ETH zürich

Institute for Dynamic Systems and Control

Zhichao Sun

A Comprehensive Gazebo Simulation for the Autonomous Racing GoKart Research Platform

Semester Project

Institute for Dynamic Systems and Control
ETH Zurich

Supervision

Dr. Maurilio Di Cicco
Prof. Dr. Emilio Frazzoli

March 2024

IDSC-proj-sp-Sun

IDSC

Abstract

Simulation is pivotal in developing control strategies for mobile robots, enabling researchers to test software components, assess robot behaviors, and evaluate control algorithms under varied environmental conditions. Robot simulators, equipped with advanced physics engines, high-quality graphics, and user-friendly interfaces, allow the theoretical concepts to be tested and optimized in a virtual setting before their application in real-world scenarios. This project introduces a comprehensive simulation framework designed for the IDSC-Frazzoli Go-Kart platform. The framework aims to closely mimic realistic behavior, integrate seamlessly with existing autonomous driving stacks, support multi-agent operations with distinct control mechanisms, and provide adaptable environmental settings.

Keywords: Robotics Simulation, Gazebo, Autonomous Vehicle Racing, Go-Kart, Robotics.

Contents

Chapter 1

Introduction

1.1 Motivation

Robot simulators are increasingly vital in mobile robot research, providing a fast, efficient, and cost-effective means to test new concepts, methods, and algorithms before real-world application.

Currently, Go-Kart platform testing at Winti Lab faces several limitations. Firstly, time and safety concerns arise due to the lab's distance and the capacity for only one test at a time, prolonging testing duration and posing safety risks, especially considering potential crashes. Secondly, the limited availability of only three Go-Karts restricts the scale of testing. Lastly, the constraint in testing scenarios, with only one fixed scenario available, hinders the exploration of diverse testing environments and conditions.

In addition to the limitations mentioned above, the capability of simulation to change layout offers significant advantages. By exploring various sensor layouts before implementing changes to the physical layout of real Go-Kart, simulation provides a valuable opportunity for thorough experimentation. This allows researchers to assess different configurations and optimize sensor design without changing the physical layout on the real Go-Kart, thereby streamlining the design process and ensuring optimal performance in real world scenarios.

Given these benefits, a comprehensive simulation for the Go-Kart platform is essential for overcoming the constraints imposed by real-world testing and maximizing the efficiency and effectiveness of development efforts.

1.2 Related work

In recent years, the development of robot simulators has witnessed significant progress, with various platforms emerging [1, 2], each with its own distinct focus on complexity, accuracy, and flexibility. These simulators offer diverse capabilities, catering to the specific needs and requirements of researchers and developers. Differences exist not only in the level of complexity and accuracy of simulation but also in the flexibility provided for creating and integrating custom robot models and virtual environments.

Accurate physics simulations for realistic interactions, high-quality rendering for precise visual representations, ROS integration for software compatibility, and multi-platform support are essential for advanced robotics research. These features enable effective modeling of robots and environments and support the iterative design process essential for innovation in robotics.

Gazebo [3] is a widely used 3D simulator for the dynamic of rigid bodies. With a readily available package in the official Robot Operating System(ROS) [4] repository, Gazebo is considered as the

standard simulator used in ROS framework. Notably, several well-known robots are simulated on the ROS and Gazebo platform. For instance, Turtlebot [5] functions as a robotics education platform, while simulations of manipulation tasks, such as grasp and place motions, are detailed in [6]. Furthermore, [7, 8] delve into the results and experimental challenges encountered in UAV navigation in simulations.

1.3 Contribution

This work utilizes mesh models to construct a detailed simulation environment for Go-Kart platform. The primary contributions of this study encompass:

- Developing a Go-Kart model in simulation that accurately replicates real-world dynamics, handling, and response characteristics.
- Creating an adaptable environment model that reflects the Winti Lab testing scenario with customization capabilities.
- Designing a simulation framework that integrates seamlessly with existing autonomous driving stack for efficient testing.
- Supporting multi-agent operations with individual control mechanisms for each GoKart.

1.4 Structure of the Thesis

This report begin with the general architecture of ROS and Gazebo in chapter 2. Following this, chapter 3 introduces the design of robot model for Go-Kart, covering aspects including the robot frame, sensors, and actuators, and chapter 4 presents the environment model in simulation, which is designed to replicate the current testing environment (Winti lab). The subsequent chapter 5 elaborates on the design of agents utilized for simulation, including both the manual agent and the MPCC controller. Finally, chapter 6 showcases the results of successfully executing the current manual and autonomous control stack in simulation.

Chapter 2

ROS and Gazebo

2.1 ROS

The Robot Operating System (ROS) [4] is an open-source software framework widely used in robotics, collaborative robot systems, and machine vision research. Its ecosystem offers a diverse range of tools, libraries, and conventions that facilitate the development of complex robotics applications and is compatible with various robotic platforms. ROS operates through Linux-based command tool, features an interprocess communication system alongside extensive repositories of application-related packages. Within this framework, processes are executed as `Nodes` which utilize a `Publish` or `Subscribe` model to communicate through designated `Topics`. `Publishers` disseminate data across these `Topics`, which `Subscribers` can then access. This setup enhances the integration of user-developed libraries with ROS executables, making it a robust environment for development.

Additionally, ROS includes many resources such as sensor drivers, navigation tools, environment mapping utilities, path planning solutions, and visualization tools, which collectively expedite the development and testing of robotic systems.

2.2 Gazebo

Gazebo [3] is an advanced open-source 3D robotics simulator that incorporates OpenGL rendering and supports multiple physics engines including DART [9], ODE [10], Bullet [11] and Simbody [12]. This simulator is capable of rendering robots in 3D spaces with high fidelity, both indoor and outdoor. Gazebo offers an extensive collection of robot models and environmental settings that include a wide range of sensors, along with accessible programmatic and graphical interfaces.

The simulator operates on a `Client/Server` architecture and uses a `Publish/Subscribe` model for interprocess communication, allowing data access through shared memory. This architecture enables precise control over simulation objects by linking them with controllers that process commands and update the object's state. The data is then shared across processes, allowing for comprehensive interaction between the robot control software and the simulator, regardless of the programming language or hardware being used. This system is particularly effective for distributed simulation tasks, where the `Client` and `Server` can be operated on separate machines.

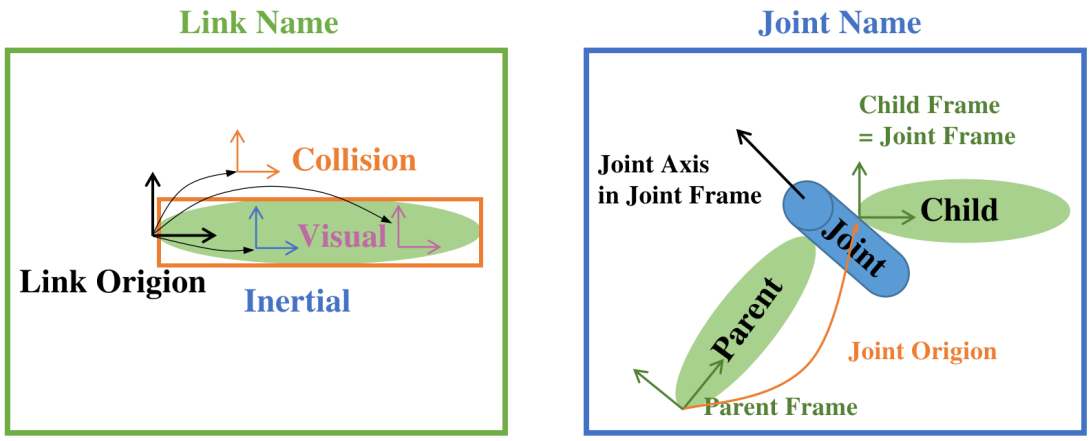
The integration of simulator with ROS is streamlined through the ROS Plugin, creating a seamless interface that mimics the connection between ROS and physical robot hardware. This integration allows for the use of the same control software for both simulated and real robots, significantly enhancing the development and testing process.

2.3 Robot and Environment Modeling

Robot Description Formats. The initial step in creating a robot simulation involves defining the robot's kinematic and dynamic properties. This can be achieved either through Gazebo's graphical user interface or by directly programming the parameters into XML files using formats like URDF (Universal Robot Description Format) or SDF (Simulation Description Format). These formats are essential for detailing the robot's structural properties and are tailored specifically for use in Gazebo or ROS applications.

SDF is tailored for the Gazebo simulator, offering comprehensive description capabilities for the simulated environment and the complete robot model. On the other hand, URDF is primarily developed for ROS applications, focusing on specifying the kinematic and dynamic properties of a robot. To integrate URDF files effectively within the Gazebo simulator, supplementary simulation-specific tags are introduced, encompassing parameters such as robot pose, friction coefficients, inertial properties, and other relevant attributes [13]. These additional properties facilitate the alignment of the original URDF files with Gazebo's native model description format, SDF.

Modeling of Robot Links and Joints. The representation of each link in the Gazebo simulator is achieved by referencing XML files utilizing the `link` variable. As depicted in Figure 2.1(a), each link requires a corresponding 3D model to define its `visual` and `collision` variables, allowing for the specification of properties such as material and surface characteristics. While basic robots can be directly constructed within XML files using primitive geometric shapes provided by built-in XML variables such as boxes, cylinders, and spheres, more complex robots necessitate 3D modeling software. These sophisticated models are referenced in XML files using the `mesh` variable, specifying the path to the 3D model file, typically in STL (.stl), Wavefront (.obj), or Collada DAE (.dae) formats. Coordinate systems within XML files define the origin position with Cartesian coordinates and the orientation with roll, pitch, and yaw angles, rigidly attached relative to the variable.



(a) (b)

Figure 2.1: XML variables (a) Link, (b) Joint.

Inertial properties such as mass, center of mass, and inertia tensor are crucial for accurate simulations in Gazebo's physics engine and are specified in the `inertial` variable of XML files. To calculate the inertia tensor for each robot link, MeshLab software [14] is used. It computes inertia tensors from the mesh volume and unit density, producing numerical values by scaling the tensor components with the corresponding density based on mass and volume.

