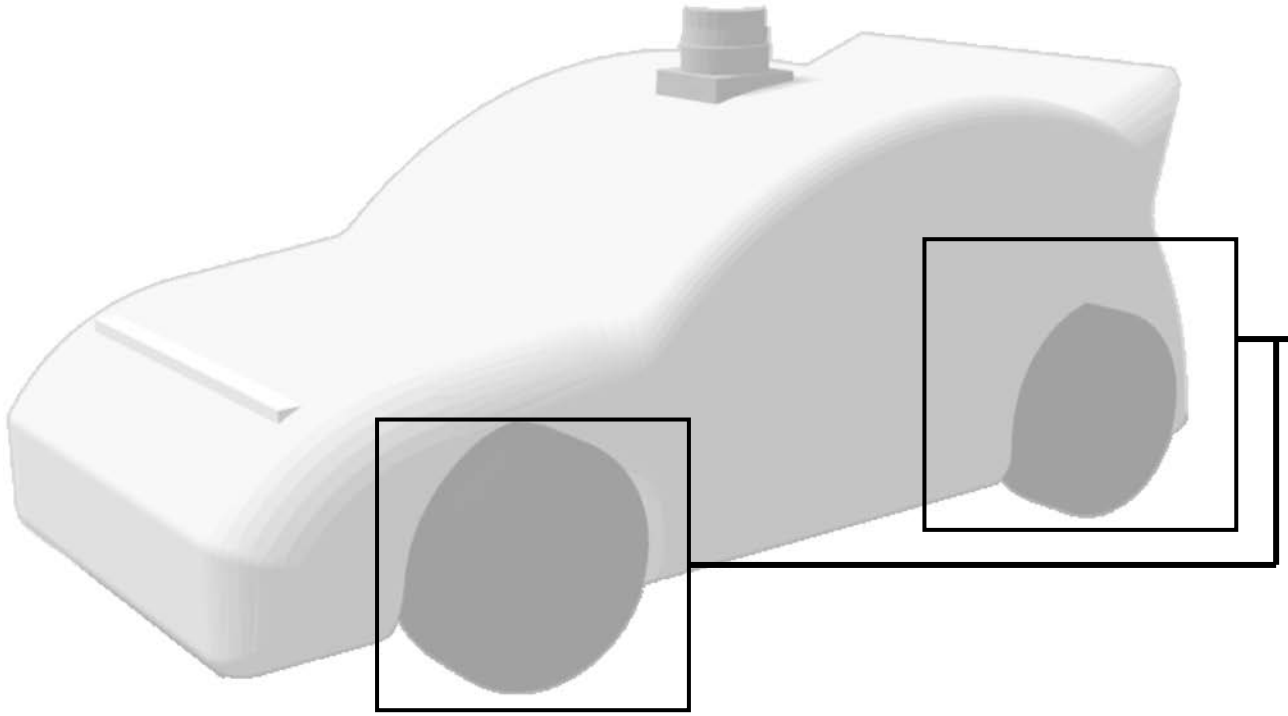


Message Type:

