Project 1: CAD Modeling and Simulation using Gazebo

ENPM662: Introduction To Robot Modeling
Anirudh Swarankar (UID: 121150653)
Munyaradzi P Antony (UID: 120482731)
Pranav Deshakulkarni Manjunath (UID: 121162090)
Varad Nerlekar (UID: 120501135)

Under the guidance of Dr. Reza Monfaredi

Contents

1 Introduction

This project involved the design and implementation of a Cyber Truck and Trailer (Robot) model, which was initially created in Fusion 360 and later transferred to SolidWorks. Each component was designed separately, ensuring precise dimensions and accurate mass properties. The assembly process included applying constraints to allow for rotational movements, specifically using revolute joints to establish the necessary degrees of freedom for the robot's functionality. Once the assembly file was organized and optimized, it was exported as a URDF file. The next step involved integrating a LIDAR sensor into the model to facilitate the visualization of LIDAR data points in RVIZ. This visualization was essential for the subsequent phase, which enabled teleoperation of the robot within a Gazebo simulation environment. For teleoperation, two control methods were developed: one using keyboard inputs and another utilizing joystick control. A Gazebo model plugin was employed to manage the robot's kinematics and joint controllers, ensuring synchronization across different controllers. This plugin also handled the computation and publication of important data, such as odometry. The final stage of the project involved creating a simple proportional controller to guide the robot from a starting point A(0,0) to a destination point B(10,10). It was assumed that the robot would initially align with either the y-axis or the x-axis.

1.1 Workflow

In summary the flow of work is to proceed as follows

- Model creation in Fusion 360 and SolidWorks
- Conversion of the model for ROS2 compatibility
- Integration of the controller and LIDAR sensor
- Activation of teleoperation features
- Implementation of the proportional controller

1.2 Inspiration

Our design drew significant inspiration from the Tesla Cybertruck, known for its futuristic aesthetics and robust functionality. We aimed to replicate its distinctive angular shape and durable materials, which embody both innovation and strength. The Cybertruck's unique features, such as its off-road capabilities and cutting-edge technology, inspired us to integrate advanced sensors and ensure high maneuverability in our robot model.

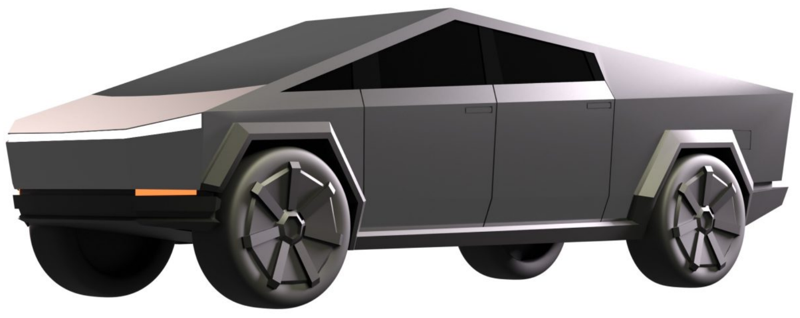

Figure 1: Inspiration from Tesla Cybertruck

2 Procedure

2.1 Building CAD model

- Definition of requirements -The dimensions and constraints required by the question were the ones that were used, the chassis width of 38 inches, the Breadth of the chassis (Lateral) 20 inches, the wheelbase 20 inches, the Wheel thickness of 3 inches and the Wheel diameter 8 inches.
- Creating individual parts - The model was made from two major components the car and the trailer. Car parts were created separately i.e. chassis, wheels and axles and the trailer parts were also designed separately i.e. trailer, wheels and axle, and hitch connector.
- Assembling the car- all the parts were aligned. The chassis and axles were fixed in place.
- Assembling the trailer - this involved mating the wheels and axles allowing proper alignment. The trailer body was then added and the hitch was positioned in the appropriate area in preparation for connection with the car assembly
- Connecting the car and the trailer assembly - a new top-level assembly was made which contained both the car and the trailer
- Adding details and features - to make the robot realistic attributes like color and extra mass were added
- Testing the assembly - the connected car and the trailer assembly were moved around to check on the hitch connection and see if the trailer naturally followed the car.