After defining robot links, their connections are established using the `joint` variable in XML files. As illustrated in Figure 2.1(b), this variable requires specification of both the parent and child link for each joint. Additionally, the type of joint must be defined as follows:

- **Revolute:** A hinge joint that rotates along an axis, with defined upper and lower rotational limits.
- **Continuous:** A hinge joint that rotates around an axis without limits.
- **Prismatic:** A sliding joint that moves along an axis, also with specified range limits.
- **Fixed:** A joint where all degrees of freedom are locked, preventing any movement.
- **Floating:** A joint allowing motion in all six degrees of freedom.
- **Planar:** A joint permitting motion within a plane perpendicular to a specified axis.

Programming and Control Design with ROS. The integration of ROS nodes with Gazebo is facilitated through specialized interfaces provided by the `gazebo_ros_pkgs` package [15]. These interfaces, encompassing topics, services, and dynamic reconfiguration functionalities, bridge the communication gap between ROS nodes and Gazebo. Notably, gazebo plugins establish a communication interface between ROS and Gazebo, facilitating interprocess communication.

Within the ecosystem of Gazebo plugins, a variety of options are detailed in the Gazebo plugins tutorial [16]. These plugins encompass a broad spectrum of functionalities, ranging from sensors such as Cameras, Lasers, IMUs (Inertial Measurement Units), and controllers like the `ros_control` plugin [17]. The `ros_control` plugin, in particular, is equipped with pre-configured controllers designed for standard robotic tasks. Moreover, it provides a versatile framework that allows users to either develop custom controllers or modify existing ones to meet specific operational needs.

Chapter 3

Robot Modeling

In this section, we explore the design elements of the Go-Kart simulation framework, structured into four main components as depicted in Figure 3.1.

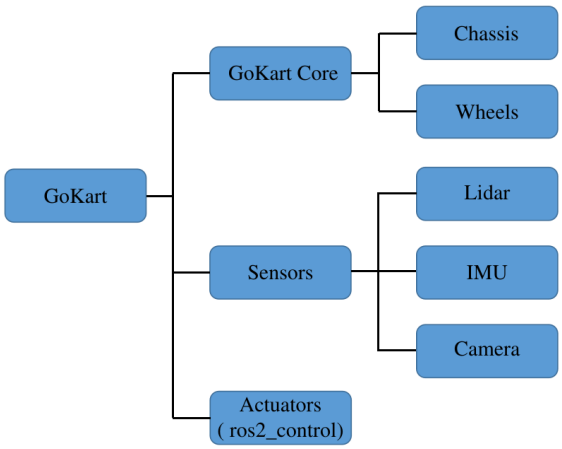
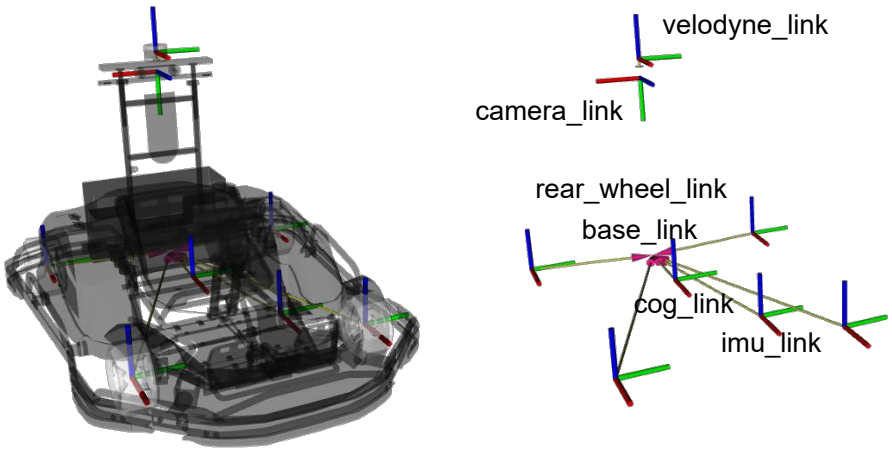


Figure 3.1: Go-Kart simulation design framework.

The framework includes the structural design of the robot frames, the core Go-Kart structure comprising the chassis and wheels, actuators that simulate dynamic behaviors, and the sensors crucial for interaction with the environment.

3.1 Robot Frames

The robot frames encompass three primary components. Firstly, the `base_link` anchors all frame transformations and forms the foundation. It is oriented with the x-axis facing forward, the y-axis pointing left, and the z-axis pointing upward, aligning with the Go-Kart's direction of motion. Additionally, the `cog_link` denotes the center of gravity, vital for stability. The second component comprises sensor links, including essential components such as the Velodyne LiDAR, IMU, and camera. Notably, the LiDAR and IMU contribute to localization capabilities. Finally, the third component consists of the actuators, including `rear_wheel_links` controlled by velocity controllers and `front_wheel_steer_links` controlled by steering controllers. The spatial distribution of these frames is depicted in Figure 3.2, with detailed relationships outlined in Table 3.1 and Table 3.4.



(a) (b)

Figure 3.2: Visualization of Robot Frames in the Go-Kart Simulation Environment. (a) Spatial Distribution of Frames. (b) Naming Convention of Frames.

3.2 Go-Kart Core

In this section, we examine the fundamental elements of the Go-Kart, including chassis dimensions, wheel specifications, and their interconnections via joints. We begin by detailing the chassis' dimensions and discuss the wheels' sizes and configurations. Additionally, we describe the complex network of joints that connect these components, enabling motion and controll. This analysis aims to provide a thorough understanding of the Go-Kart's core design principles and functionalities.

3.2.1 Chassis

The chassis serves as a structural backbone of the Go-Kart. Figure 3.3 illustrates the 3D mesh representation of the chassis, which mimic the design of the real Go-Kart. The dimensions of the chassis are detailed in Figure 3.1. In this work, all mechanical components such as motor and braking system, the autonomous steering mechanism, PC box, power supply box, and other essential components are attached to the chassis.

	Bumblebee		Herbie	
Parameter	Value	Unit	Value	Unit
Vehicle length	2.07	[m]	2.07	[m]
Track width front	1.39	[m]	1.39	[m]
Track width rear	1.45	[m]	1.45	[m]
Tire radius front	0.117	[m]	0.117	[m]
Tire radius rear	0.127	[m]	0.127	[m]
Mass vehicle	283.4	[kg]	281.9	[kg]
Mass tire front	2.0	[kg]	2.0	[kg]
Mass tire back	2.0	[kg]	2.0	[kg]
COG: Length front	1.29	[m]	1.29	[m]
COG: Length rear	0.81	[m]	0.81	[m]
COG: Width left	0.71	[m]	0.71	[m]
COG: Width right	0.71	[m]	0.70	[m]
COG: Height	0.24	[m]	0.24	[m]

Table 3.1: Go-Kart Parts Parameters.

Figure 3.3: 3D mesh representation of the Go-Kart chassis.

3.2.2 Wheels

Go-Karts utilize tread compounds with Dunlop SL1 and SL3 dry track tires as depicted in Figure 3.4(a). Figure 3.4(b) shows the 3D mesh representation of these wheels, which accurately mirror the design of the actual Go-Kart. The dimensions of the wheels are specified in Table 3.1.

(a) (b)

Figure 3.4: Visualization of tires. (a) Real. (b) Simulation.

3.3 Actuators

Actuators convert control commands into physical motion within the Go-Kart. This section details two key actuators: the motor, which controls the velocity of the rear wheels for propulsion, and the steering controller, which adjusts the front wheels' angle for directional control. Together, these actuators manage the dynamic movement and flexibility of the Go-Kart.

3.3.1 Motor

To simulate the Go-Kart's motor, we use a continuous joint connecting the `rear_wheel_joint` to the `base_link`. The positional transformations for both the left and right rear wheel joints are specified in Table 3.2, and they share the same orientation. For example, as shown in Figure 3.5, the `rear_left_wheel_joint` allows the `rear_left_wheel_link` to rotate around the y-axis.

Figure 3.5: Rear Left Wheel Joint Configuration.

We control the velocity of the `rear_left_wheel_joint` using the Forward Command Controller in the Gazebo plugin `libgazebo_ros2_control`. This plugin facilitates real-time dynamics manipulation within Gazebo, applied similarly to the `rear_right_wheel_link`. Motor commands for acceleration or braking are derived from either a manual agent or an MPCC controller and are converted into the appropriate velocities for the motors. This setup ensures precise propulsion control and realistically replicates the motor's behavior in the simulation, enhancing the dynamic motion control of the Go-Kart.

Joint Name	Joint Type	Joint Axis	Parent Link	Child Link	Translation		
					x	y	z
cog_j	fixed	-	base_l	cog_l	0.36	0.0	0.0
rl	continuous	y	base_l	rl_l	-0.077	0.5625	0.0
rr	continuous	y	base_l	rr_l	-0.077	-0.5625	0.0
fl_steer_j	revolute	z	base_l	fl_steer_l	1.129	0.47	-0.015
fl_j	continuous	y	fl_steer_l	fl_l	0.0	0.0	0.0
fr_steer_j	revolute	z	base_l	fr_steer_l	1.129	-0.47	-0.015
fr_j	continuous	y	fr_steer_l	fr_l	0.0	0.0	0.0

Table 3.2: Go-Kart joint frames transformation for Bumblebee and Herbie. (rl: rear_left_wheel, rr: rear_right_wheel, fl: front_left_wheel, fr: front_right_wheel, _j: joint, _l: link)

3.3.2 Steering System

The steering controller offers precise control over the vehicle's directional movement. Through accurately modeling the steering mechanism and leveraging control interfaces, we ensure realistic steering dynamics and responsive control, enhancing the overall fidelity of the Go-Kart simulation.