`AckermannDrive.msg`

Control Plugin

Command & Control



Wheel RPM Controller

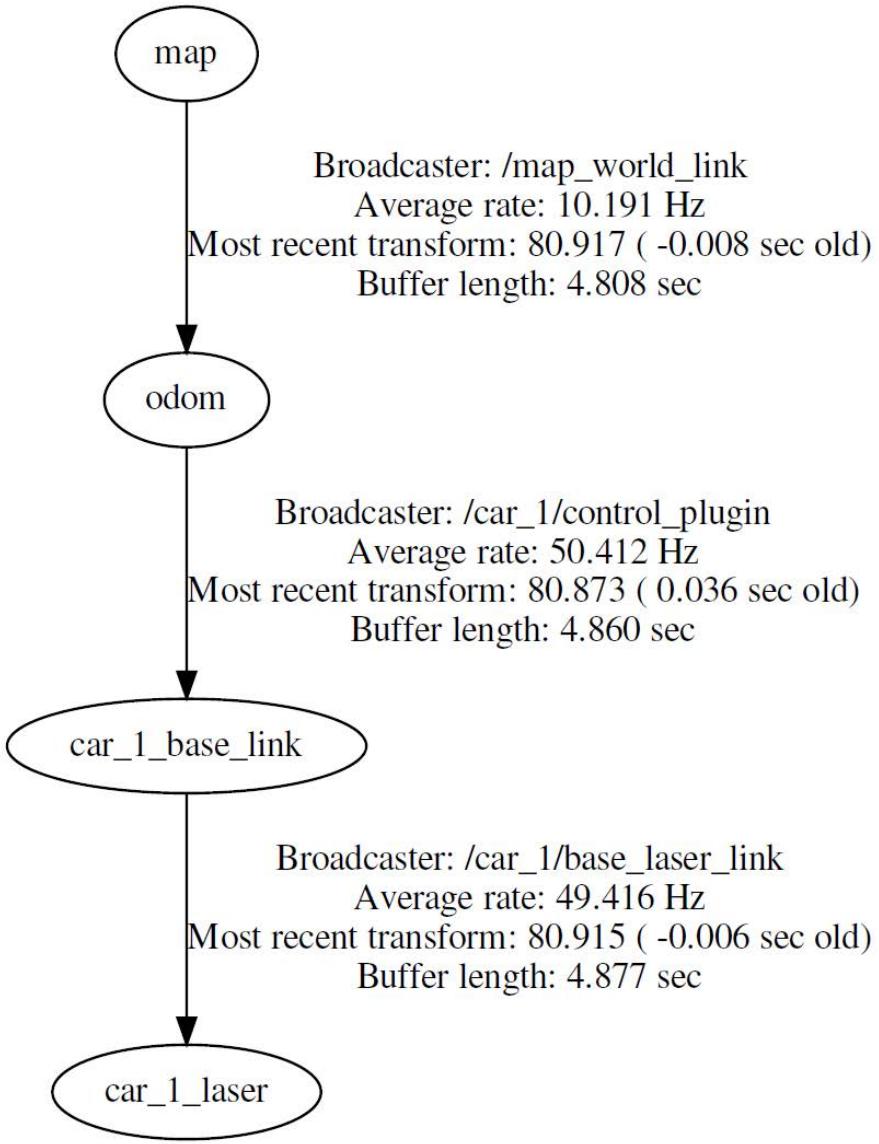
PID Speed Control

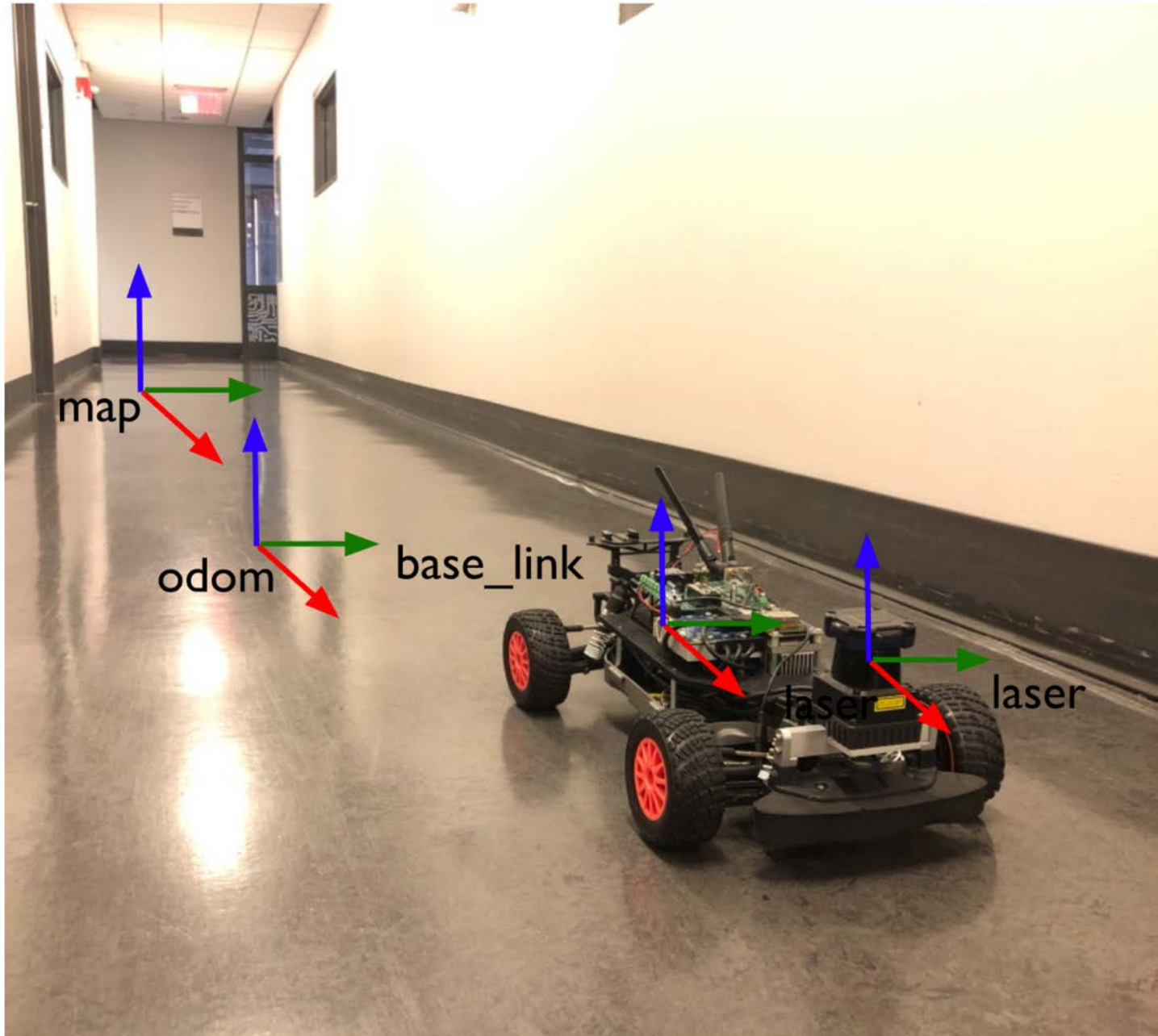
`Float64.msg`

Message Type:

`AckermannDrive.msg`

tf-Tree


map
Broadcaster: /map_world_link
Average rate: 10.191 Hz
Most recent transform: 80.917 (-0.008 sec old)
Buffer length: 4.808 sec
odom
Broadcaster: /car_1/control_plugin
Average rate: 50.412 Hz
Most recent transform: 80.873 (0.036 sec old)
Buffer length: 4.860 sec
car_1_base_link
Broadcaster: /car_1/base_laser_link
Average rate: 49.416 Hz
Most recent transform: 80.915 (-0.006 sec old)
Buffer length: 4.877 sec
car_1_laser


map
odom
base_link
laser
laser

Racecar Object Creation

```
<!-- spawn car using the set global parameters -->
<include      file          = '$(find f1tenth-sim)/launch/simulator.launch'>
<arg          name          = 'car_name'
              value         = '$(arg car_name)'/>
<arg          name          = 'paint'
              value         = '$(arg car_paint)'/>
<arg          name          = 'run_gazebo'
              value         = '$(arg run_gazebo)'/>
<arg          name          = 'x_pos'
              value         = '$(arg x_pos)'/>
<arg          name          = 'y_pos'
              value         = '$(arg y_pos)'/> </include>
```

Racecar Object Creation

```
<!-- spawn car using the set global parameters -->
<include      file          = '$(find f1tenth-sim)/launch/simulator.launch'>
<arg          name          = 'car_name'
              value         = '$(arg car_name)'/>
<arg          name          = 'paint'
              value         = '$(arg car_paint)'/>
