PROJECT REPORT : RACING OF AUTONOMOUS CARS IN ROS

Chupin Clément
University of Clermont-Ferrand

Guigues Enzo
SIGMA of Clermont-Ferrand

January 26, 2023

ABSTRACT

An autonomous car is a vehicle that is capable of navigating and driving itself without human input. Autonomous cars are becoming increasingly prevalent in various industries, including transportation, logistics, and defense. However, developing and testing autonomous cars can be complex and costly, as it requires real-world testing and validation. To overcome these challenges, many researchers and developers use simulation environments to develop and test autonomous cars. Simulation environments offer a safe and cost-effective way to test autonomous car systems, and they can be used to evaluate different algorithms and design choices in a controlled and repeatable manner. In this project, we present an overview of the challenges and benefits of using simulation environments for autonomous car development, and we discuss the various factors that need to be considered when designing and implementing an autonomous car in a simulation environment.

1 Introduction :

1.1 Problem :

There are several challenges that need to be addressed when designing and implementing a control system for an autonomous car in a simulation environment:

Realism: The control system must be able to accurately simulate the physics and behavior of a real car, including factors such as acceleration, deceleration, turning, and braking.

Decision-making: The control system must be able to make intelligent decisions about how to navigate the environment and respond to obstacles and other challenges. This may involve implementing algorithms for path planning, object avoidance, and traffic rules compliance.

Sensor Fusion: The control system may need to integrate and process data from multiple sensors, such as cameras, lidar, radar, and GPS, to create a comprehensive view of the environment.

Robustness: The control system must be able to handle unexpected or adverse conditions, such as sensor failures or unexpected obstacles, without crashing or behaving erratically.

Efficiency: The control system should be designed to be as efficient as possible, to minimize the computational resources required to run the simulation.

Overall, designing and implementing a control system for an autonomous car in a simulation environment requires a combination of skills in physics simulation, artificial intelligence, and sensor processing, and it requires careful consideration of factors such as realism, decision-making, sensor fusion, robustness, and efficiency.

1.2 Robotic's solutions :

1.2.1 Range sensor

Range sensors are sensors that are used to measure the distance to an object or surface. In the context of an autonomous car, range sensors can be used to help control the car by providing information about the environment around the car. There are several ways that range sensors can be used to control an autonomous car:

Obstacle avoidance: Range sensors such as lidar or ultrasonic sensors can be used to detect obstacles in the car's path and trigger evasive actions to avoid them.

Navigation: Range sensors can be used to create a map of the environment, which can be used by the control system to plan routes and navigate the car to its destination.

Localization: Range sensors can be used to determine the car's position and orientation relative to its surroundings, which can be used to improve the accuracy of the car's localization and navigation.

Overall, range sensors are an important component of an autonomous car's control system, as they provide information about the car's environment that can be used to make intelligent and safe driving decisions.

1.2.2 Camera sensor

Cameras are sensors that capture images of the environment and can be used to help control an autonomous car. There are several ways that camera sensors can be used to control an autonomous car:

Obstacle avoidance: Cameras can be used to detect obstacles in the car's path and trigger evasive actions to avoid them.

Navigation: Cameras can be used to create a map of the environment, which can be used by the control system to plan routes and navigate the car to its destination.

Localization: Cameras can be used to determine the car's position and orientation relative to its surroundings, which can be used to improve the accuracy of the car's localization and navigation.

Overall, camera sensors are an important component of an autonomous car's control system, as they provide a wealth of information about the car's environment that can be used to make intelligent and safe driving decisions.

1.2.3 PID control

PID control (proportional-integral-derivative control) is a control algorithm that is commonly used in control systems to maintain a desired output or setpoint. In the context of an autonomous car, PID control can be used to help control the car's speed, position, or other variables to maintain a desired setpoint.

Here are some examples of how PID control can be used to help control an autonomous car:

Speed control: PID control can be used to maintain a desired speed setpoint by adjusting the car's acceleration or braking. The control system can compare the car's current speed to the setpoint, and adjust the acceleration or braking based on the difference between the two.

Position control: PID control can be used to maintain a desired position setpoint by adjusting the car's steering or acceleration. The control system can compare the car's current position to the setpoint, and adjust the steering or acceleration based on the difference between the two.