Steering Controller Simulation

Similar to the motor simulation, the steering controller utilizes revolute joints connecting the `base_link` to the `front_wheel_steering_link` of Go-Kart. Additionally, the `front_wheel_link` is connected to `front_wheel_steering_link` via a continuous joint. Table 3.2 provides comprehensive information on the transformations associated with the steering link and link of the front wheels, specifying their positions in the Go-Kart system. Taking the front left wheel as an example, as shown in Figure 3.6 the steering mechanism revolves around the z-axis of the corresponding `front_left_wheel_sreer_joint`, allowing for continuous steering motion. On top of this, the `front_left_wheel_link` rotates around the y-axis of the `front_left_wheel_joint` .

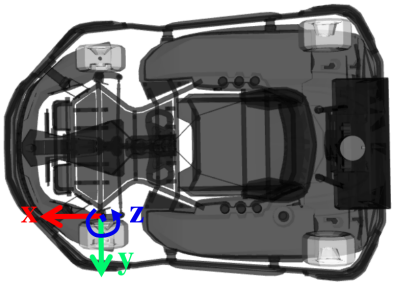


Figure 3.6: Front Left Wheel Joint Configuration.

Similar to the motor, we employ the Forward Command Controller to directly control the position of the `front_left_wheel_steer_joint`. However, for the front wheels, we don't control the velocity of `front_left_wheel_joint`, allowing the front wheels to rotate freely. The Go-Kart receives the required angle of the steering wheel from the steering command, mapping it to the angle of the left and right front wheels.

Ackerman Steering Measurement

The steering mechanism of the Go-Kart adheres to Ackermann Steering principles, ensuring optimal turning dynamics without wheel slip. As illustrated in Figure 3.7(a), this configuration aligns the axles of all wheels to intersect at a common point, facilitating smooth turns. As illustrated in Figure 3.7(b), a simplified approximation to achieve near-perfect Ackermann steering geometry can be attained by shifting the steering pivot points inward. This adjustment aligns the pivot points along a line extending from the steering kingpins, which serve as the pivot points, to the center of the rear axle [18].

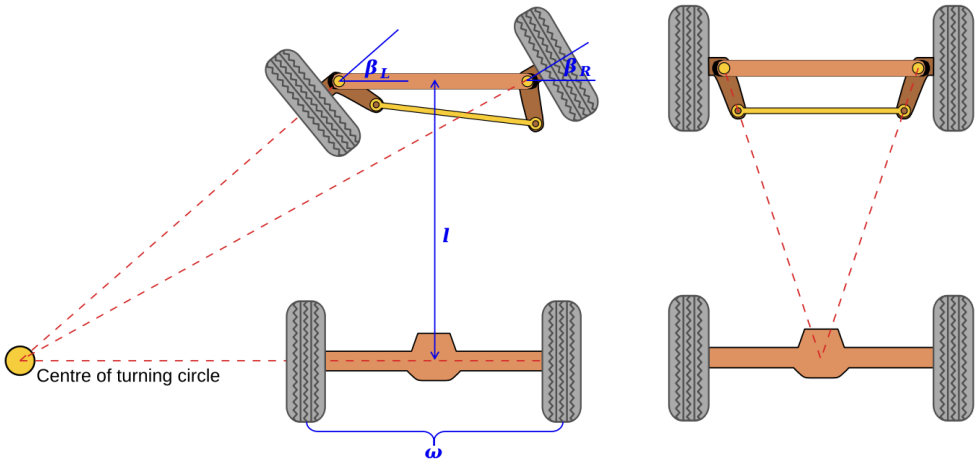


Figure 3.7: Ackermann Steering. (a) Geometry, (b) Design Approximation.

The Ackermann steering angles are contingent upon a steering angle denoted as β_C for an imaginary wheel positioned at the center of the front axle:

$$\beta_L = tan^{-1}(\frac{l \cdot tan(\beta_C)}{l + 0.5\omega \cdot tan(\beta_C)}) \tag{3.1}$$

$$\beta_R = tan^{-1}(\frac{l \cdot tan(\beta_C)}{l - 0.5\omega \cdot tan(\beta_C)}) \tag{3.2}$$

where, as marked in Figure 3.7(a):

$\omega[m]$: Track of the vehicle, for Go-Kart$\omega = 0.94$.
$l[m]$: Wheel base of the vehicle, for Go-Kart$l = 1.27$.
$\beta_C[rad]$: Angle of the steering wheel.
$\beta_L[rad]$: Ackermann steering angle of the left wheel.
$\beta_R[rad]$: Ackermann steering angle of the right wheel.

Despite the model, upon analyzing the steering of the Go-Kart, it becomes evident that the actual mechanism significantly deviates from Ackermann steering principles. Hence, we present measurements aimed at accurately modeling the steering mechanism of the Go-Kart as shown in 3.8. These measurements involved examining the relationship between the electronically measured steering wheel angle β_C and the manually measured left and right wheel steering angles β_L and β_R. This work assumes that the steering mechanism is symmetric within reasonable bounds, so only the right wheel sheering angles are measured.

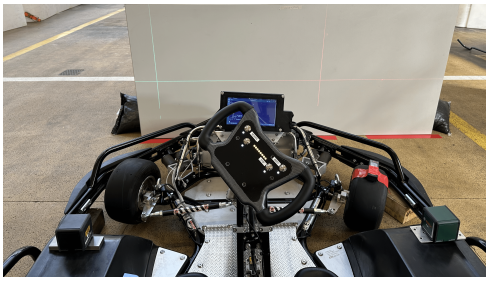


Figure 3.8: Ackermann Steering Measurement Setup.

In the first method, as depicted in Figure 3.9(a), the Go-Kart was aligned with its forward axis perpendicular to a flat surface. A laser pointer was affixed to the midpoint of the right wheel, positioned around 0.126 meters from the wheel hub. Then, the lateral offset m of the projection on the flat surface and the distance d measured by the laser pointer were recorded for different steering wheel angles. The wheel steering angle can be calculated as follows:

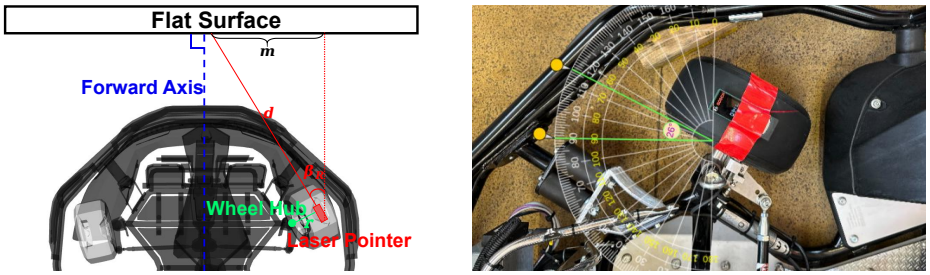
$$m = d \cdot sin(\beta_R) + \frac{r}{cos(\beta_R)} - r \tag{3.3}$$

where, as marked in Figure 3.9(a):

$d[m]$: Distance to the flat surface measured by the laser pointer.

$r[m]$: Sideways distance of the laser pointer from the steering pivot, in this setup$r = 0.126$.

$m[rad]$: Lateral offset of the laser pointer projection on the flat surface.



(a) (b)

Figure 3.9: Ackermann Steering Measurement. (a) Method 1, (b) Method 2.

The second method involved placing a camera on a horizontal surface to capture images of the right wheel. A virtual protractor [19] was then used to measure the steering angle directly. This method served as a sanity check for the first method, as the protractor recorded angles as integers.

Following the collection of measurements corresponding to 24 different steering wheel encoder values as detailed in Table 3.3, the results were graphed in Figure 3.10. Various polynomial degrees were fitted, and the fitting residuals ceased improving upon the use of a third-order polynomial, resulting in the following approximating function for the right wheel:

$$\beta_R(\beta_C) = -0.0355\beta_C^3 + 0.0455\beta_C^2 + 0.36\beta_C \tag{3.4}$$

Given the assumption of symmetry in the steering mechanism, the approximation of the left wheel steering behavior mirrors that of the right wheel:

$$\beta_L(\beta_C) = -0.0355\beta_C^3 - 0.0455\beta_C^2 + 0.36\beta_C \tag{3.5}$$

$\beta_C[rad]$	$d[mm]$	$m[mm]$	$\beta_C[^\circ]$(method 2)
-1.8448	1029	553	-34
-1.6815	1001	510	-32
-1.524	971	460	-30
-1.3642	941	407	-27
-1.2013	913	353	-24
-1.0339	887	298	-21
-0.8847	866	250	-18
-0.7292	846	200	-18
-0.2688	830	150	-11
-0.372	814	92	-7
-0.2065	805	44	-3
-0.0717	800	5	-0
0.0836	797	-33	2
0.2009	794	-63	5
0.3862	794	-105	8
0.5606	797	-141	11
0.7308	799	-170	13
0.882	802	-193	15
1.0411	804	-212	16
1.1932	806	-224	18
1.3541	809	-230	19
1.5208	808	-228	19
1.6865	806	-216	18
1.7319	805	-215	18

Table 3.3: Steering wheel measurements.

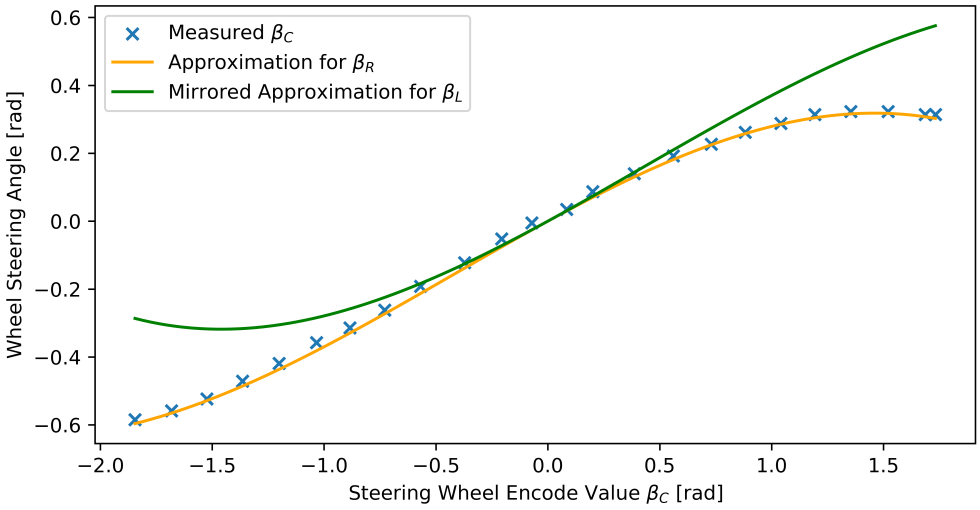


Figure 3.10: Steering wheel measurements.