<arg          name          = 'run_gazebo'
              value         = '$(arg run_gazebo)'/>
<arg          name          = 'x_pos'
              value         = '$(arg x_pos)'/>
<arg          name          = 'y_pos'
              value         = '$(arg y_pos)'/> </include>
```

Unique namespace

Racecar Object Creation

```
<!-- spawn car using the set global parameters -->
<include      file          = '$(find f1tenth-sim)/launch/simulator.launch'>
<arg          name          = 'car_name'
              value         = '$(arg car_name)'/>
<arg          name          = 'paint'
              value         = '$(arg car_paint)'/>
<arg          name          = 'run_gazebo'
              value         = '$(arg run_gazebo)'/>
<arg          name          = 'x_pos'
              value         = '$(arg x_pos)'/>
<arg          name          = 'y_pos'
              value         = '$(arg y_pos)'/> </include>
```

Visual paint scheme

Racecar Object Creation

```
<!-- spawn car using the set global parameters -->
<include      file          = '$(find f1tenth-sim)/launch/simulator.launch'>
<arg          name          = 'car_name'
              value         = '$(arg car_name)'/>
<arg          name          = 'paint'
              value         = '$(arg car_paint)'/>
<arg          name          = 'run_gazebo'
              value         = '$(arg run_gazebo)'/>
<arg          name          = 'x_pos'
              value         = '$(arg x_pos)'/>
<arg          name          = 'y_pos'
              value         = '$(arg y_pos)'/> </include>
```

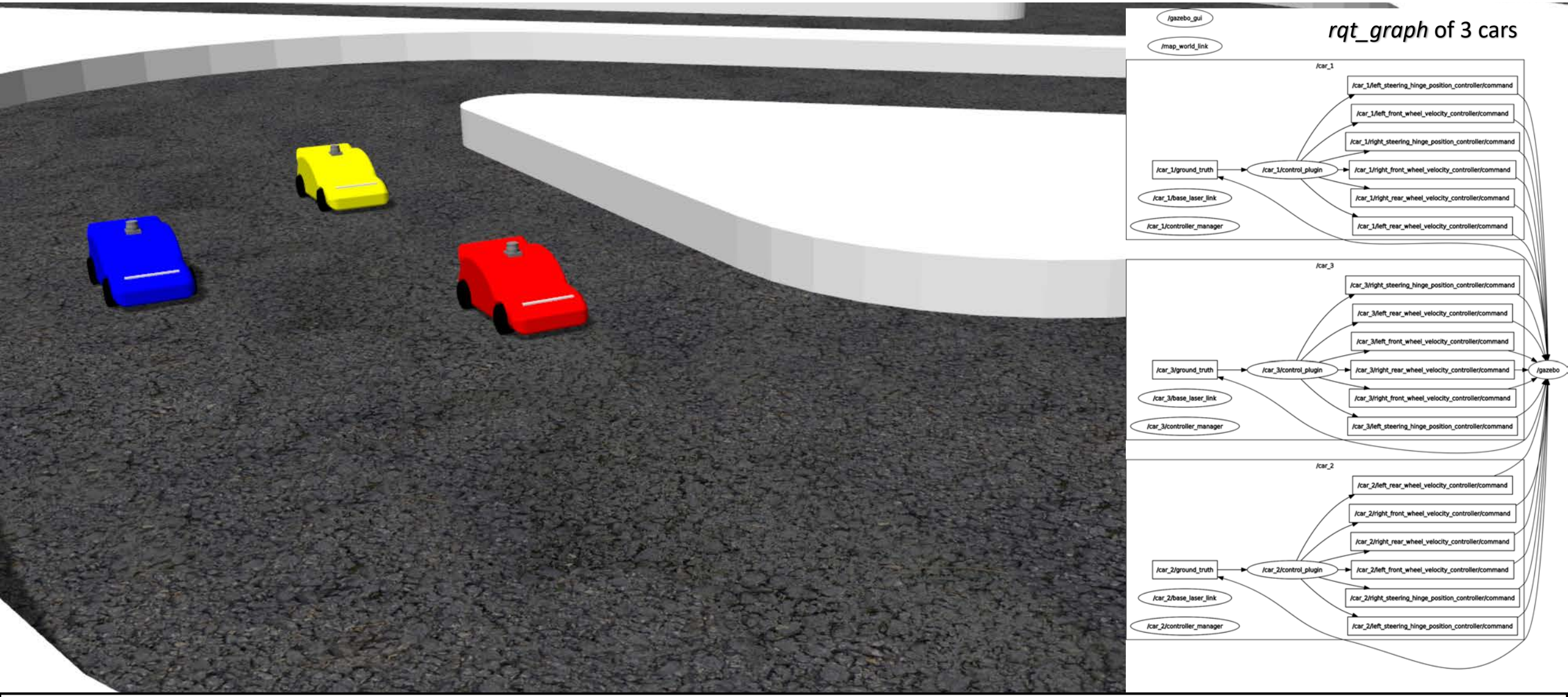
Circuit-Breaker (keeps only one session active)