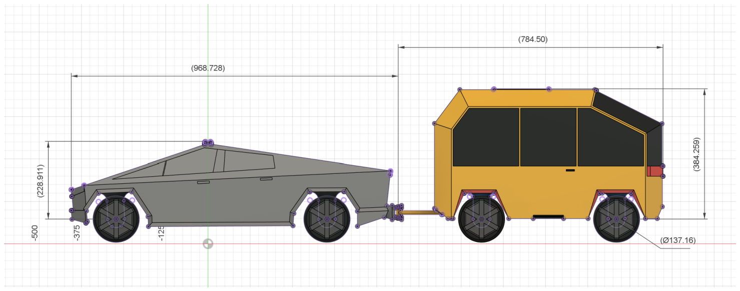

Figure 2: CAD Model in Fusion 360

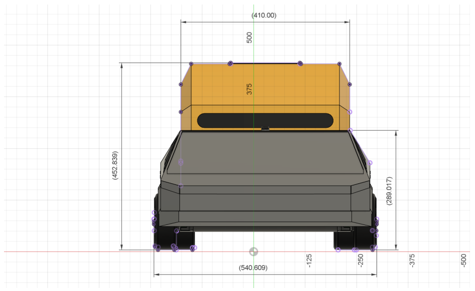

Figure 3: CAD Model in Fusion 360

The parts were then joined together using the assembly option provided in SolidWorks. This allowed mating of components allowing proper alignment and to subject connection to proper constrains.

2.1.1 Challenges

- Software Familiarity: We opted to use Fusion 360 for creating model parts individually due to our team's familiarity with its workflow, which added slight overhead for the integration into SolidWorks later on.
- Material Constraints: Determining the correct material constraints required extensive experimentation. Incorrect weight combinations led to issues such as wheel wobbling and unintended movement without control, necessitating several adjustments to achieve a stable design.
- Updating Frames and Axes: Each component's frames and axes needed careful updating along the joints. Failing to do so resulted in incorrect spawning of the robot in Gazebo, which disrupted the physics simulation and affected testing.
- URDF Conversion: The URDF converter tool add-on in SolidWorks exports the package in ROS1 format, which presented challenges when integrating with newer ROS2 systems, requiring additional steps for compatibility (as discussed in the next section).

2.2 URDF Conversion to ROS2 package

Converting our URDF model to a ROS2-compatible package involved several critical steps to ensure proper functionality and integration with ROS2 framework. Below are the key steps we took during this conversion process:

- **Removing Encoding from XML Header:** The original URDF files generated by SolidWorks URDF tool included an XML header with an encoding specification that was not compatible with ROS2. We needed to manually edit the header to align it with the expected format, allowing for proper parsing by the ROS2 framework.
- **Using XACRO with URDF:** To improve modularity and minimize redundancy, we utilized XACRO (XML Macros) to streamline our URDF files.
- **Updates to CMakeLists.txt and package.xml:** We made considerable changes to the CMakeLists.txt and package.xml files to ensure compliance with ROS2 standards. This included updating dependencies, properly linking necessary libraries, and defining build instructions that work with the ROS2 build system (Colcon).
- **Launch File Restructuring:** We created and adjusted launch files to fit the new package structure in ROS2. This involved setting up node configurations, defining parameters, and ensuring that all robot components were instantiated correctly within the launch environment.
- **Overall Package Restructure:** The conversion process required a comprehensive restructuring of the package layout to adhere to ROS2 conventions. This included organizing directories for source code, configuration files, and resources, which is crucial for effective package management and ease of navigation within the ROS2 workspace.