3.4 Sensors

Sensors provide essential data for environmental perception, navigation, and control for Go-Kart. This section details the sensors deployed on the Go-Kart platform, namely the Velodyne LiDAR, IMU, and camera. Table 3.4 summarizes the spatial transformations of each sensor relative to the `base_link` of Go-Kart. These transformations are essential for accurately interpreting sensor data and integrating it into the Go-Kart's control algorithms.

Go-Kart Type	Child Frame	Translation			Rotation		
		x	y	z	roll	pitch	yaw
Bumblebee	imu_link	0.78	0.26	0.08	0.0	0.0	0.0
	velodyne_link	0.0	0.0	1.154	0.0	0.0	0.0
	camera_link	0.0	0.0	0.875	$-\pi/2$	0.0	$-\pi/2$
Herbie	imu_link	0.745	0.0	0.08	0.0	0.0	0.0
	velodyne_link	0.0	0.0	1.2	0.0	0.0	0.0
	camera_link	0.0	0.0	0.875	$-\pi/2$	0.0	$-\pi/2$

Table 3.4: Go-Kart sensor frames transformation (base_link serves as the parent frame for all).

3.4.1 Lidar

Velodyne lidar serves as a critical element for localization and object detection in the Go-Kart platform. Its capacity to accurately map the environment in three dimensions allows the Go-Kart to precisely ascertain its position and orientation relative to its surroundings. Furthermore, this sensor effectively identifies nearby objects, supplying vital data for obstacle detection.

Types of Sensor

Within the Go-Kart platform project, two distinct Velodyne Lidar models have been selected for their unique functionalities: the VLP-16 and the HDL-32E. Figure 3.11 displays the visual representation of these two models, and Table 3.5 outlines their properties.

(a) (b)

Figure 3.11: Visual Representation of Velodyne LiDAR. (a) VLP-16, (b) HDL-32E.

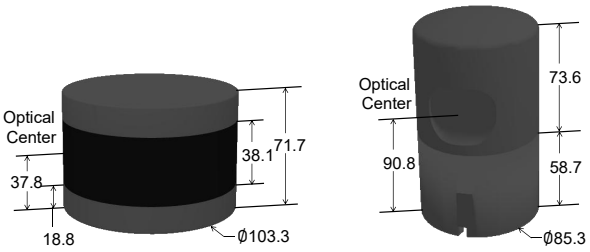
Properties	VLP-16	HDL-32E
Channel	16	32
Measurement Range $[m]$	100	100
Range Accuracy $[cm]$	± 3	± 2
Field of View (Vertical) [°]	$+15.0 \sim -15.0(30.0)$	$+10.67^{\sim} - 30.67(41.33)$
Angular Resolution (Vertical) [°]	2.0	1.33
Field of View (Horizontal) [°]	360	360
Angular Resolution (Horizontal) [°]	$0.1 \sim 0.4$	$0.08 \sim 0.33$
Rotation Rate $[Hz]$	$5 \sim 20$	$5 \sim 20$

Table 3.5: Properties for Velodyne LiDAR.

Sensor Model in Simulation

To ensure accurate simulation of the LiDAR, several steps are required. First, a mesh representing the LiDAR's physical appearance must be created to closely resemble its actual appearance. Next, the LiDAR's position and orientation need to be meticulously aligned with the LiDAR's real-world setup on the simulated Go-Kart. Lastly, the sensor must be configured in the simulation to exhibit behavior consistent with real sensors. Adhering to these steps ensures that the LiDAR's functionality is accurately replicated and that it provides reliable data within the simulation.

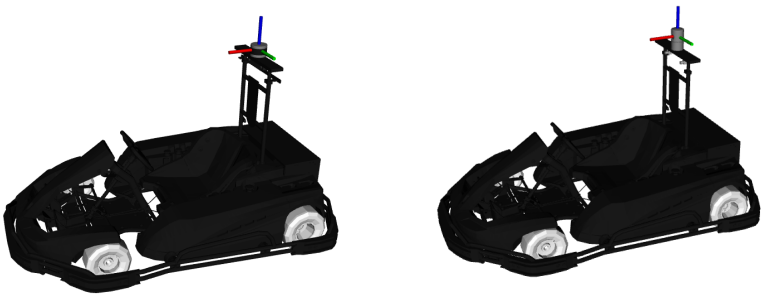
Visual Representation in Simulation. This section offers a visual representation of the sensors' mesh and configuration within the virtual environment. Figure 3.12(a) illustrates the mesh representation of the VLP-16 LiDAR sensor, and Figure 3.12(b) displays the mesh representation of the HDL-32E LiDAR sensor, showcasing their design and components.



(a) (b)

Figure 3.12: Visual Representation of Velodyne LiDARs in Simulation. (a) VLP-16, (b) HDL-32E.

Mounting on Go-Kart Core. Accurate simulation of the Velodyne LiDAR sensors' perspective and coverage on the Go-Kart platform depends critically on their precise mounting positions. This section details these positions using a transformation provided in Table 3.4, which maps the `base_link` of the Go-Kart to `velodyne_link` for proper alignment. Additionally, Figure 3.13 visually depicts the LiDAR sensors' placement and orientation on the simulated Go-Kart, illustrating how they are configured within the simulation environment.



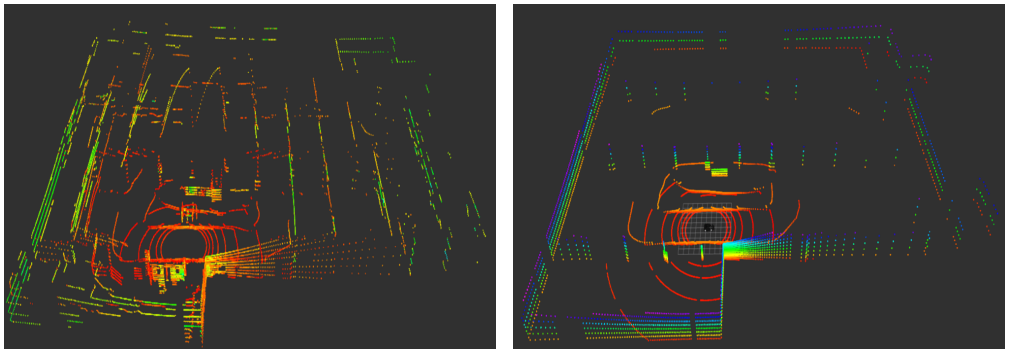
(a) (b)

Figure 3.13: LiDAR Sensor Placement on Go-Kart in Simulation. (a) Bumblebee, (b) Herbie.

Integration with Gazebo Simulation. In this work, the Velodyne LiDAR sensors are integrated into the Gazebo simulation environment using the `velodyne_gazebo_plugins` [20]. This plugin facilitates the configuration of critical parameters like vertical and horizontal range, update rate, and the number of lasers and samples, as detailed in Table 3.5. It's noteworthy that although this plugin effectively replicates the LiDAR sensors' data acquisition behaviors, it does not mimic the actual spinning motion of the real sensors.

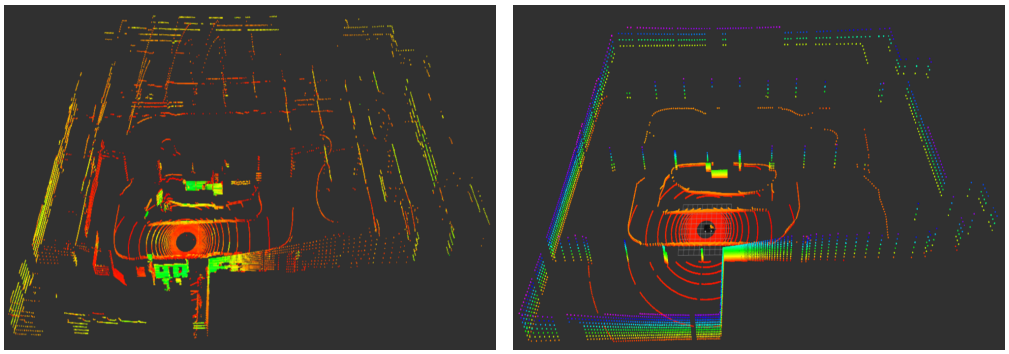
Comparison of Real and Simulated Results

The comparison between real-world and simulated results of VLP-16 and HDL-32E are shown in Figure 3.14 and 3.15. This comparison reveals significant similarities, with lower levels of noise in simulated data. Despite such differences, he consistent performance of both sensor models in tasks like localization and object detection validates the simulation's accuracy in emulating real-world LiDAR operations.



(a) (b)

Figure 3.14: Comparison of Real and Simulated Results for VLP-16. (a) Real, (b) Simulation.



(a) (b)

Figure 3.15: Comparison of Real and Simulated Results for HDL-32E. (a) Real, (b) Simulation.

3.4.2 IMU

The Inertial Measurement Unit (IMU) accurately measures acceleration, angular velocity, and orientation, enabling the Go-Kart to monitor its dynamic state and movement in real time. Additionally, the IMU works in conjunction with the LiDAR to improve localization stability and accuracy, thereby improving the Go-Kart's navigation capabilities.

Types of Sensor

The Go-Kart platform uses two specific IMU sensor models. The Bumblebee model employs the IRIMU-V2 unit from Izze-Racing (Figure 3.16(a)), which measures acceleration and angular rates across three orthogonal axes. The Herbie model features the 3DM-GX5-AHRS IMU sensor from MicroStrain Sensing (Figure 3.16(b)), a high-performance, industrial-grade sensor that delivers a

full range of triaxial inertial measurements and navigation outputs. Detailed specifications for the measurement range, accuracy, and data output rate of each sensor are outlined in Table 3.6.