Racecar Object Creation

```
<!-- spawn car using the set global parameters -->
<include      file          = '$(find f1tenth-sim)/launch/simulator.launch'>
<arg          name          = 'car_name'
              value         = '$(arg car_name)'/>
<arg          name          = 'paint'
              value         = '$(arg car_paint)'/>
<arg          name          = 'run_gazebo'
              value         = '$(arg run_gazebo)'/>
<arg          name          = 'x_pos'
              value         = '$(arg x_pos)'/>
<arg          name          = 'y_pos'
              value         = '$(arg y_pos)'/> </include>
```

Racecar spawn location

Spawn Multiple Racecars

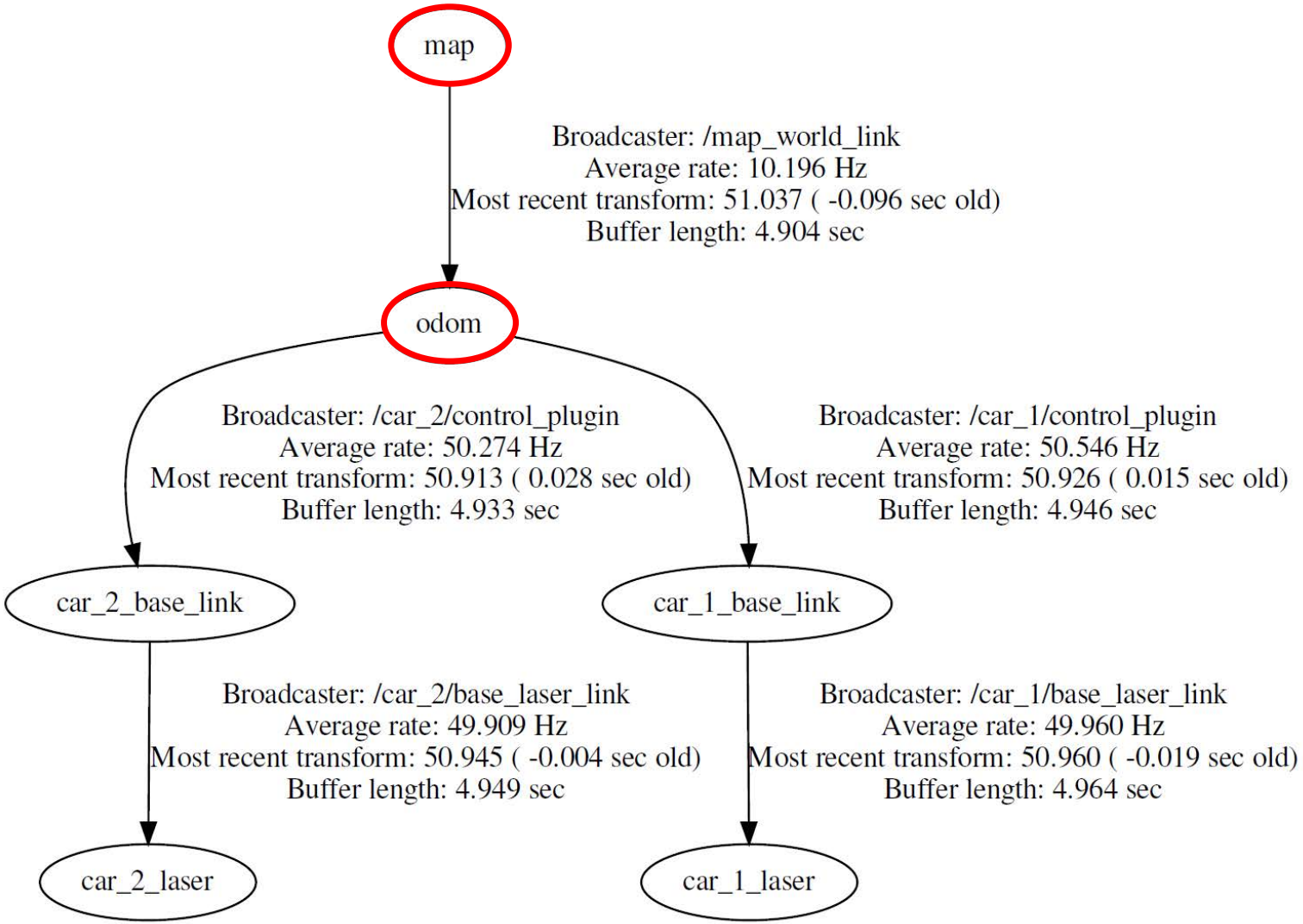


rqt_graph of 3 cars

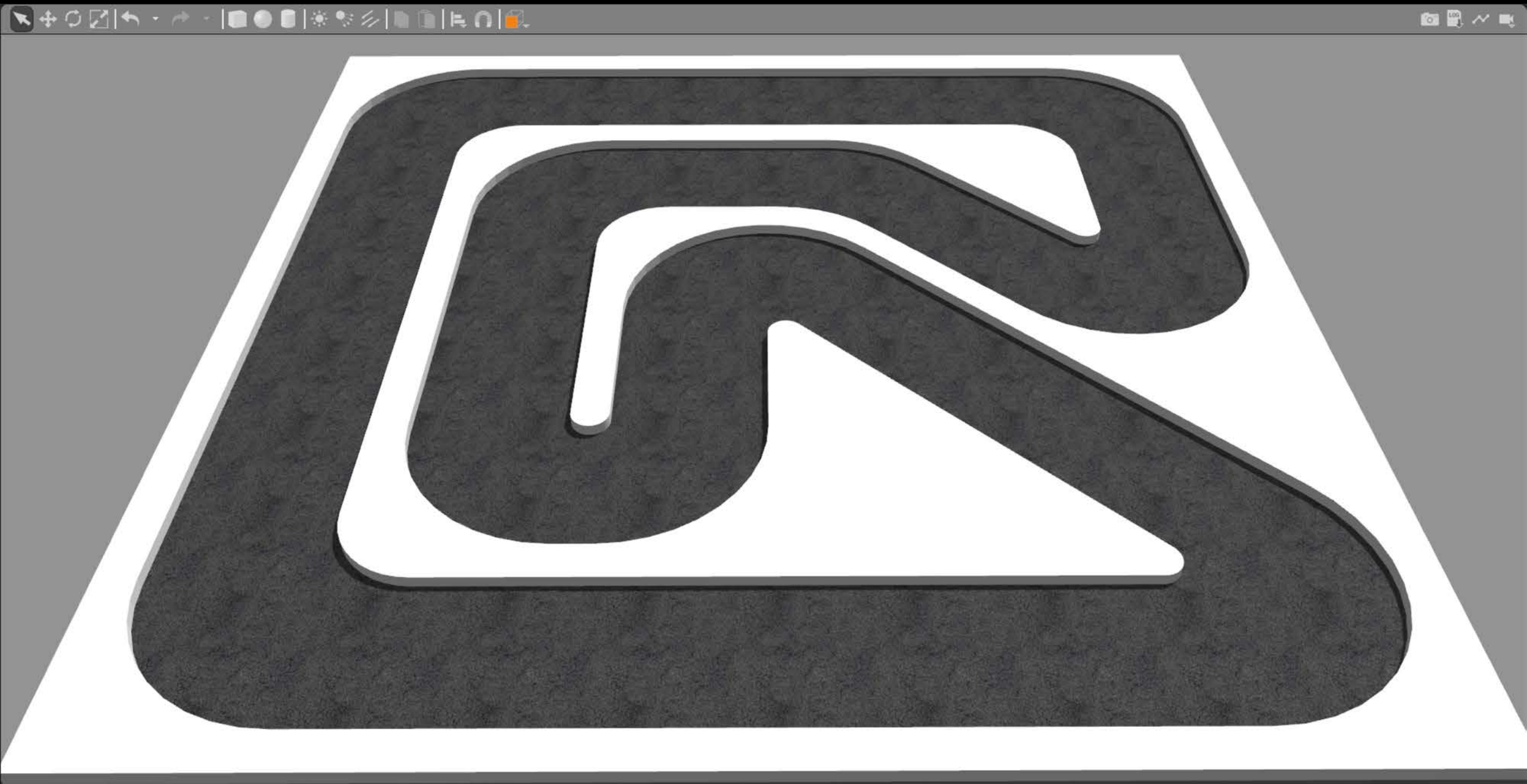
Namespace resolution handled by ROS

Capable of implementing different controllers

Multiple Racecars (tf-frames)