Overall, PID control is a useful tool for controlling an autonomous car, as it allows the control system to maintain a desired setpoint for variables such as speed, position, or trajectory, and it can be used to adjust the car's acceleration, braking, and steering to achieve the desired setpoint.

1.3 ROS :

ROS (Robot Operating System) is a collection of software libraries and tools that assist developers in creating robot applications. It simplifies the process by providing a variety of functionalities such as device driver support, message-passing communication, visualizers, libraries, and more. ROS can be used on a large variety of robots, and is often employed in research, education and industry. It supports multiple programming languages like C++ and Python. [1]

We have used ROS for the realization of this project.

It will allow us to manage the different elements of our simulation:

- The control of the vehicle through a joystick
- The vehicle control by a control law (PID) based on the lidar measurement.
- The detection of the winner through a color measurement of the wheels by camera.
- Launching of the race with a countdown
- A display to inform the user about the countdown and the winner.

2 Method :

2.1 World :

2.1.1 Blender :

Blender is a free and open-source 3D creation software that can be used for a variety of purposes, such as creating animations, models, art and video games. It offers a comprehensive set of tools, including sculpting, texturing, motion tracking and a built-in game engine. In addition, Blender has a built-in video editor and allows you to use Python scripts to create custom tools and add-on. It is known for its wide range of uses in film, television, and video games, as well as for personal and amateur projects. [2]

Blender has a wide range of applications:

- To create detailed 3D models of objects, characters and environments that can be used in animation, video games, architectural visualization and product design.

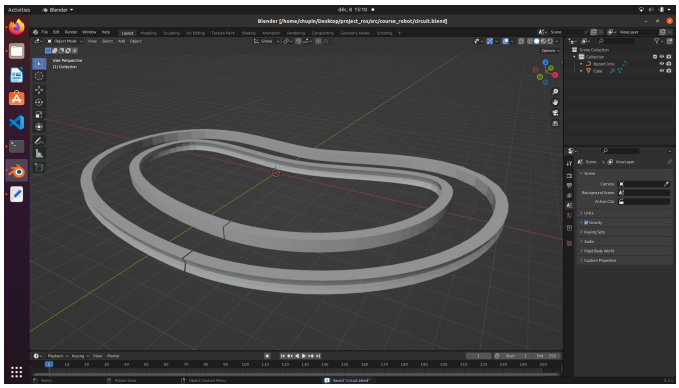
- To create images and videos by rendering 3D models and animations, using a variety of rendering engines to create realistic or stylized images and animations.

- To create and edit animations for characters, objects that can be used in movies, video games with an integrated animation system.

- To create realistic animations of water and different materials with physical simulation tools.

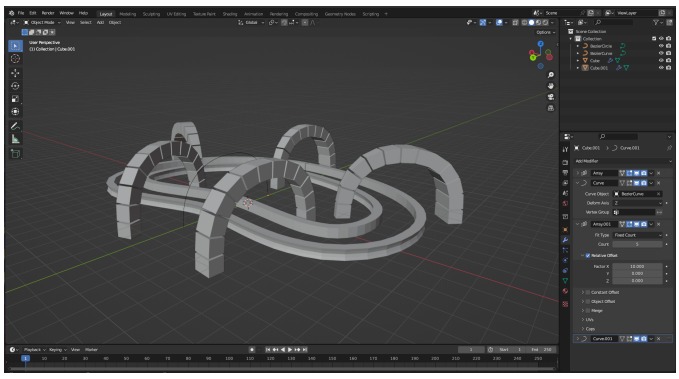
We chose to model our circuit on Blender because it has all the functionality we needed. The models made in Blender also have the advantage of being used directly in Gazebo.