(a) (b)

Figure 3.16: Visual Representation of IMU sensors. (a) IRIMU-V2, (b) 3DM-GX5-AHRS.

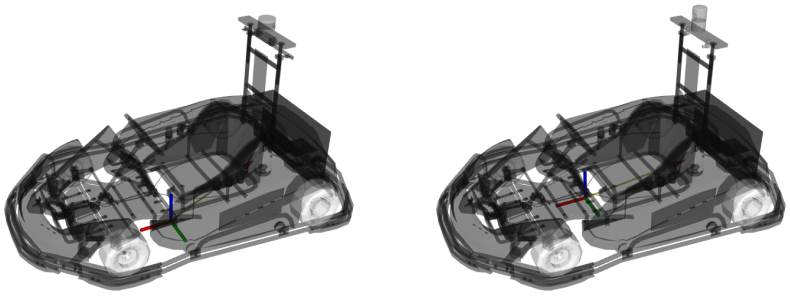
Properties	IRIMU-V2	3DM-GX5-AHRS
Accelerometer Range $[m^2/s]$	$\pm 2, \pm 4, \pm 8$(default), ± 16	$\pm 2, \pm 4, \pm 8$(default), $\pm 20, \pm 40$
Accelerometer Accuracy $[fs]$	$< 1\%$	$\pm 0.02\%$
Gyroscope Range $[^\circ/s]$	± 245(default), $\pm 500, \pm 2000$	$\pm 75, \pm 150, \pm 300$(default), ± 900
Gyroscope Accuracy $[fs]$	$< 1.5\%$	$\pm 0.02\%$
Data Output Rate $[Hz]$	$1 \sim 1000$	$10 \sim 480$

Table 3.6: Properties for IMUs.

Sensor Model in Simulation

To accurately simulate the IMU, several critical steps are required. First, as IMUs are embedded within the Go-Kart chassis and not visible externally, so no physical representation is created for them in this simulation. Next, it is essential to align the IMU's position and orientation on the simulated Go-Kart accurately with its real-world setup. Lastly, the simulation must configure the IMU to behave as it does in reality. Adhering to these steps ensures that the IMU's functionality is realistically replicated, providing reliable data for simulation use.

Mounting on Go-Kart Core. Precise placement of the IMU sensors on the Go-Kart is crucial for accurately monitoring its state. Table 3.4 shows the transformation matrix from the Go-Kart's `base_link` to the `imu_link`, ensuring exact alignment. Figure 3.17 illustrates the simulated position and orientation of the IMU sensors, providing a clear visual reference for their setup within the simulation environment.



(a) (b)

Figure 3.17: IMU Sensor Placement on Go-Kart in Simulation. (a) Bumblebee, (b) Herbie.

Integration with Gazebo Simulation. In this work, the IMU sensors are integrated into the Gazebo simulation environment using the `GazeboRosImu` plugin from package `gazebo_plugins` [21]. This plugin enables the configuration of critical parameters including the update rate and noise levels for each axis, tailored to align with the specifications listed in Table 3.6.

Comparison of Real and Simulated Results

The comparison between real and simulated IMU data shows remarkable consistency in key state parameters of the Go-Kart, including localization, velocity, and acceleration. Furthermore, integrating IMU data with LiDAR sensors in the simulation enhances localization stability, leading to reliable and consistent Go-Kart positioning. These results confirm the fidelity and accuracy of the IMU sensor in simulation.

3.4.3 Camera

The camera system, though not yet operational on the Go-Kart platform, offers considerable potential for enhancing environmental perception and decision-making. It enables the Go-Kart to identify objects, navigate complex environments, and avoid obstacles effectively.

Types of Sensor

The Go-Kart platform integrates two distinct cameras. For the Bumblebee model, a ZED camera (Figure 3.18(a)) is employed. The ZED is a stereo camera renowned for its high-definition imaging and precise depth measurement capabilities, making it well-suited for demanding applications such as autonomous vehicle control, mobile mapping, and security. The Herbie model features an acA800-510uc camera (Figure 3.18(b)) from Basler. This camera is equipped with an onsemi PYTHON 500 CMOS sensor, known for its reliability, high-speed data transfer and seamless integration.

(a) (b)

Figure 3.18: Visual Representation of cameras. (a) ZED, (b) Basler ac800-510us.

Sensor Model in Simulation

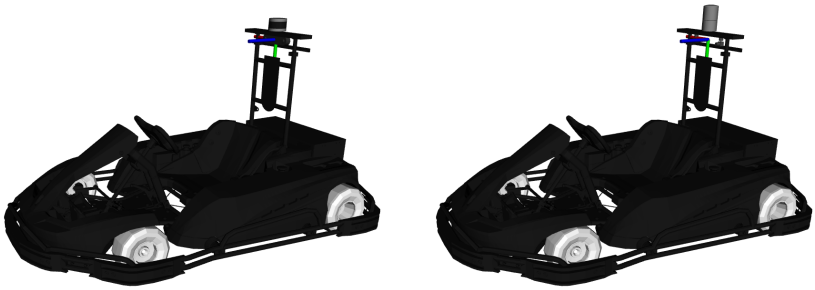
To accurately simulate the cameras, three key steps are undertaken. First, a model resembling the camera's real-world physical appearance is created. Next, the camera's position and orientation on the simulated Go-Kart are aligned with its actual placement. Finally, the camera's parameters like resolution, frame rate, and exposure settings, is configured in simulation to mirror real-world operations.

Visual Representation in Simulation. The mesh representation of the cameras in the simulation offers a rough visual portrayal of their physical appearance and structure within the virtual environment. Figure 3.19(a) and Figure 3.19(b) illustrates the mesh representation of the ZED camera and the Basler ac800-510us camera, offering a visual representation of their design and structural features.

(a) (b)

Figure 3.19: Visual Representation of Cameras in Simulation. (a) ZED, (b) Basler ac800-510us.

Mounting on Go-Kart Core. Since the camera is not currently in use, the positioning of the camera in simulation is based on a rough estimation and should be refined once the camera is actively utilized. Table 3.4 offers details of the transformation from the `base_link` to the `camera_link` in simulation. Additionally, Figure 3.20 illustrates the simulated position and orientation of the cameras on the Go-Kart, offering a preliminary depiction that will be further refined upon integration of the camera system.



(a) (b)

Figure 3.20: Camera Placement on Go-Kart in Simulation. (a) Bumblebee, (b) Herbie.

Integration with Gazebo Simulation. This work integrates the cameras into the Gazebo simulation environment using the `GazeboRosCamera` plugin from the `gazebo_plugins` package [21]. This plugin enables the configuration of various parameters such as update rate, resolution, and horizontal field of view (FOV). It is important to note that these parameters are currently not calibrated to match the the actual cameras installed on the Go-Kart. Instead, they are configured for visualization purposes within the simulation environment.

Simulated Results

The simulation results of the cameras are depicted in Figure 3.21. From the visual data, it is evident that the cameras effectively capture the environment.

(a) (b)

Figure 3.21: Simulated Results for camera. (a) ZED, (b) Basler ac800-510us.

Chapter 4

Environment Modeling

In this section, we explore the detailed design elements of the environment simulation aimed at replicating the Winti lab within our framework, as depicted in Figure 4.1.

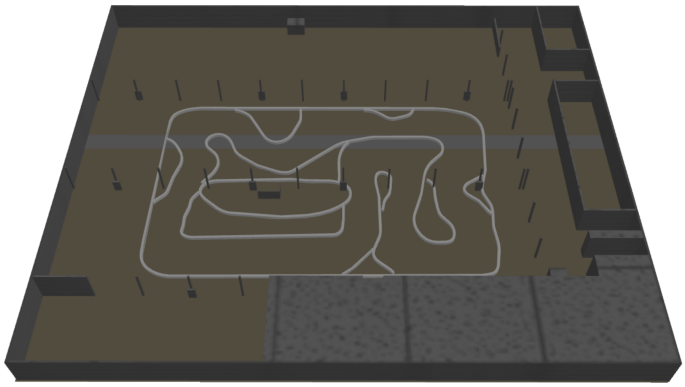


Figure 4.1: Environment simulation design framework.

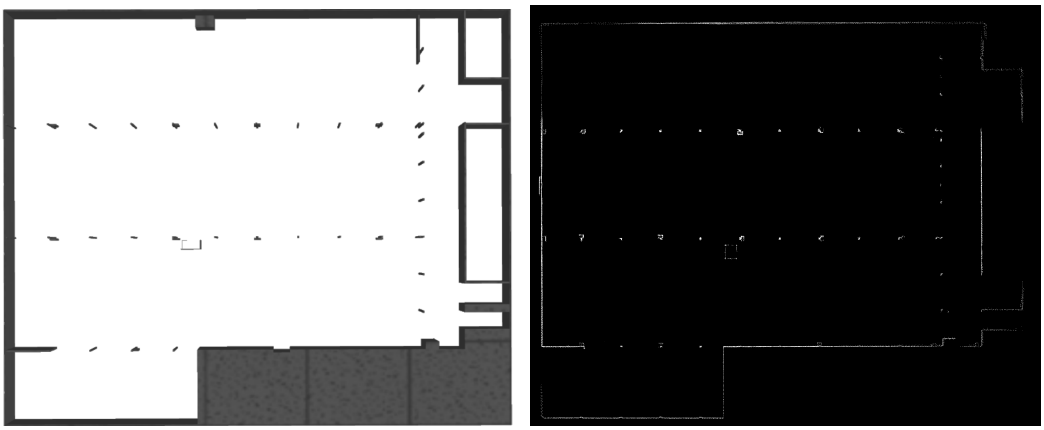
The simulation is divided into three primary components. As illustrated in Figure 4.2, the first component delves into the architectural design of the Winti lab building. The second component focuses on the floor design of the Winti lab. Lastly, the third component focuses on the design of the track layout within the lab.