Simulated Race Track



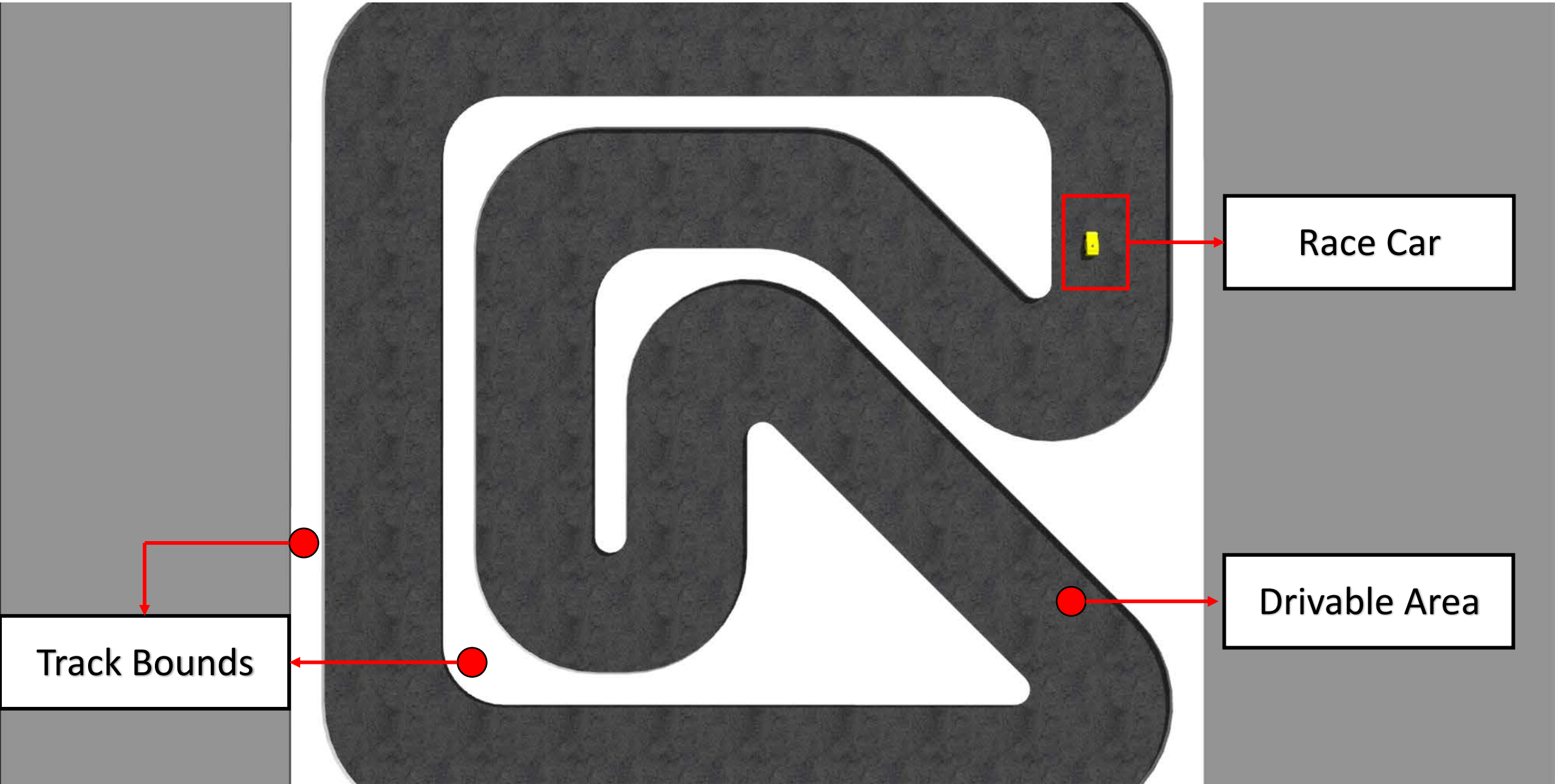
Race Track creation process

Sketch a 2D track

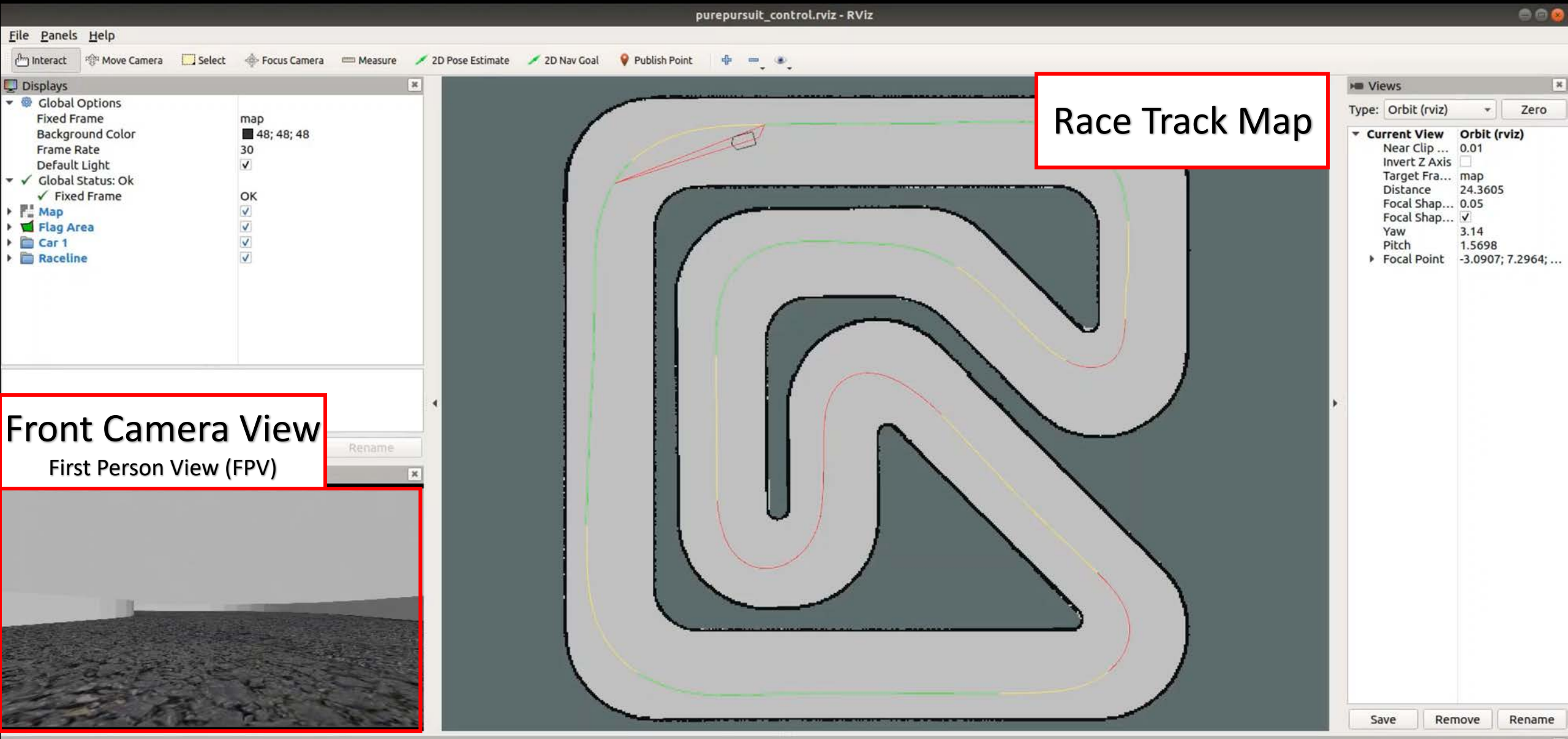
Convert to 3D mesh

Export as STL/DAE

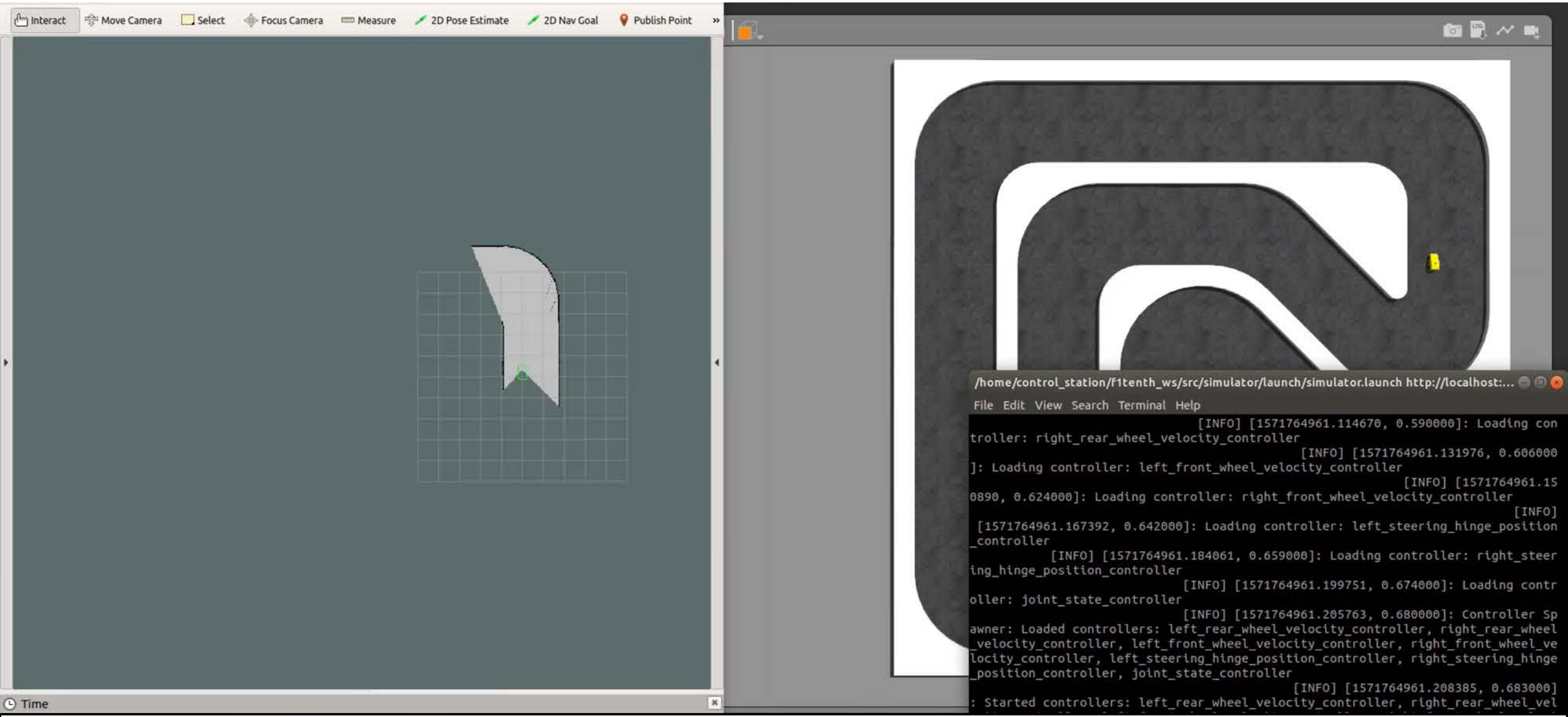
Race-Track Setup (Gazebo View)



Race-Track Setup (*rviz* View)