2.1.2 Race track :



Track collision

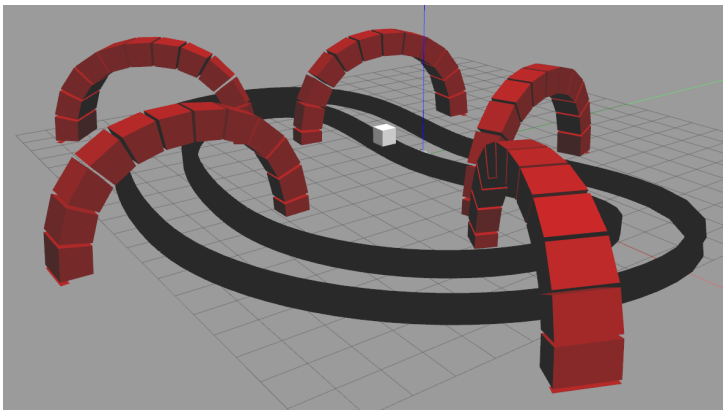
The first step is to model the layout of the circuit. For this, we made two walls on both sides of the track to delimit it. The inner side of the wall is flat to facilitate the detection of distances by lidars that could be influenced by a curved wall.
Thus, we only add a curvature on the outside for purely aesthetic purposes.



Track visual

We then add 5 arches along the circuit again simply for aesthetics. One of them will be used to represent the start and finish point of the race where the countdown and the winner will be displayed.

And this is the final on gazebo :



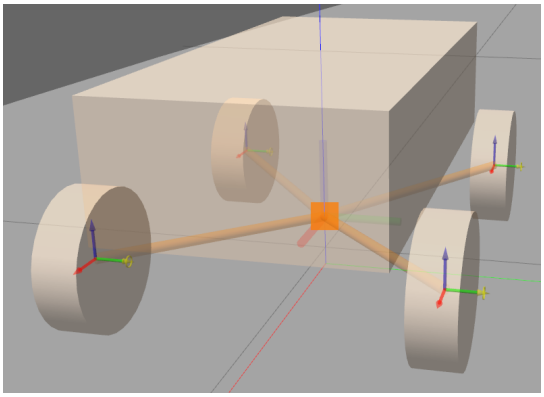
Modeling of the race track via blender

Finally, we import all the elements modeled on Gazebo. We also add color to better differentiate the circuit and the arches.

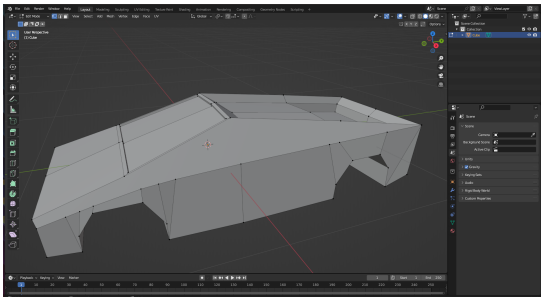
2.1.3 Autonomous car :

Now that we have modeled the whole circuit and the environment, we have to model the vehicles that will drive on it.

In order to optimize the performance of the program and to reduce the risks of bugs, we have chosen to limit ourselves to basic shapes when building the collision zone of the cars. [3]



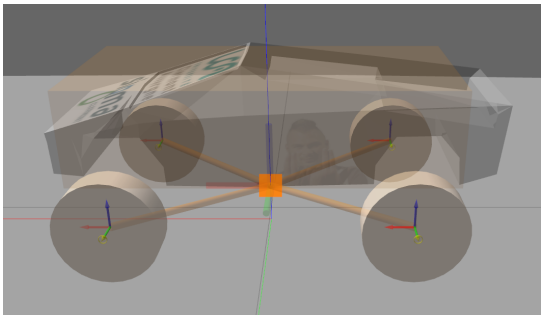
Car collision



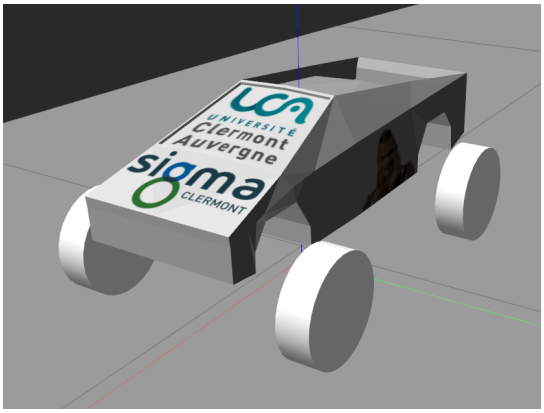
Car visual

On the left, we have our collision zone which is defined from cylinders for the wheels and a simple cuboid modeling both the chassis and the body of the vehicle.