(a) (b) (c)

Figure 4.2: Visualization of components in the Winti Lab Environment Simulation. (a) Floor, (b) Building, (c) Track.

4.1 Building

The building serves as a pivotal component for localization, as the localization algorithm relies on a map generated from Velodyne data collected at Winti lab. Therefore, to ensure accurate localization in simulation, it is important that the building structure within the simulation precisely mirrors that of the physical Winti lab environment. Figure 4.3(a) depicts the mesh representation of the Winti lab building, created by Marcus Aaltonen. By comparing it to the map currently used for localization on the real GoKart in Figure 4.3(b), we observe a perfect alignment.



(a) (b)

Figure 4.3: Visualization of Winti Lab Building Mesh and Map. (a) Floor, (b) Building, (c) Track.

Moreover, this work has aligned the axis of the building in simulation with those used in the localization algorithm. This alignment enables not only the execution of localization in simulation but also direct access to the GoKart's location from Gazebo model states [22].

4.2 Floor

Two distinct floor surfaces are present inside the Winti lab, as depicted in Figure 4.4, each offering different friction characteristics.

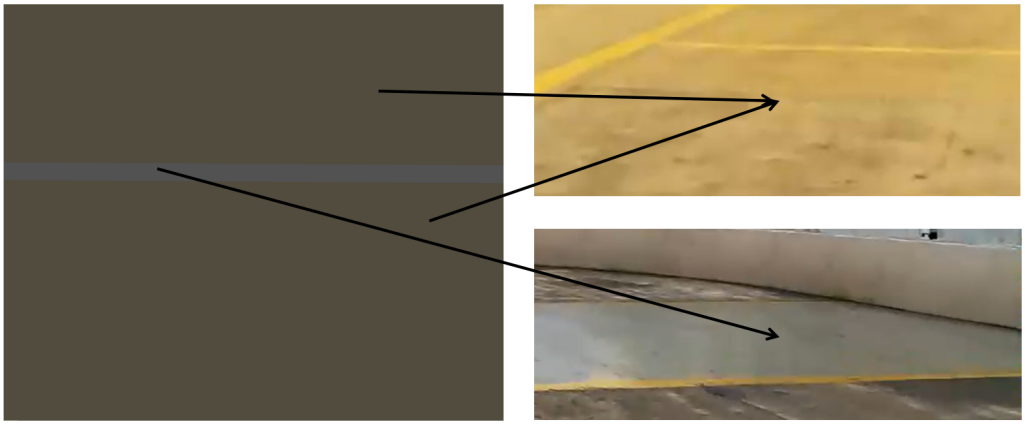
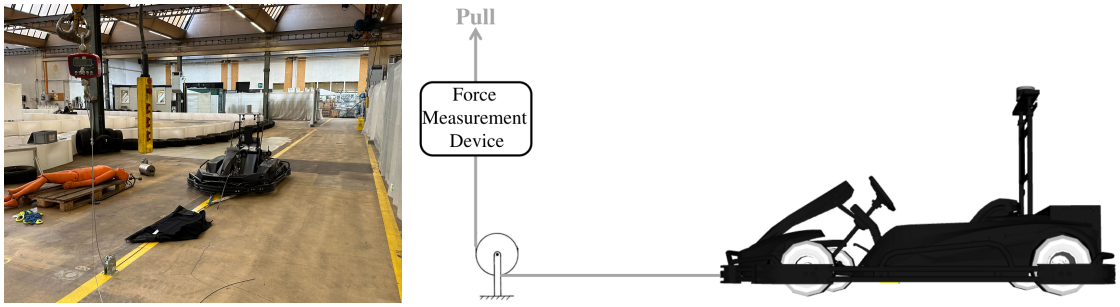


Figure 4.4: Floor Surfaces in Winti Lab.

To quantify the friction of each surface, a measurement setup shown in Figure 4.5 was constructed. During the measurement process, the brake of the GoKart is engaged to prevent the rear wheels

from moving. Subsequently, the GoKart is pulled using a rope positioned perpendicular to the ground, while the force measurement device records the pull force which is equal to the friction force f. With knowledge of the GoKart's weight G, the friction coefficient μ is calculated using the equation $\mu = \frac{f}{G}$. This measured friction coefficient is then utilized to configure the friction properties for the respective floor surfaces within the simulation environment.

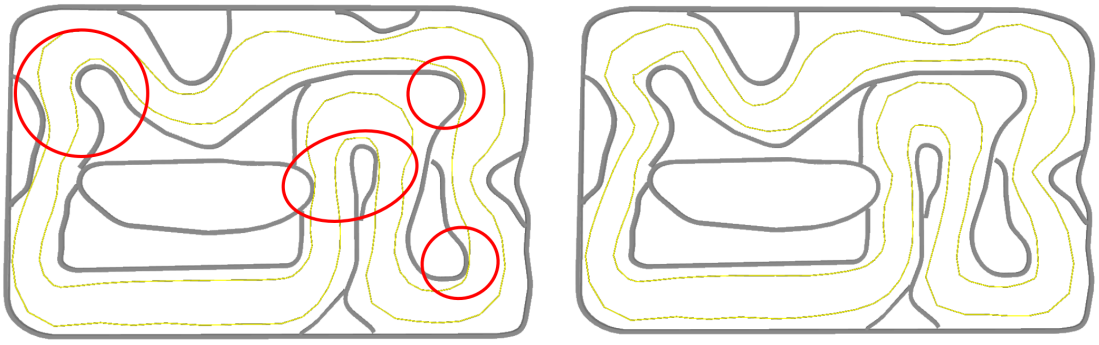


(a) (b)

Figure 4.5: Measurement Setup for Floor Surface Friction. (a) Visualization, (b) Building, (c) Schematic Diagram.

4.3 Track

The track holds significant importance as it serves as the path along which the GoKart operates. This is especially crucial for the MPCC controller, which utilizes a spline to delineate the permissible regions for the GoKart's movement. Figure 4.6(a) illustrates the spline currently employed on the real GoKart, which aligns with the existing track layout in Winti lab. However, as highlighted in red in Figure 4.6(a), mismatches between the spline and the track mesh were identified. In these areas, the spline is positioned too closely to one side of the track, potentially leading to collisions. To mitigate this risk, adjustments were made to the spline according to the track mesh, as depicted in Figure 4.6(b). With this modified spline, the GoKart should safely navigate the track without collisions.



(a) (b)

Figure 4.6: Comparison of Spline and Track Mesh. (a) Original Spline, (b) Modified Spline.

It is noteworthy that should the need arise to run the MPCC on a completely different track in simulation, the only requirement is the creation of a new spline tailored to the new track layout.

Chapter 5

Agents

In this chapter, we explore the design and implementation of agents which shape the behavior and control mechanisms of the GoKart within the simulation environment, encompassing both the manual agent and the MPCC controller.

5.1 Manual Agent

The manual agent involves utilizing controllers to maneuver the GoKart within the simulation environment, mirroring real-world driving experiences. This section delves into the selection of controller, implementation details and control mechanisms employed for manual operation.

5.1.1 Controllers

The choice of controllers plays a crucial role in emulating realistic driving experiences within the simulation environment. Given that the GoKart utilizes Ackermann steering geometry, it is essential to simulate the steering wheel accurately. As such, we have selected the PS4 joystick and the Logitech G29 Driving Force Racing Wheel and Floor Pedals as our primary controllers.

(a) (b)

Figure 5.1: Manual Agent Controller Configeration. (a) PS4 Joystick, (b) Logitech G29 Driving Force Racing Wheel and Floor Pedals.

PS4 Joystick. As shown in Figure 5.1(a), the PS4 joystick provides intuitive control over steering and throttle inputs. Its analog sticks allow for precise steering adjustments, while the trigger buttons offer responsive throttle and brake modulation.

Logitech G29 Driving Force Racing Wheel and Floor Pedals. As depicted in Figure 5.1(b), the Logitech G29 Racing Wheel and Floor Pedals offer an immersive driving experience, closely resembling the sensation of driving a real vehicle. Additionally, the racing wheel boasts advanced force feedback technology, and the floor pedals provide accurate control over throttle, and brake inputs, further enhancing the realism of the driving experience.

5.1.2 Control Mechanisms

The control mechanism of the manual agent involves utilizing the `joy` package in ROS [23] to monitor all inputs from the controllers. Figure 5.1 displays the selected inputs that correspond to throttle, brake, steering wheel, reverse driving, and emergency functions. A configuration file is generated to specify the selection of inputs, facilitating easy customization for different controllers. Subsequently, a `manual_agent` node is developed to map the inputs from the controllers to the commands sent to the GoKarts, as depicted in Figure 5.2(a). This modular approach allows for seamless integration of different controllers by simply modifying the joy configuration file, thereby enhancing flexibility and ease of use.

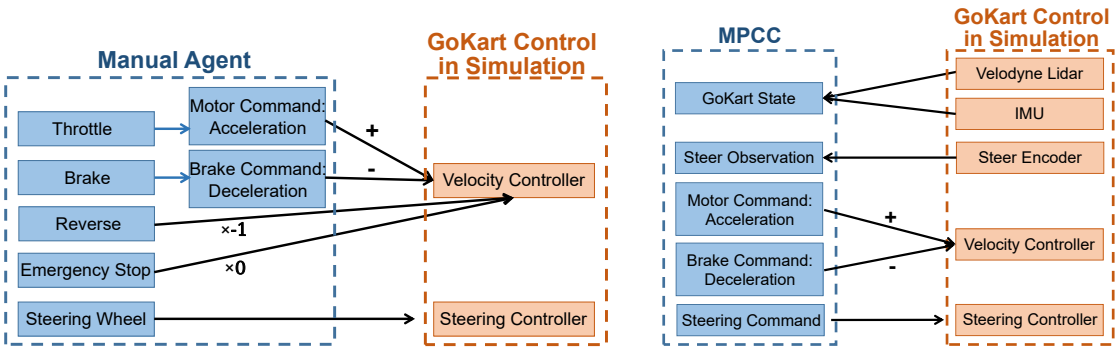


Figure 5.2: Mapping ofController Inputs to GoKart Commands. (a) Manual Agent, (b) MPCC.