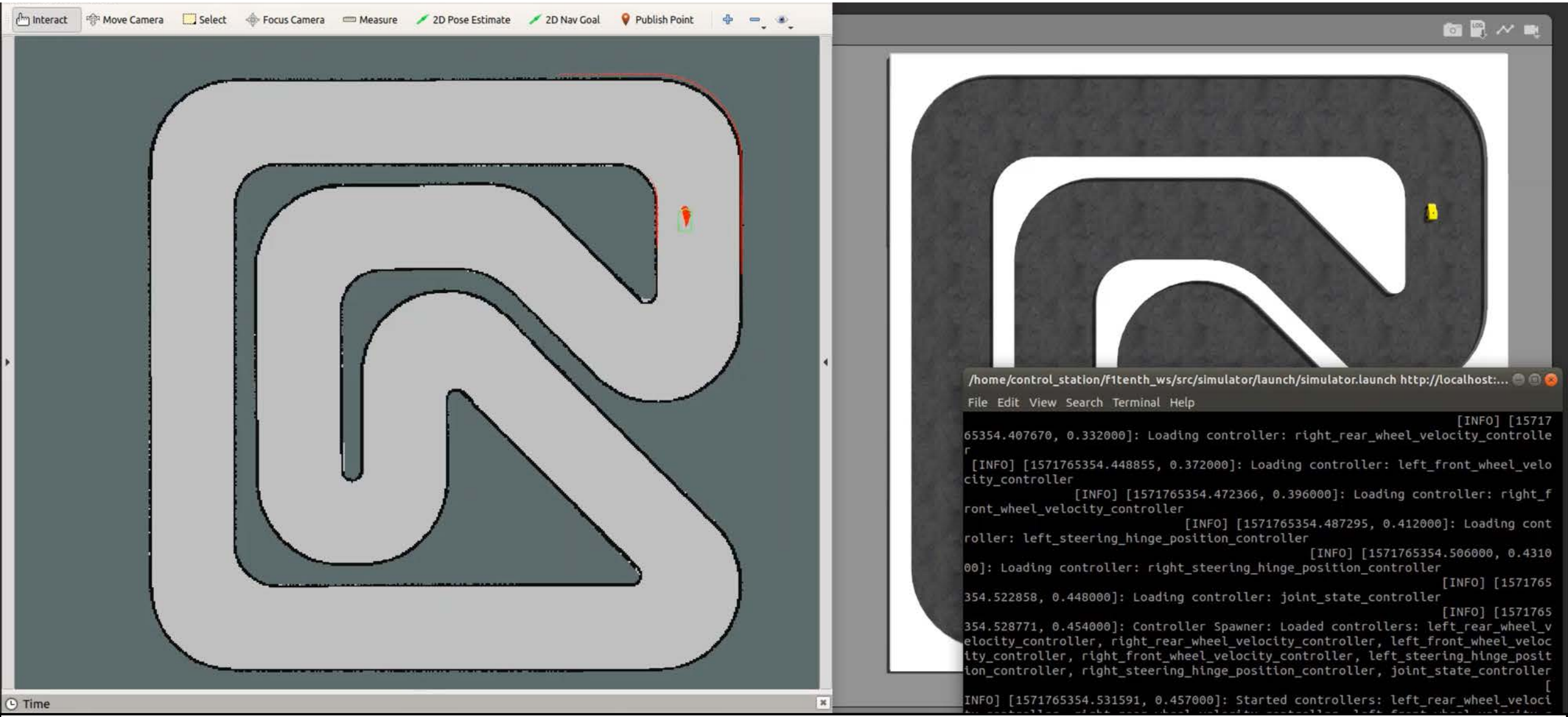


Mapping – Hector SLAM



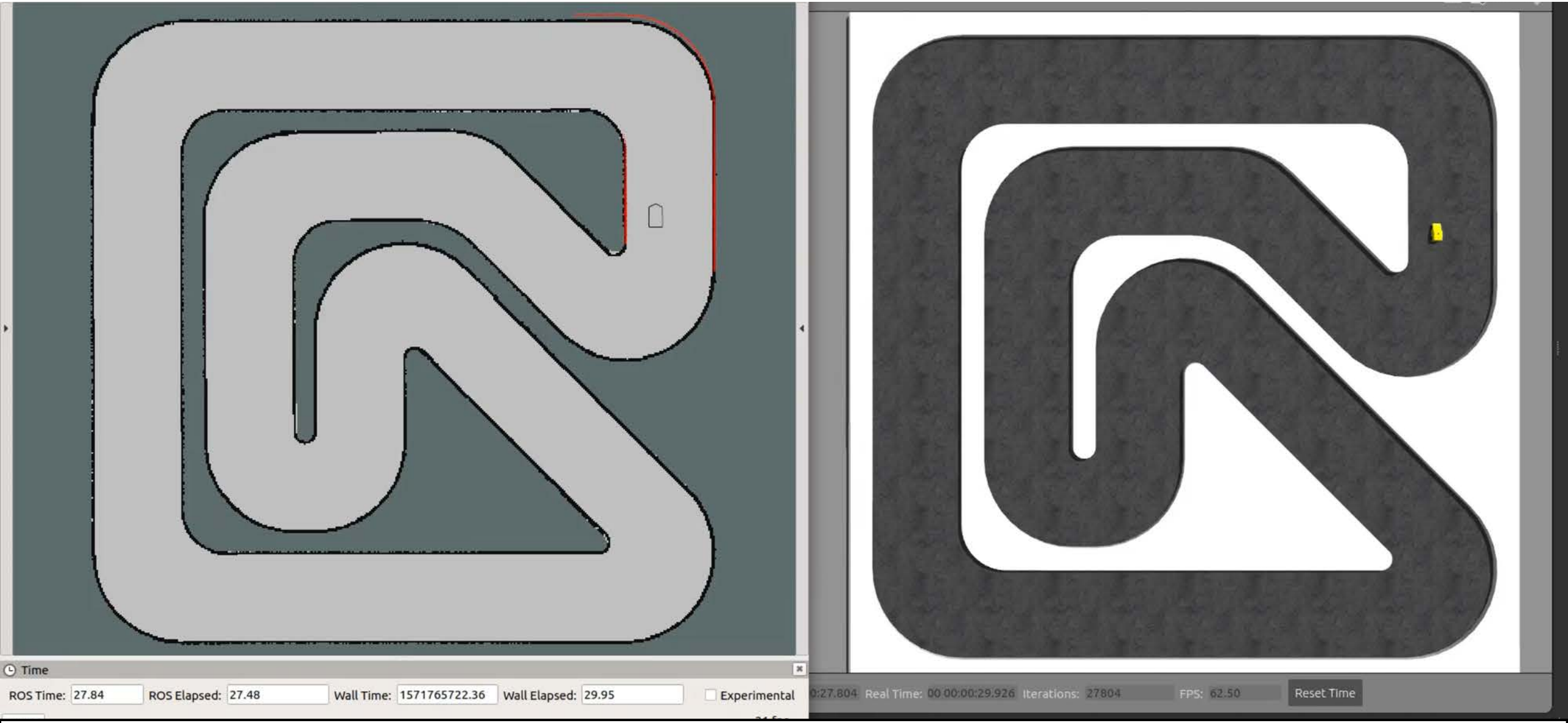

```
roslaunch f1tenth-sim simulator.launch
roslaunch f1tenth-sim mapping.launch
```

Localization – GPU Particle Filter



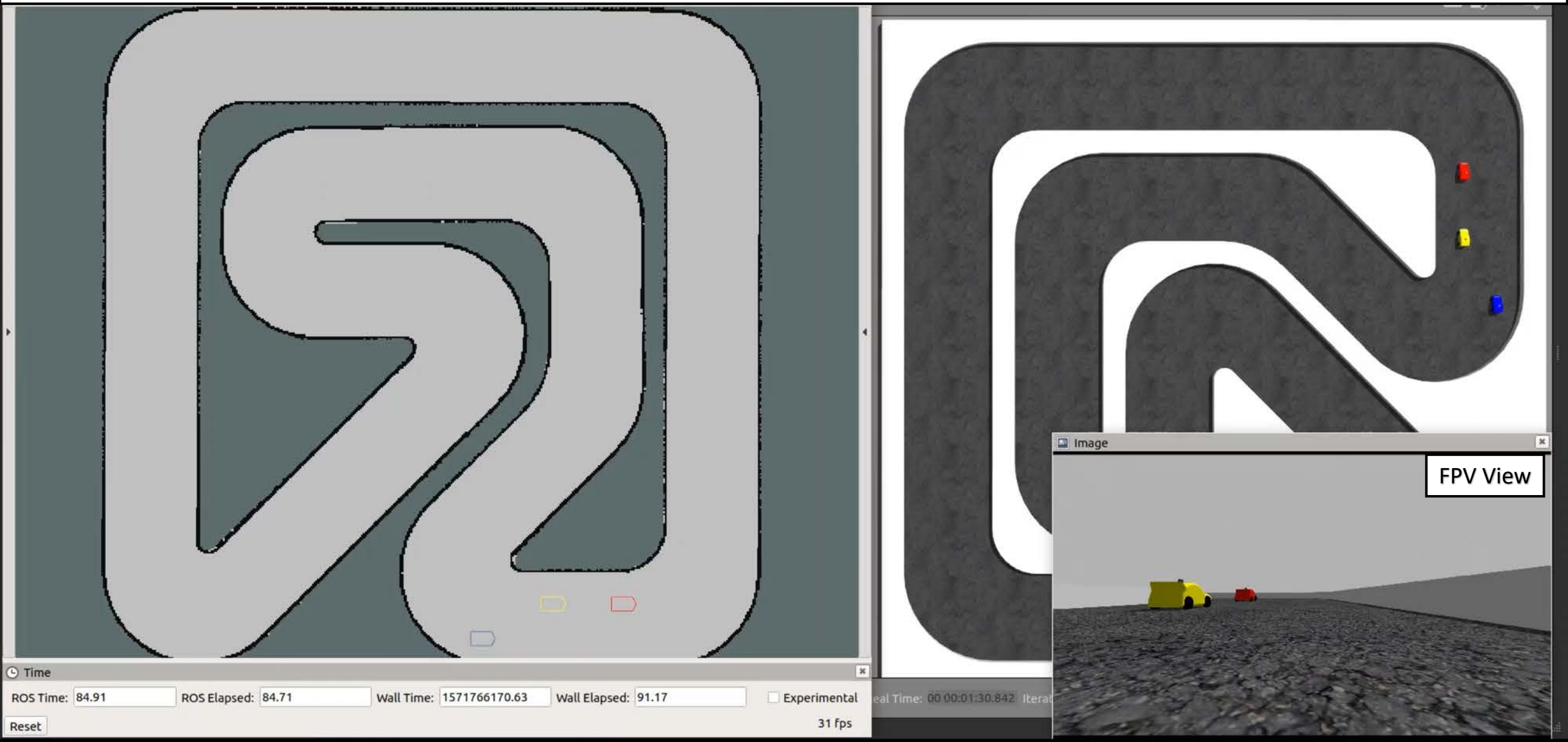

```
roslaunch f1tenth-sim simulator.launch
roslaunch f1tenth-sim localization.launch
```