The image on the right shows us the visible model which is closer to the typical shape of a real car. We can easily identify the different parts of the vehicle. This will allow the user to better project himself for the application in the real world.



Car visual



Car collision

We then import everything on Gazebo to make sure that everything works as planned:

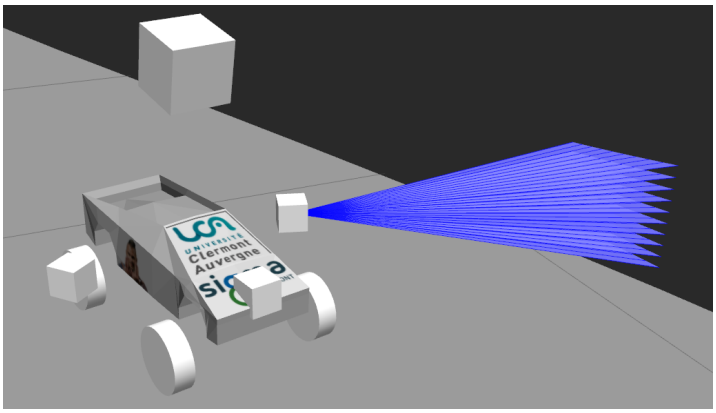
We have on the left a superimposed view that allows to see both the collision area and the visual of the vehicle. We can see that there is not a big difference between the two and that the choice of simple shapes will not have a big influence compared to the real model

The final design of the vehicle is visible on the right with a customized version by applying an image on the body. It will then be possible to modify the color of the wheels in order to differentiate the autonomous vehicle from the user controlled vehicle.

2.2 Autonomous car :

2.2.1 Sensor :

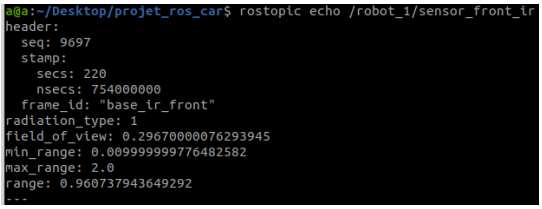
To be able to interact with our environment, we have to be able to understand the world with sensor. We will use RangeSensor and CameraSensor.



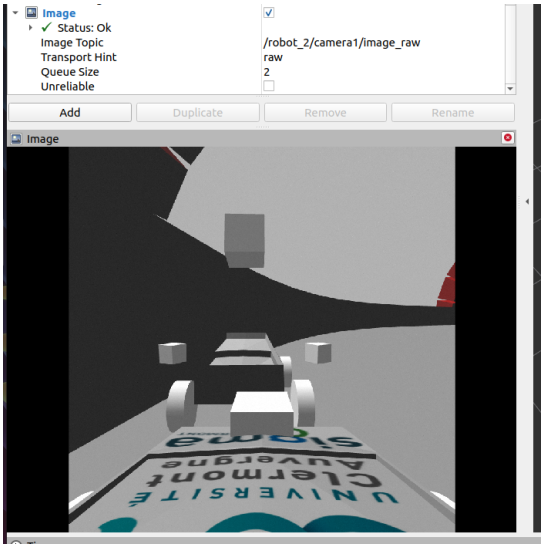
Sensor of the car

We then use 2 distance sensors on each side of the vehicle which will allow to measure on each side the distance to the edge of the circuit. It is also possible to change their orientation to make sure to take the right measurements. We also add one at the front to prevent some collisions (not used in our program).

A camera completes the lidars by allowing the user to drive the vehicle in a realistic way in the first person and to observe what the autonomous car sees.