5.2 MPCC Controller

The control mechanism of the MPCC controller in simulation closely mirrors that of the real GoKart. As depicted in Figure 5.2(b), utilizing lidar and IMU data, the localization algorithm determines the state of the GoKart. Additionally, the MPCC controller obtains the current steering wheel angle observation from the encoder in the simulation. Subsequently, based on the current GoKart state, the MPCC controller calculates the required steering wheel angle, motor acceleration, and brake deceleration.

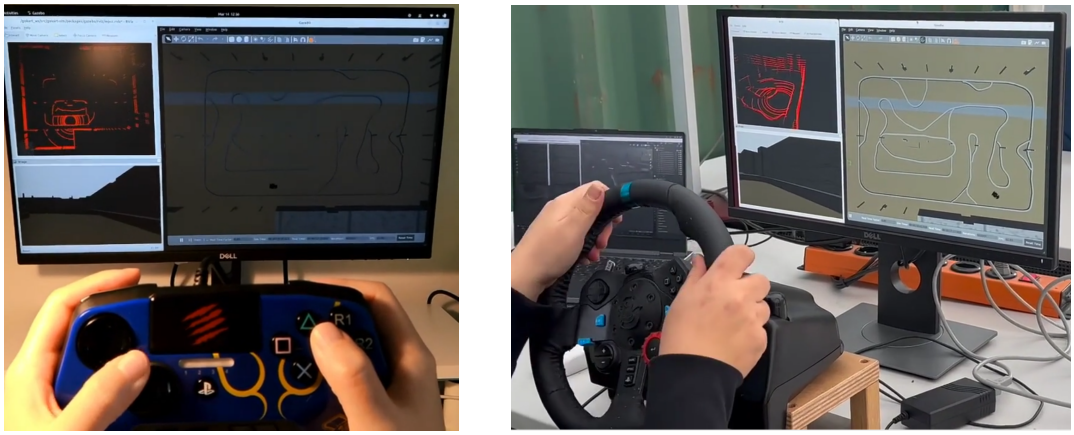
The primary distinction between the control mechanisms of the real and simulated GoKarts lies in the format of the commands sent. For the real GoKart, the steering command is the required torque for turning the steering wheel, while for the simulated GoKart, it is the desired steering angle. Similarly, the motor command sent to the real GoKart involves the current fed into the motor, whereas for the simulated GoKart, it represents acceleration. Additionally, the brake command for the real GoKart acts on the mechanical brake of the back wheels, while for the simulated GoKart, it signifies deceleration.

Chapter 6

Results

In this section, we present the outcomes of simulations conducted on the GoKart platform within the Gazebo environment. The results are categorized into several key aspects, each representing a critical component of the autonomous driving system. We begin by evaluating the performance of the manual agent. Subsequently, we delve into the assessment of localization accuracy, followed by an analysis of the autonomous driving capabilities enabled by the MPCC controller. Additionally, we examine the effectiveness of obstacle detection and avoidance mechanisms. Finally, we explore scenarios involving multiple agents.

6.1 Manual Agent



(a) (b)

Figure 6.1: Performance of Manual Agent with Different Input Devices.

In this section, we evaluate the efficacy of the manual agent, enabling human operators to navigate the GoKart within the simulation environment. Figure 6.1 illustrates the usage of two different controller types for steering the GoKart in simulation. For a comprehensive demonstration, please refer to the video of using joystick[1] and the video of using the driving wheel[2], showcasing the controller's ability to regulate speed, steering, and facilitate reverse driving, mirroring real-world GoKart operations.

[1] results/result_manual_agent_joystick.gif
[2] results/result_manual_agent_drivewheel.gif

6.2 Localization

This part evaluates the performance of the localization algorithm by comparing the estimated position of the GoKart with ground truth data acquired from model states within the simulation environment. Figure 6.2 displays the obtained motion-compensated point cloud from running localization in simulation. For a more comprehensive demonstration, please refer to the demonstration video[3]. The motion-compensated point cloud demonstrates stability, and the estimated position by localization aligns closely with the state of the GoKart model in Gazebo. These results indicate that localization operates effectively and exhibits enhanced stability in simulation compared to real-world scenarios.

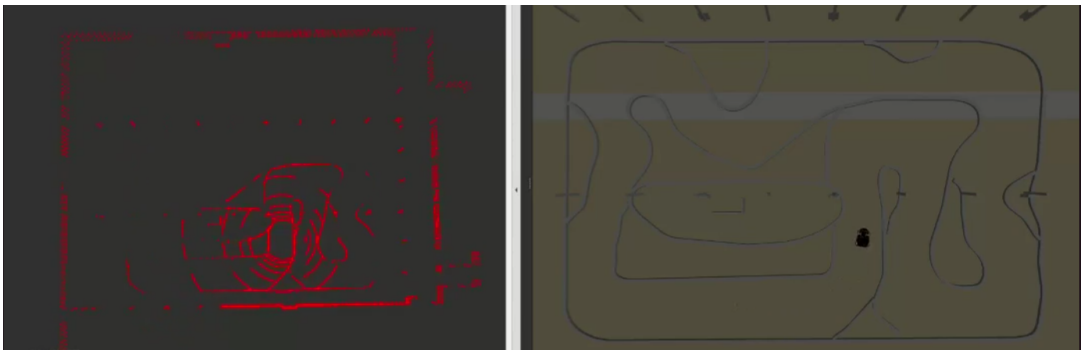


Figure 6.2: Result of Running Localization in Simulation.

6.3 MPCC Controller Automatic Driving

In this section, we analyze how the MPCC controller guides the GoKart autonomously through predefined trajectories and scenarios in the simulation environment. Figure 6.3 illustrates the spline horizon (in green or red line) and horizon track (in blue lines) of MPCC, providing insight into the controller's trajectory planning. Additionally, the accompanying video[4] demonstrates the GoKart successfully completing full laps without incident.

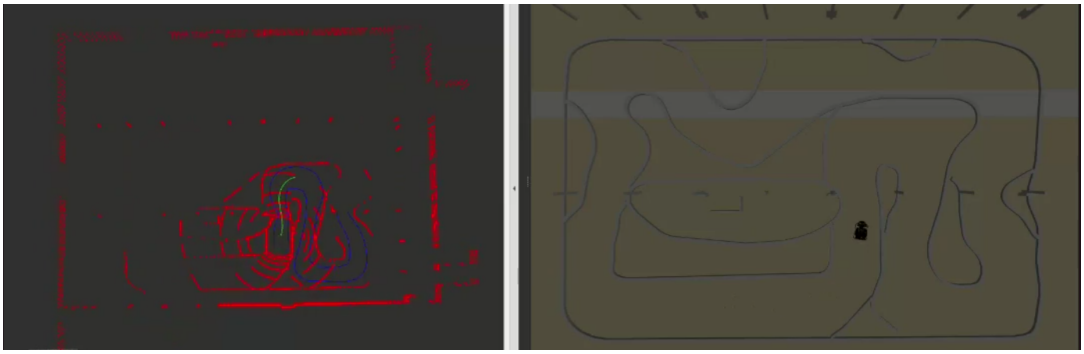


Figure 6.3: Result of Running MPCC in Simulation.

[3] results/result_localization.gif
[4] results/result_mpcc.gif

6.4 Obstacle Detection

This section scrutinizes the effectiveness of the obstacle detection mechanism within the simulation environment. Figure 6.4 showcases the bounding boxes delineating detected obstacles, with green indicating static obstacles and red denoting dynamic ones. Additionally, the camera view of the GoKart provides context for the actual location of the obstacles. For a comprehensive demonstration of obstacle detection, please refer to the accompanying video[5], where you can observe the detection of various obstacles.

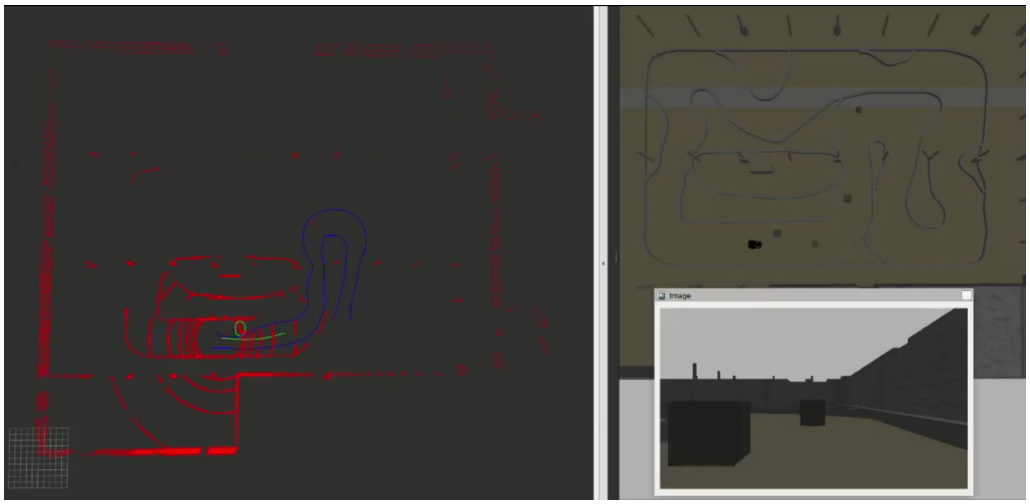


Figure 6.4: Result of Running Object Detection in Simulation.

6.5 Obstacle Avoidance

Building upon the obstacle detection capabilities, obstacle avoidance mechanisms are designed to autonomously maneuver the GoKart to avoid collisions with detected obstacles. However, due to limitations in the current obstacle detection method's accuracy and efficiency, we resort to obtaining obstacle locations using model states in Gazebo, representing the ground truth locations of obstacles. These locations are depicted as green circles on the left side of Figure 6.5. Additionally, the horizon track (illustrated by blue lines) is adjusted to circumvent the obstacles. For a comprehensive demonstration, please refer to the accompanying video[6], where you can witness the successful avoidance of all obstacles by the GoKart.