Motion Planning – ROS Navigation with TEB Planner

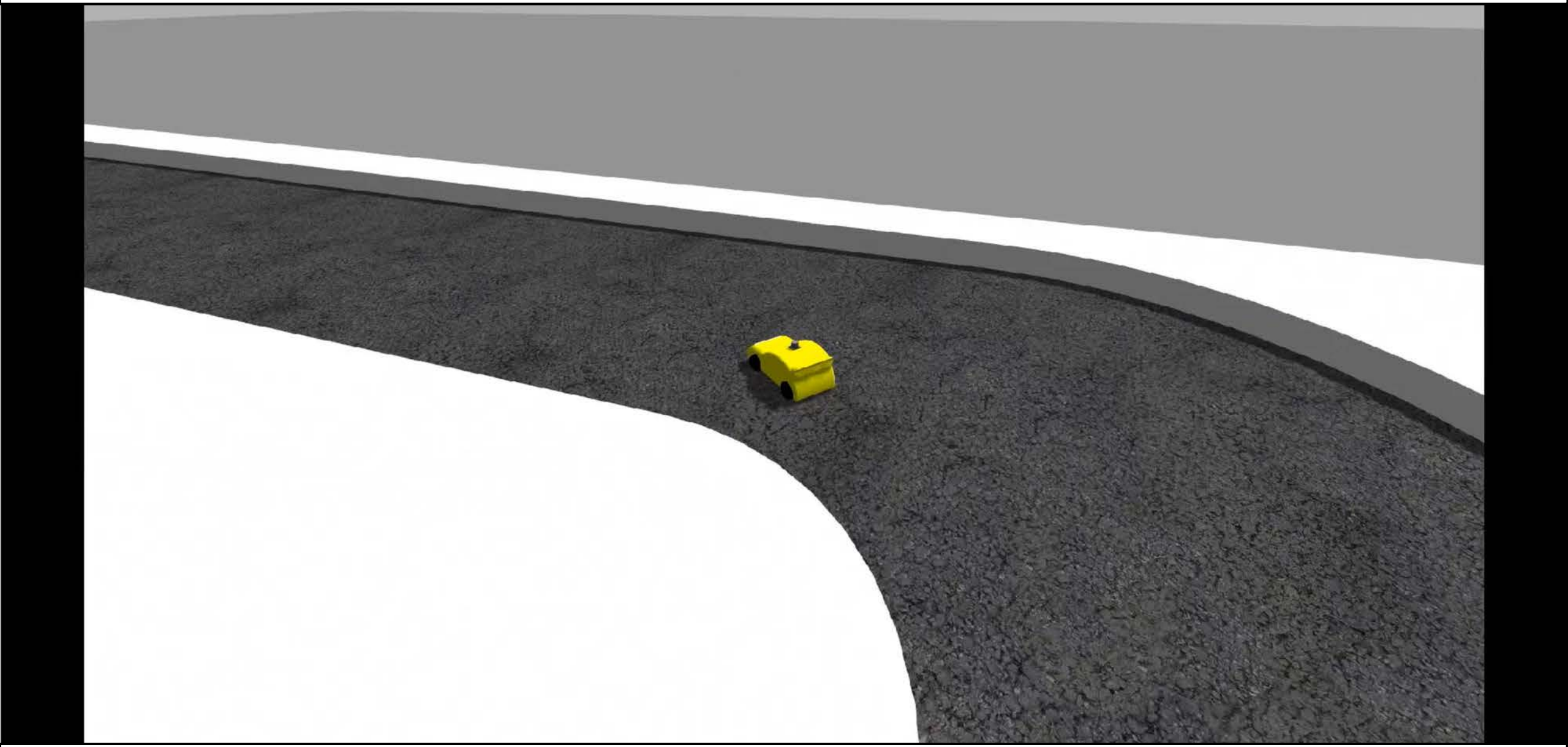



```
roslaunch f1tenth-sim navigation.launch
```