Range plot by terminal



Camera pov by rviz

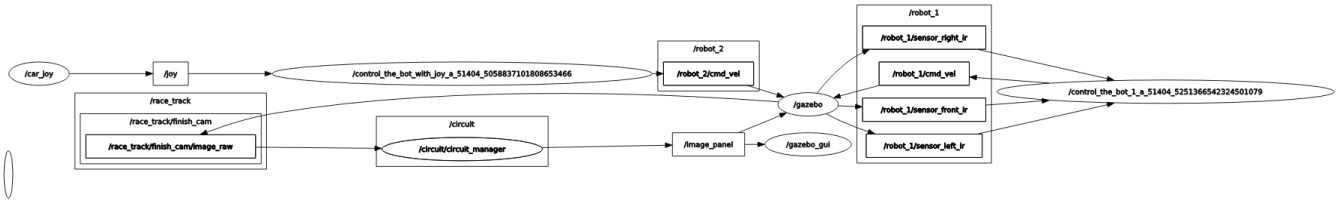
Here for example, you have on the left the information returned by the front lidar. And the camera return on the right.

2.3 Control node

To control different car, we design control rules based on different sensor.

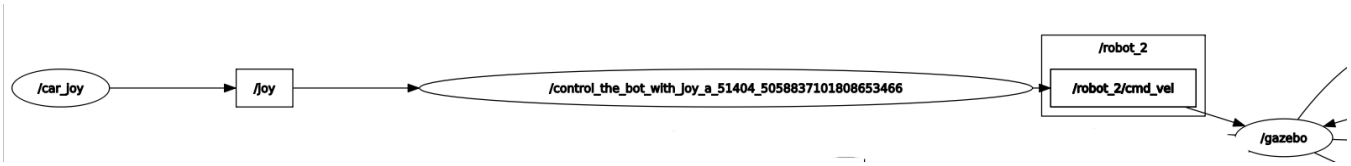
2.4 RQT / packages / nodes :

This is the full rqt graph:



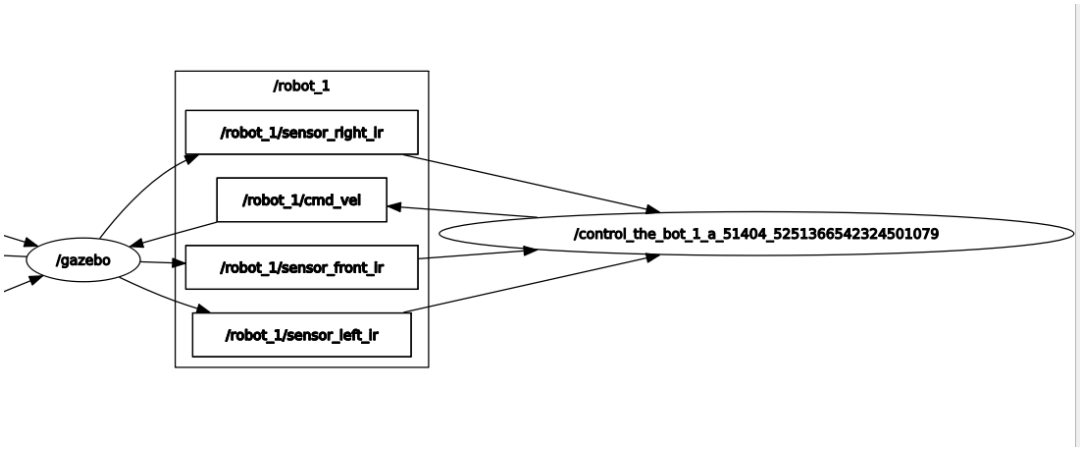
Full RQT graph

In order for the user to control a vehicle with a joystick, it is sufficient to send twisted messages to the wheel controller topic from the "carjoy" node.



Car control by joy RQT

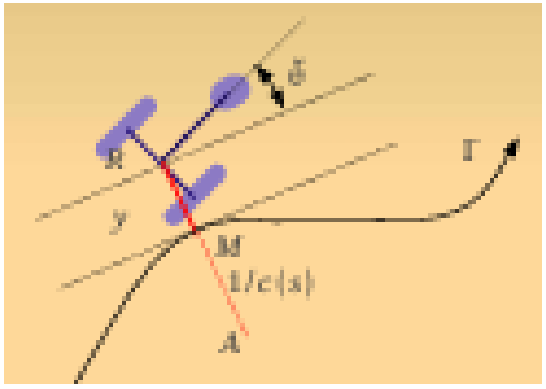
If we now want a vehicle to be controlled by the distance sensors we use the control node. It receives information from the three lidars and deduces a command that will be sent to the robot.