2.3 Mathematical Modeling

1. **Mathematical Modeling:**

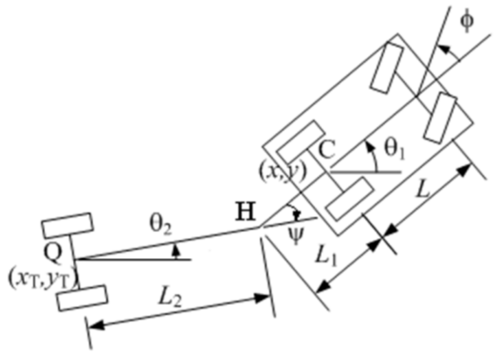

Figure 4: Pictorial Of The Set Up

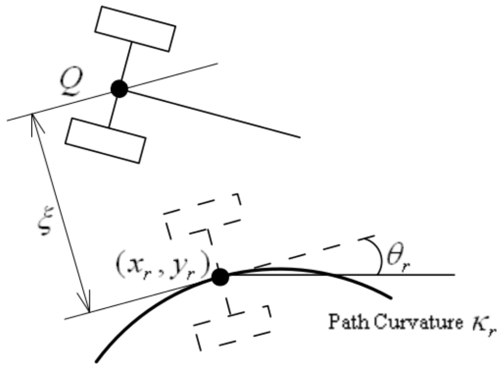

Figure 5: Free Body Diagram

a. Tractor-Trailer Kinematic Model

The geometry of a vehicle-trailer system is shown in Fig.4, where:

- ϕ is the front wheel angle with respect to the vehicle's longitudinal axis.
- θ_1 represents the vehicle heading.
- (x, y) is the vehicle's position, defined at the center of the rear axle C.
- θ_2 is the trailer heading, and $\psi = \theta_2 - \theta_1$ defines the hitch angle.
- (x_T, y_T) is the trailer's position, defined at the center of its rear axle Q.
- L is the vehicle wheelbase, L_1 is the hitch length (distance between C and hitch point H), and L_2 is the trailer tongue length (distance from H to Q).

According to the standard kinematic model[1], the trailer velocity v_T is derived from the hitch point velocity v_H as follows:

$$v_T = v\left[\cos(\psi) - L_1 F_\kappa(\phi)\sin(\psi)\right] \tag{1}$$

The trailer trajectory curvature κ_2 is calculated by:

$$\kappa_2 = \frac{\dot{\theta}_2}{v_T} = \frac{\sin(\psi)/L_2 + L_1 F_\kappa(\phi)\cos(\psi)/L_2}{\cos(\psi) - L_1 F_\kappa(\phi)\sin(\psi)} \tag{2}$$

This curvature is controlled directly by the steering wheel angle ϕ.

b. Steering Controller Development

A. Path Tracking

For the vehicle-trailer system, the kinematic model is defined as:

$$\dot{x} = v\cos(\theta_1) \qquad \dot{y} = v\sin(\theta_1) \qquad \dot{\theta}_1 = \frac{v\tan(\phi)}{L} \tag{3}$$

$$\dot{\psi} = v\left(\frac{\sin(\psi)}{L_2} - \left(\frac{L_1}{L}\right)\frac{\tan(\phi)\cos(\psi)}{L_2}\right) \tag{4}$$

To improve control accuracy, a general vehicle-trailer model is proposed as:

$$\dot{x} = v\cos(\theta_1) \qquad \dot{y} = v\sin(\theta_1) \qquad \dot{\theta}_1 = vF_\kappa(\phi) \tag{5}$$

$$\dot{\psi} = v\left(\frac{\sin(\psi)}{L_2} + \frac{L_1 F_\kappa(\phi)\cos(\psi)}{L_2}\right) \tag{6}$$

where $F_\kappa(\phi)$ is the steering wheel map that relates the steering wheel angle ϕ to vehicle curvature.

To determine the trailer velocity v_T at point Q, the hitch point velocity v_H must be calculated. Given:

$$\Omega = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \quad \text{and} \quad v = \begin{bmatrix} v\cos\theta_1 \\ v\sin\theta_1 \\ 0 \end{bmatrix} \tag{7}$$

$$v_H = v + \Omega \times CH = v\begin{bmatrix} \cos\theta_1 + L_1 F_\kappa(\phi)\sin\theta_1 \\ \sin\theta_1 - L_1 F_\kappa(\phi)\cos\theta_1 \\ 0 \end{bmatrix} \tag{8}$$

c. Curvature Control

From equation (6), if the desired trailer motion curvature κ_d is known, the required steering angle ϕ_d can be determined as:

$$\phi_d = F_\kappa^{-1}\left(\frac{\kappa_d L_2\cos\psi + \sin\psi}{\kappa_d L_1 L_2 \sin\psi - L_1\cos\psi}\right) \tag{9}$$