Multiple Independent Navigation

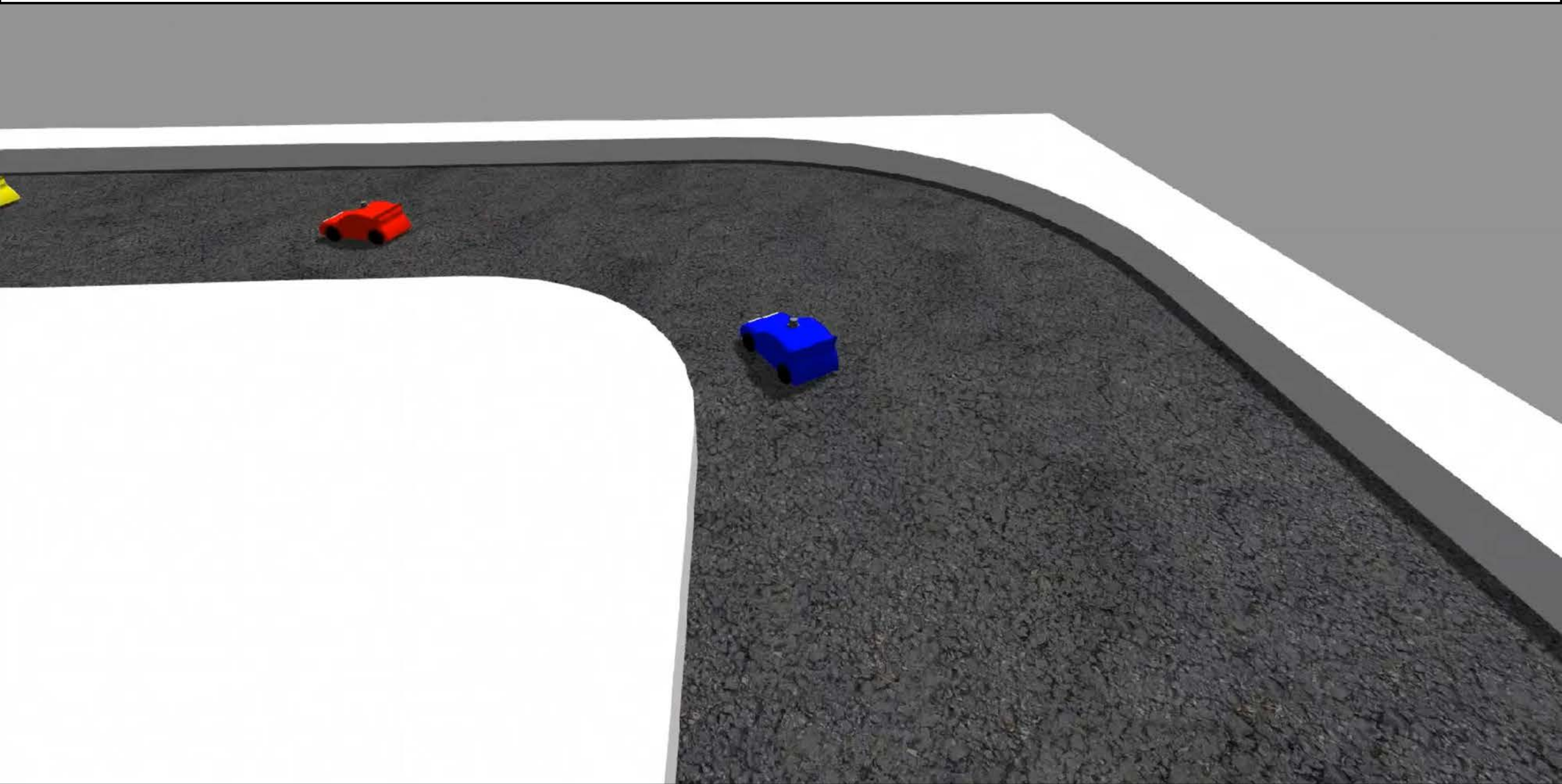


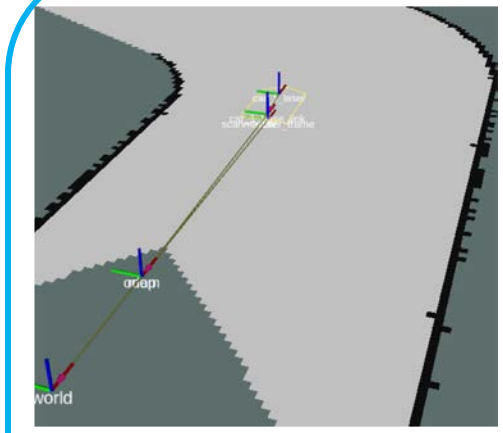
Racing Algorithms (Single Racecar)



`roslaunch f1tenth-sim purepursuit_one_car.launch`

Racing Algorithms (Multiple Racecars)





Algorithms

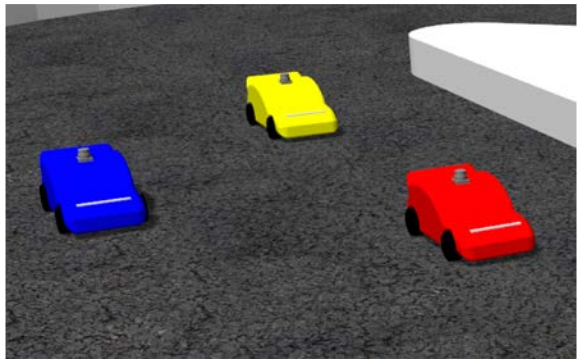
Mapping
Hector, Cartographer

Localization
AMCL, GPU Particle Filter

Motion Planning
ROS Navigation
TEB Planner

Sensors
Hokuyo 2D LiDAR
RGBD Camera
Collision Sensor

Control
Ackermann Steering
Traction Control
Realistic dynamics



Modularity

f1tenth.dev

Open Source

GitHub docker aws *

* Planned

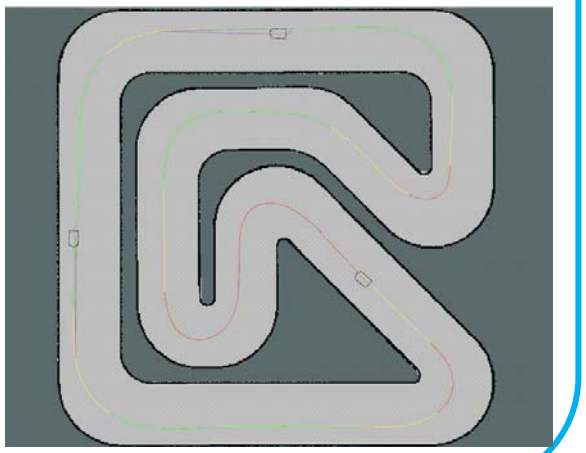
Autonomous Racing Research

Advanced Pure-Pursuit
With Adaptive Lookahead

Autonomous Overtake

Head-to-Head Racing

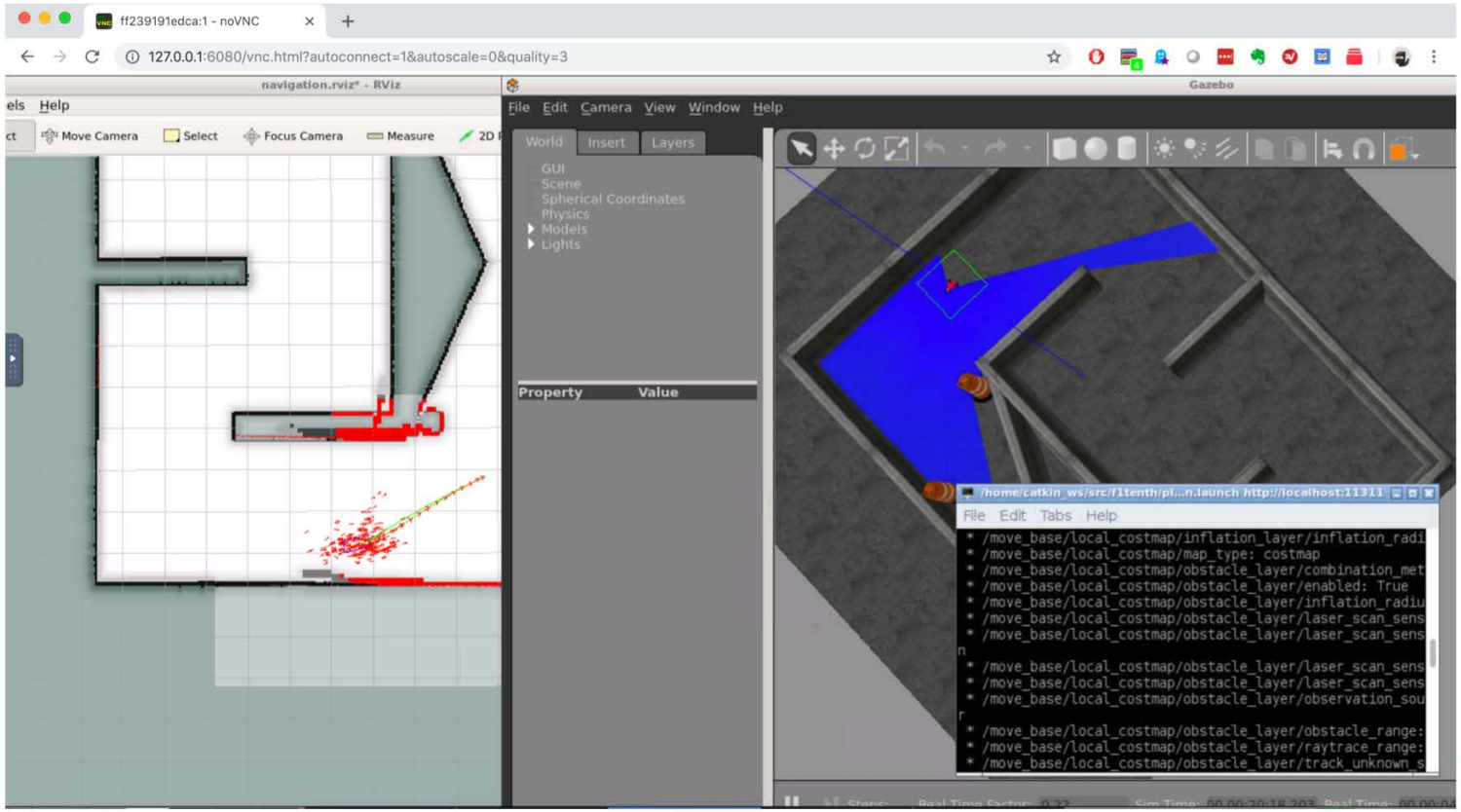
Dynamic Obstacle Avoidance



F1/10 Autonomous Racing Simulator is now open-source and available!

Step 1) `docker run -it --rm -p 6080:80 madhurbehl/f1tenth`

Step 2) Visit 127.0.0.1:6080 in your favorite browser



Algorithms Supported:

Follow-the-gap planner
Hector-SLAM mapping
Adaptive Monte Carlo Localization
Time-Elastic Band Local Planner

Visit madhurbehl/f1tenth
on DockerHub or click on the icons below

docker GitHub