Car control by sensor RQT

The control law is based on a trajectory tracking method.

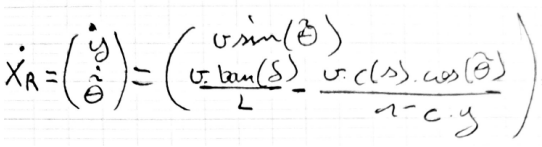
We begin by defining the trajectory to follow as the curve that runs through the circuit, remaining at equal distance from the edges.



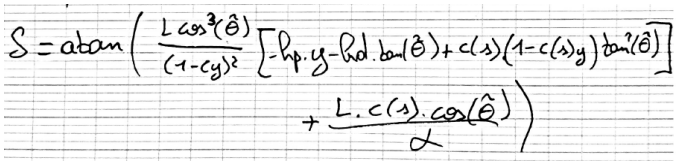
Trajectory tracking diagram

We try to make the deviation from this curve tend towards 0. This means that the lateral lidars measure should tend to the same value when the trajectory is oriented along the objective curve.

This control law involves both the curvature of the trajectory but also the lateral and angular deviation from it : [4]



System model



Control law

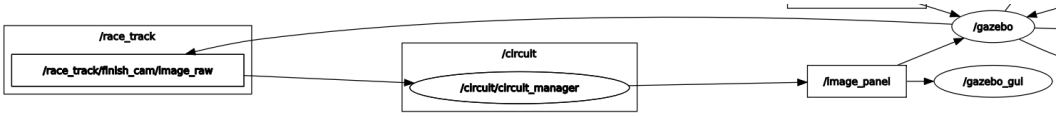
The curvature can be estimated at any time by the following formula with y the lateral deviation: [5]

$$\gamma(x) = \frac{f''(x)}{(1 + f'^2(x))^{3/2}}.$$

Curvature formula

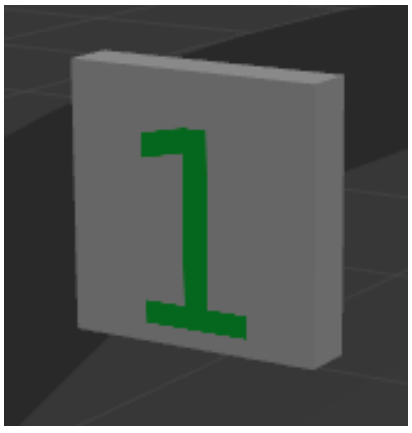
We can also determine the angular error by integrating its derivative as described in the model.

Finally, we find the race manager node which will manage all the elements of the race. It takes care of launching the race and determining the winner based on the color of the wheels measured by a camera at the finish. It also manages the display on the panel of different information such as the countdown of the start and the display of the winner.

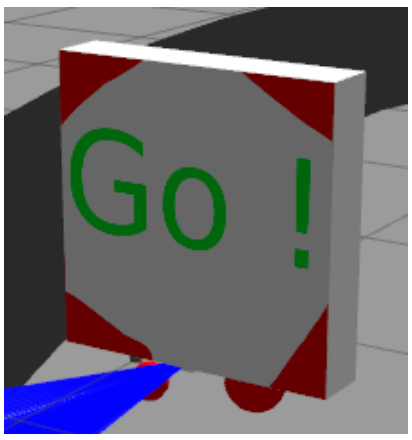


Control the track RQT

The panel :



Control the track RQT



Control the track RQT

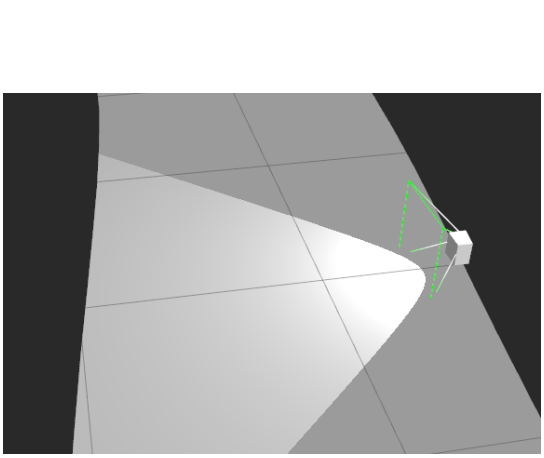
Control the track RQT

The panel displays all useful information for the user.