To address potential inaccuracies in trailer parameters and environmental disturbances, a steering gain k_ϕ is introduced, modifying the control function as follows:

$$\phi_d = k_\phi F_\kappa^{-1}\left(\frac{\kappa_d L_2\cos\psi + \sin\psi}{\kappa_d L_1 L_2 \sin\psi - L_1\cos\psi}\right) \tag{10}$$

The gain k_ϕ is typically set slightly above 1 (e.g., 1.1 or 1.2) and can be tuned based on chattering levels near the path. Higher values improve resistance to inaccuracies and disturbances but may increase chattering.

d. Steering Wheel Map

The function $F_\kappa(\phi)$ represents the relationship between the steering wheel angle ϕ and the vehicle's trajectory curvature, which is generally nonlinear and challenging to model precisely. In this application, $F_\kappa(\phi)$ is considered velocity-invariant, as trailer backing is typically conducted at low speeds (below 0.7 m/s).

To construct $F_\kappa(\phi)$, the vehicle is driven manually with fixed steering angles at low speed, and the trajectory radius is measured. Also an asymptotic curvature [2] stabilization can be achieved

2.4 Gazebo Plugins

2.4.1 LiDAR Sensor

The integration of the LiDAR sensor into the robot model was a critical step in enhancing the robot's perception capabilities. The LiDAR sensor provides detailed distance measurements of the environment by emitting laser beams and analyzing the reflected signals. This data is crucial for navigation, obstacle detection, and mapping.

To implement the LiDAR sensor, we modified the Unified Robot Description Format (URDF) file, incorporating a $< gazebo >$ tag specifically for the LiDAR sensor. This ensures proper simulation in Gazebo.

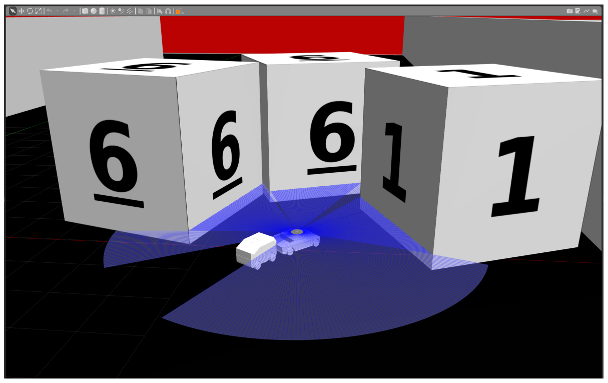

Figure 6: LiDAR visualization on Gazebo

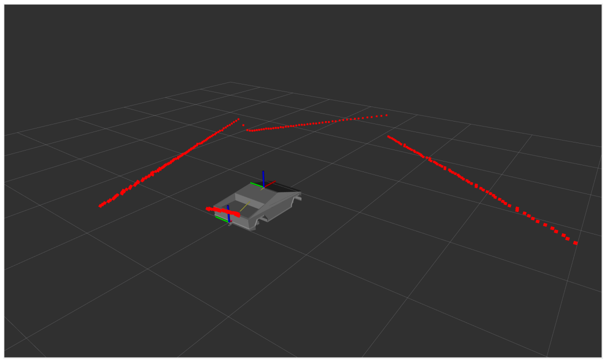

Figure 7: LiDAR visualization on RViz

Key Components:

- Pose: Specifies the position and orientation of the sensor relative to the robot.
- Update Rate: Sets the frequency of data updates to 5 Hz.
- Ray Configuration: Defines the scanning parameters, including:
  - Horizontal Samples: 360 samples for a complete 360-degree field of view.
  - Range: Specifies the minimum (0.12 m) and maximum (3.5 m) detection ranges.
  - Noise Model: Implements a Gaussian noise model to simulate real-world inaccuracies.

RViz Setup: To visualize the LiDAR data in RViz, perform the following steps:

- Add a **LaserScan** display and set the topic to */scan*.
- Set the Fixed Frame to *lidar_link* to ensure accurate visualization from the LiDAR frame.
- Change the Reliability setting to **BestEffort** to handle any potential data loss during transmission.

Through these integrations and visualizations, we ensured that the LiDAR sensor effectively contributes to the robot's navigation and obstacle avoidance capabilities, providing reliable data for autonomous operations.