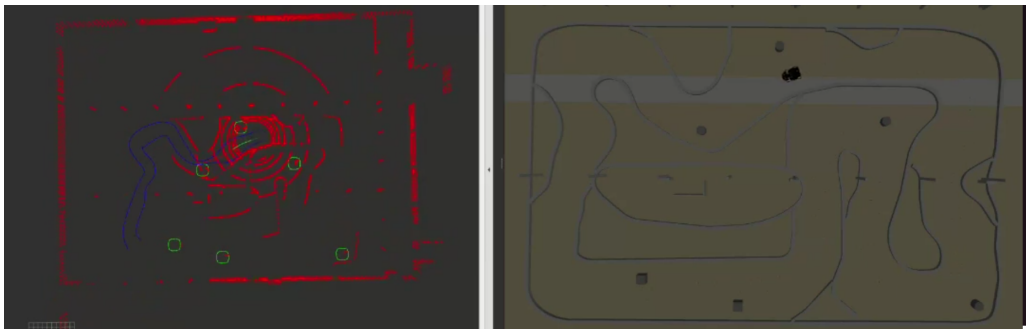


Figure 6.5: Result of Running Obstacle Avoidance in Simulation.

[5] results/result_obstacle_detection.gif
[6] results/result_obstacle_avoidance.gif

6.6 Multi-Agent Scenario

While multi-agent scenarios haven't been tested in real-world settings due to the risk of collisions, their development in simulation holds significant value. Within the simulation environment, multiple GoKarts can be spawned with their own namespaces so that we can handle their data separately/ As illustrated in Figure 6.6, where two GoKarts are spawned. The leading GoKart is controlled by a manual agent, while the other is controlled by an MPCC controller. On the left side, we observe the MPCC spline of the MPCC agetn alongside the camera view of the manually controlled GoKart. In the comprehensive demonstration video[7], the MPCC agent identifies the other GoKart as an obstacle. Given that the manual agent operates at a slower speed than the MPCC agent, the latter manages to maneuver past it. However, in certain scenarios, collisions between the GoKarts occur. This underscores the importance of simulation as a valuable tool for testing multi-agent interactions before implementation on real GoKarts.

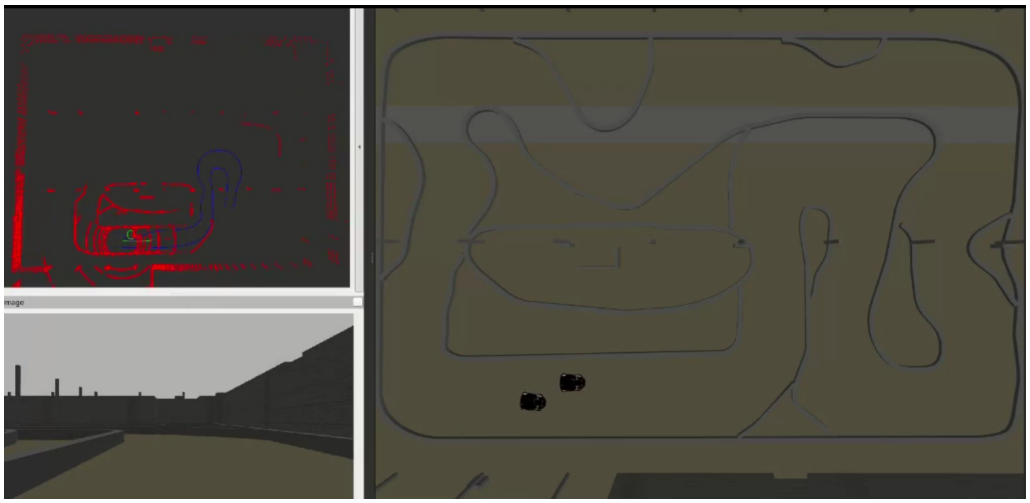


Figure 6.6: Result of Multi-Agent Scenario in Simulation.

[7] results/result_multi_agent.gif

Chapter 7

Conclusion

This project presents a comprehensive exploration of the simulation framework for the GoKart platform. The simulation closely mirrors real-world counterpart of the GoKart, both in terms of physical design and control mechanisms. This fidelity ensures that the behavior of the simulated GoKart accurately reflects that of the real GoKart, facilitating comprehensive testing and evaluation. Furthermore, the simulation environment itself closely resembles the conditions found in the Winti Lab, providing an immersive setting for testing and experimentation. This environment also offers precise GoKart and obstacle localization capabilities, which contribute to creating a realistic testing environment, allowing for thorough evaluation and validation of autonomous systems. Additionally, successful execution of the autonomous control stack in the simulation environment validates its feasibility and effectiveness in navigating and controlling the simulated GoKart.

Despite the fidelity of the simulation environment, there remain potential discrepancies between simulated and actual conditions that may impact result accuracy. Variations in sensor and actuator performance between simulation and reality could introduce differences in data and control execution, affecting the reliability of test outcomes. Therefore, careful consideration and validation are necessary when interpreting results obtained from simulation experiments.

Future work will focus on enhancing the simulation framework to address identified limitations and further improve its utility and reliability. First, we can employ system identification techniques to refine motor and steering plugins, increasing accuracy and realism in simulation. Additionally, comprehensive validation of the simulation environment can be conducted to ensure fidelity and reliability across various scenarios. Furthermore, the implementation of a simulation-based continuous integration pipeline will automate testing procedures, ensuring software quality and performance across diverse algorithms. These efforts will contribute to advancing the capabilities and effectiveness of the simulation platform for GoKart experimentation and development.

Overall, the simulation platform holds great promise for advancing GoKart experimentation and development, providing a valuable tool for testing and validating autonomous systems in a safe and controlled environment. With continued refinement and validation, the simulation framework will play a pivotal role in accelerating progress and innovation in autonomous vehicle technology.

Chapter 8

Acknowledgement

I want to thank the Frazzoli Group with Professor Emilio Frazzoli and my supervisor Dr. Maurilio Di Cicco for for their invaluable guidance, insightful feedback, and continuous support throughout this project. Additionally, I am grateful to Marcus Aaltonen, Shengjie Hu, and Matteo Penlington for their assistance in conducting experiments and measurements at Winti lab.

Bibliography

[1] A. Farley, J. Wang, and J. A. Marshall, "How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion," *Simulation Modelling Practice and Theory*, vol. 120, p. 102629, 2022.

[2] M. Santos Pessoa de Melo, J. Gomes da Silva Neto, P. Jorge Lima da Silva, J. M. X. Natario Teixeira, and V. Teichrieb, "Analysis and comparison of robotics 3d simulators," in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, 2019, pp. 242–251.

[3] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[4] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[5] R. Amsters and P. Slaets, "Turtlebot 3 as a robotics education platform," in *Robotics in Education*, M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, and D. Obdržálek, Eds. Cham: Springer International Publishing, 2020, pp. 170–181.

[6] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang, "Manipulation task simulation using ros and gazebo," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 2594–2598.

[7] J. García and J. M. Molina, "Simulation in real conditions of navigation and obstacle avoidance with px4/gazebo platform," *Personal and Ubiquitous Computing*, vol. 26, no. 4, pp. 1171–1191, 2022.

[8] S. Chen, W. Zhou, A.-S. Yang, H. Chen, B. Li, and C.-Y. Wen, "An end-to-end uav simulation platform for visual slam and navigation," *Aerospace*, vol. 9, no. 2, p. 48, 2022.

[9] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. Karen Liu, "Dart: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018.

[10] R. Smith *et al.*, "Open dynamics engine," 2005.

[11] E. Coumans, "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, 2015, p. 1.

[12] M. A. Sherman, A. Seth, and S. L. Delp, "Simbody: multibody dynamics for biomedical research," *Procedia Iutam*, vol. 2, pp. 241–261, 2011.

[13] Gazebo Development Team, "Tutorial: Using a urdf in gazebo," http://classic.gazebosim.org/tutorials/?tut=ros_urdf, 2020.

[14] "Meshlab," https://www.meshlab.net/, Visual Computing Lab - ISTI - CNR, 2023, accessed: 07 March 2024.

[15] "Gazebo-ros packages github page," https://github.com/ros-simulation/gazebo_ros_pkgs, accessed 02 March 2024.

[16] Gazebo Development Team, "Tutorial: Using gazebo plugins with ros," https://classic.gazebosim.org/tutorials?tut=ros_gzplugins, 2021.

[17] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke *et al.*, "ros_control: A generic and simple control framework for ros," *Journal of Open Source Software*, vol. 2, no. 20, pp. 456–456, 2017.

[18] W. Norris, *Modern Steam Road Wagons*. Longmans, 1906.

[19] "Online protractor tool," https://www.ginifab.com/feeds/angle_measurement/, accessed 15 March 2024.

[20] "Velodyne Gazebo Plugins," https://wiki.ros.org/velodyne_gazebo_plugins, accessed 10 March 2024.

[21] "Gazebo Plugins," https://wiki.ros.org/gazebo_plugins, accessed 10 March 2024.

[22] "Gazebo Model state," https://github.com/ros-simulation/gazebo_ros_pkgs/wiki/ROS-2-Migration:-Entity-states, accessed 15 March 2024.

[23] ROS. Joy. http://wiki.ros.org/joy. Accessed 12 March 2024.

ETH zürich

Institute for Dynamic Systems and Control

Prof. Dr. R. D'Andrea, Prof. Dr. E. Frazzoli, Prof. Dr. Lino Guzzella, Prof. Dr. C. Onder, Prof. Dr. M. Zeilinger

Title of work:

A Comprehensive Gazebo Simulation for the Autonomous Racing GoKart Research Platform

Thesis type and date:

Semester Project, March 2024

Supervision:

Dr. Maurilio Di Cicco
Prof. Dr. Emilio Frazzoli

Student:

Name: Zhichao Sun
E-mail: zhisun@student.ethz.ch
Legi-Nr.: 22-958-227
Semester: FS 2024

Statement regarding plagiarism:

By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Declaration of Originality:

https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf

Zurich, 13.5.2024: 孙朝超