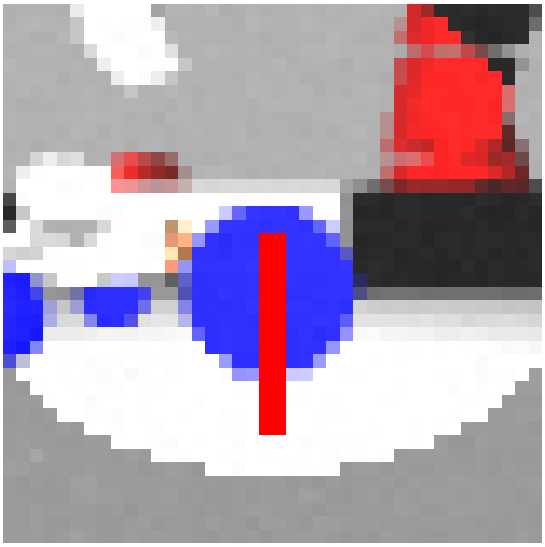
- A 10-second countdown followed by the "Go!" instruction announces the start.

- Displays the winner once the race is over.

2.5 Image processing :



Camera pov



Camera pov

2.5.1 Nodes

In this program, we are utilizing the Robot Operating System (ROS) to control a car using various nodes. The first node, called "mine", is responsible for controlling the car using a joystick. It subscribes to the joystick input and then publishes the corresponding control commands to the car. The second node is for the range sensor, it reads the sensor data by subscribing to it, then process the data by applying a PID control algorithm and finally it sends the control commands to the car through a publisher. The last node is used to manage the track, it reads the image from a camera by subscribing to it, then detects the winner and manages it, and finally sends the image to the panel by publishing it. Each of these nodes perform specific tasks and communicate with each other through the ROS system to control the car effectively and efficiently.

```
NODES
  /
    car_joy (joy/joy_node)
    control_the_bot_1_a_46150_7206956590280369382 (circuit/control_by_sensor.py)
    control_the_bot_with_joy_a_46150_37096707620233343 (circuit/control_by_joy.py)
    gazebo (gazebo_ros/gzserver)
    gazebo_gui (gazebo_ros/gzclient)
    mobile_robot_1 (gazebo_ros/spawn_model)
    mobile_robot_2 (gazebo_ros/spawn_model)
    urdf_spawner (gazebo_ros/spawn_model)
  /circuit/
    circuit_manager (circuit/circuit_manager.py)
```

All nodes

3 Conclusion :

In conclusion, creating an autonomous car in a 3D simulation environment is a complex and challenging programming project that requires a combination of skills in 3D graphics programming, physics simulations, artificial intelligence, and sensor processing. To successfully complete this project, you will need to design and implement a control system that is able to accurately simulate the physics and behavior of a real car, make intelligent decisions about how to navigate the environment, process data from multiple sensors, and handle unexpected or adverse conditions. By using tools such as physics engines, artificial intelligence algorithms, and sensor fusion techniques, you can create a functional and effective autonomous car that is able to navigate and drive itself in a 3D simulation environment. Overall, this project can be a rewarding and educational experience that helps you develop your programming skills and deepen your understanding of autonomous systems.

About ROS, it's an open-source framework that is specifically designed for the development of robotics applications, and it offers a wide range of tools and libraries that can be useful for developing an autonomous car. Overall, using ROS can help to streamline the development process and accelerate the development of an autonomous car in a 3D simulation environment.

References

[1] Wikipedia. Robot operating system, 2021. [https://fr.wikipedia.org/wiki/Robot_Operating_System].

[2] Wikipedia. Blender, 2021. [https://fr.wikipedia.org/wiki/Blender].

[3] Pedro Alcantara. https://github.com/ros-mobile-robots/mobile_robot_description.

[4] Roland Lenain. Commande de robots mobiles à roues, 2019. [SIGMA Clermont: Commande de robots mobiles à roues].

[5] Wikipedia. Courbure d'un arc, 2021. [https://fr.wikipedia.org/wiki/Courbure_arc].