2.4.2 Challenges

- Coordinate System Alignment: Initially, the coordinate system of the LiDAR was aligned with the base of the sensor, which was mounted within a cutout on the truck chassis. This caused the LiDAR to inadvertently detect the inside of the chassis roof, along with distant points in the environment. To resolve this, we translated the frame of the LiDAR component a few inches upwards along the sensor cutout, ensuring it no longer scanned the interior of the chassis.
- Field of View Limitations: The design of the truck trailer posed another challenge. Its considerable height restricted the LiDAR's field of view toward the back of the truck, limiting the sensor's ability to scan the environment in that direction. To address this, we considered adding an additional LiDAR scanner positioned above the trailer, as altering the trailer dimensions was not a feasible option.

2.4.3 Car Gazebo Plugin

To enable the simulation and control of our robot model in the Gazebo environment, we developed a custom plugin named *CarGazeboPlugin*. This plugin integrates the vehicle's dynamics with ROS communication, allowing for real-time interaction and control. Below is an overview of the key components and functionalities of the plugin.

Package structure:

- The CarGazeboPlugin class extends gazebo::ModelPlugin and includes the following main functionalities:
- Initialization: The plugin initializes parameters and settings in the Load method, which is executed when the model is loaded into the Gazebo simulation. This process involves setting up ROS nodes and publishers for joint states and vehicle odometry.
- Vehicle Dynamics: The plugin features a CyberModel class based on a bicycle model that simulates vehicle dynamics. This model enables the adjustment of steering angles and curvature according to the robot's physical parameters, such as wheelbase length and wheel diameters.
- Joint Control: The plugin uses Gazebo's joint controller to manage the steering and axle joints, allowing precise control over the robot's movement. It incorporates PID controllers to maintain stable and responsive joint movement.
- ROS Communication: The plugin subscribes to drive commands and joystick inputs via ROS topics. It processes incoming messages to modify steering angles and velocities, thereby supporting teleoperation and automated control.

Key Functionalities:

- Energetic Control: The CyberModel class determines the suitable steering angles and velocities based on incoming commands. The model accommodates both linear and angular velocities, enabling smooth navigation.
- Odometry Publishing: The plugin regularly publishes odometry data for the front axles, allowing other nodes within the ROS ecosystem to accurately track the robot's position and movement.
- Change Broadcasting: A change broadcaster is employed to publish the robot's pose within the Gazebo environment, assisting integration with additional components and ensuring accurate representation of the robot's position in the ROS framework.
- Subscriber Callbacks: The plugin offers callbacks for handling drive and twist commands, allowing for adaptable control methods. The cyberdrive callback function modifies steering and velocity based on the received messages, while the twist callback function converts twist commands into drive messages.

Challenges:

- Coordinate System Issues: It was essential to make sure that the robot's coordinate system aligned correctly with the Gazebo environment. Careful adjustments were necessary, as misalignment initially resulted in discrepancies in movement and sensor readings.
- Realistic Dynamics: Accurately simulating vehicle dynamics was challenging, particularly in tuning the PID controllers for responsive and stable control. This required extensive testing and iteration to meet the desired performance.
- Integration with ROS: Effectively integrating the plugin with the ROS ecosystem demanded careful attention to detail, especially in managing message types and ensuring proper communication between components.

2.5 Control Modes

2.5.1 Teleoperations with Keyboard

In this section, we discuss the implementation of a teleoperation system that allows users to control the robot via keyboard inputs. The teleoperation node leverages the *pynput* library to capture keyboard events and *curses* for a user-friendly display of information. Users can control the robot's linear and angular velocities by pressing specific keys, which directly influence the robot's movement in the environment.

Key Features:

- Velocity Control: Users can adjust both linear and angular speeds. The default speeds are set to allow for manageable control, with options to increase or decrease these speeds by 10% using designated keys:
  - Linear Speed: Adjusted with 'Q' to increase and 'Z' to decrease.
  - Angular Speed: Adjusted with 'W' to increase and 'X' to decrease.
  - Reset Speed: Pressing 'R' resets speeds to their default values.
- Continuous vs. Non-Continuous Mode: The teleoperation supports both continuous movement and discrete commands:
  - In Continuous Mode, pressing a directional key will keep the robot moving until the key is released.
  - In Non-Continuous Mode, the robot will only move as long as the key is held down.
- User Interface: The system provides real-time feedback through a terminal window managed by curses. It displays the current velocities, control instructions, and a simple ASCII art representation of the robot, enhancing user experience and providing important status updates.

Control Logic:

- Keyboard Input Handling: The *on_press* and *on_release* methods manage user inputs, setting the appropriate linear and angular velocities based on key presses. Key mappings include:
  - Arrow Keys: Control movement (up for forward, down for backward, left for rotation left, right for rotation right).
  - 'S' Key: Stops all movement.
  - 'Caps Lock': Toggles between continuous and non-continuous control modes.
  - Publishing Velocity Commands: The system periodically publishes the current velocity commands to the */cmd_vel* topic using the *publish_twist method.* This ensures that the robot receives up-to-date commands based on the latest user inputs.

  System Information Display: The *print_system_information* method refreshes the display to show current control parameters and system status, helping users understand their control state.

This teleoperation setup is ideal for scenarios requiring manual control, providing users with a responsive and engaging interface to navigate the robot effectively.

2.5.2 Teleoperations with Joystick

This teleoperation system enables users to control a robot using joystick inputs, offering two distinct control schemes to cater to different preferences. The design emphasizes intuitive operation and real-time feedback, allowing users to maneuver the robot effectively in various environments. Each control scheme includes unique button mappings and movement mechanics, ensuring flexibility and comfort for operators.

Traditional Control Scheme

- Movement Control:
  - Forward/backward with L2/R2 triggers.
  - Turning with the left joystick (left/right).
- Button Functions:
  - Triangle: Increase angular velocity.
  - Circle: Decrease linear velocity.
  - Cross: Decrease angular velocity.
  - Square: Increase linear velocity.
  - L1: Stop motion.
  - R1: Reset speeds.
  - R3: Toggle continuous mode.
- User Feedback: Real-time updates on velocities, mode, and control instructions displayed on the terminal.

Alternate Control Scheme

- Inverted Control:
  - Forward/backward movement is inverted (up for backward, down for forward).
  - Left joystick controls turning left/right.
- Button Functions:
  - Circle: Reset speed.
  - Cross: Stop motion.
  - Square: Toggle continuous mode.
  - L1: Decrease angular velocity.
  - R1: Increase angular velocity.
  - L2: Decrease linear velocity.
  - R2: Increase linear velocity.
- User Feedback: Similar real-time updates as the primary scheme.

Both schemes allow dynamic adjustments to velocities and provide clear feedback, enhancing the usability and flexibility of the teleoperation system. This versatility makes the system suitable for various applications, from casual use to more demanding operational environments.

2.5.3 Simple Proportional Controller

In the autonomous mode, a proportional controller was implemented to guide the robot from the starting point A (0,0) to the target point B (10,10). Another set of target point C (0, 0) is also provided to return the robot back to the origin. The controller operates by calculating the difference between the robot's current position and the desired goal, using this error to adjust the robot's velocity commands in real-time.

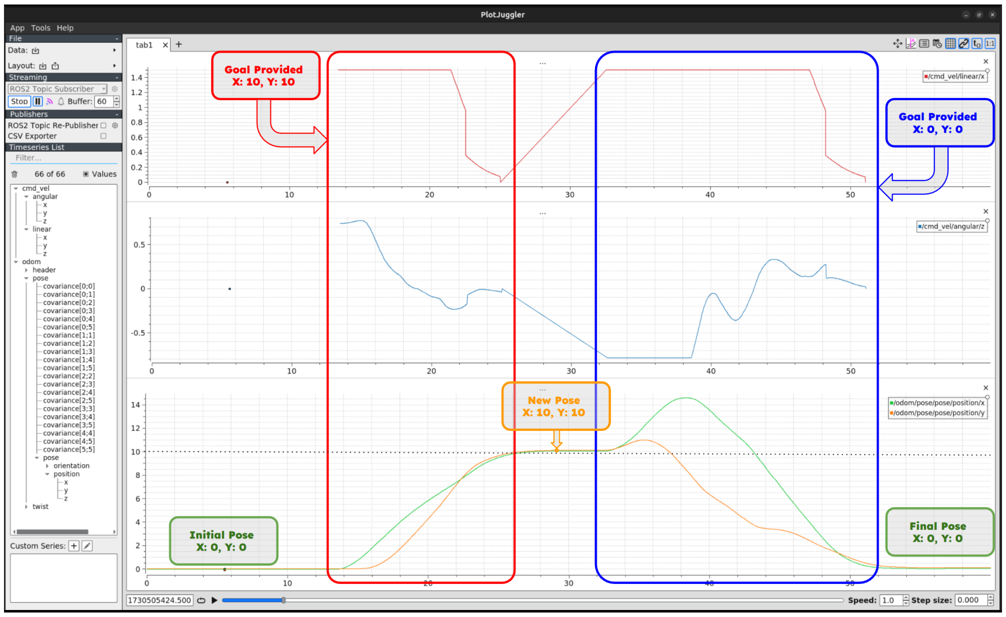

Figure 8: Pose plot of robot trajectory in autonomous mode

Controller Design The controller is designed to balance two key elements: linear and angular motion. The controller uses different proportional gains for coarse and fine adjustments, which allows it to respond appropriately based on the robot's distance from the goal:

- Coarse Control: Applied when the robot is far from the target (at least 6 times the goal threshold), utilizing higher gains to enable quicker corrections.
- Fine Control: Engaged when the robot is closer to the goal, applying lower gains to allow for more precise movements and prevent overshooting.

The proportional controller continuously updates the robot's velocity commands based on the current odometry data. When a new goal is set through a service call, the robot begins to calculate the necessary adjustments. The following steps summarize the control logic:

- Error Calculation: The position errors in the x and y directions are computed. If the errors fall within a defined threshold, the robot stops, indicating that the goal has been reached.
- Velocity Command Adjustment:
  - Linear Velocity: Calculated based on the distance to the goal. If the robot is far from the target, it moves quickly; if it is close, it slows down to ensure accuracy.
  - Angular Velocity: Based on the difference between the desired heading (toward the goal) and the current heading of the robot. Similar to linear velocity, this is adjusted according to the robot's proximity to the goal.
- Publishing Commands: The calculated linear and angular velocities are published as Twist messages to command the robot's movement.

Challenges with Controller Tuning

- Tuning the proportional controller for the robot's navigation involved several challenges, particularly in finding the appropriate gain values for linear and angular motions. This process required extensive experimentation to ensure effective performance across varying distances to the target points.
- Gain Value Calibration: Initially, the controller's response was overly aggressive, causing the robot to oscillate around the target rather than smoothly approaching it. This necessitated a careful adjustment of the proportional gains. We experimented with different values, seeking a balance that allowed the robot to react quickly without overshooting its target.
- Dynamic Response: Determining when to switch between coarse and fine control proved challenging. The thresholds for distance at which these modes would activate needed to be finely tuned to ensure that the robot did not slow down too early or remain too fast when still far from the target.
- Testing Different Scenarios: We conducted tests in various scenarios, adjusting the gains in response to the robot's performance during different trajectories. This included navigating tight corners and recovering from overshoots, which required iterative refinement of both the linear and angular gain values.

3 Results

Overall, the project achieved its objectives, successfully developing a functional Cyber Truck and Trailer model with integrated LIDAR capabilities and effective teleoperation controls. The proportional controller demonstrated reliable performance in navigating the robot autonomously. Future work could focus on enhancing the controller's adaptability and exploring advanced control strategies to further improve navigation accuracy in dynamic environments.

4 Links to Videos and GitHub Repository

- **Teleoperation Video:** Watch Video
- **Autonomous Mode Video:** Watch Video
- **GitHub Repository:** Repository

References

1. Zhe Leng, and Mark Minor. "A simple tractor-trailer backing control law for path following." *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2010, pp. 5538–5542, `https://doi.org/10.1109/iros.2010.5650489`.
2. Werling, Moritz, et al. "Reversing the general one-trailer system: Asymptotic curvature stabilization and path tracking." *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, Apr. 2014, pp. 627–636, `https://doi.org/10.1109/tits.2013.2285602`.