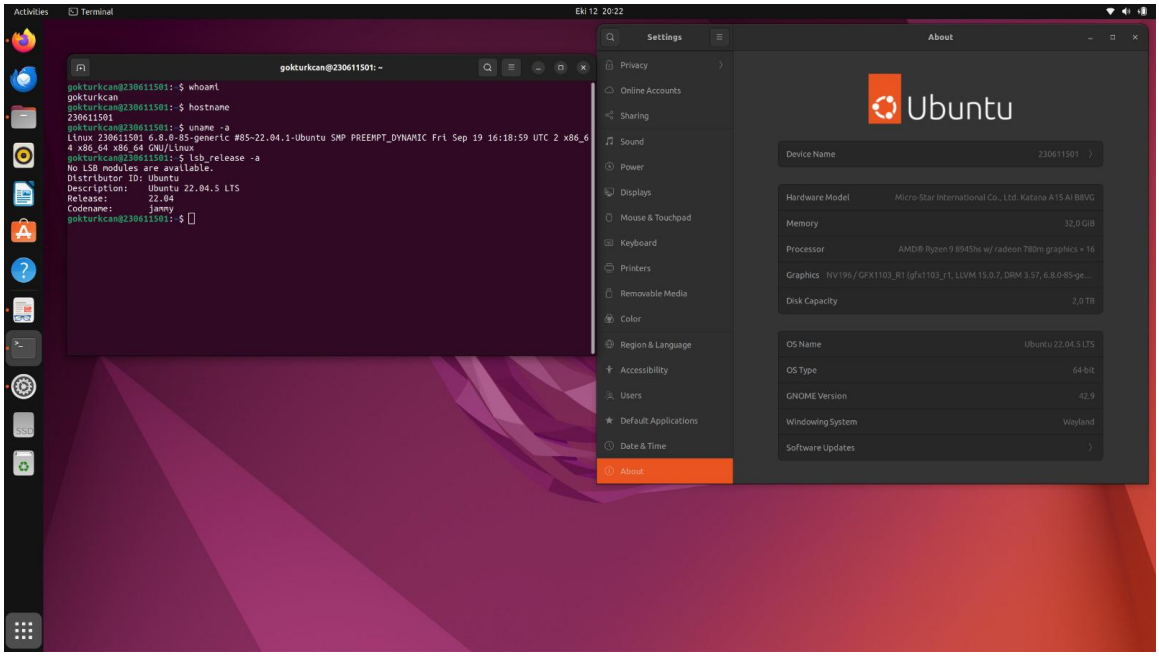
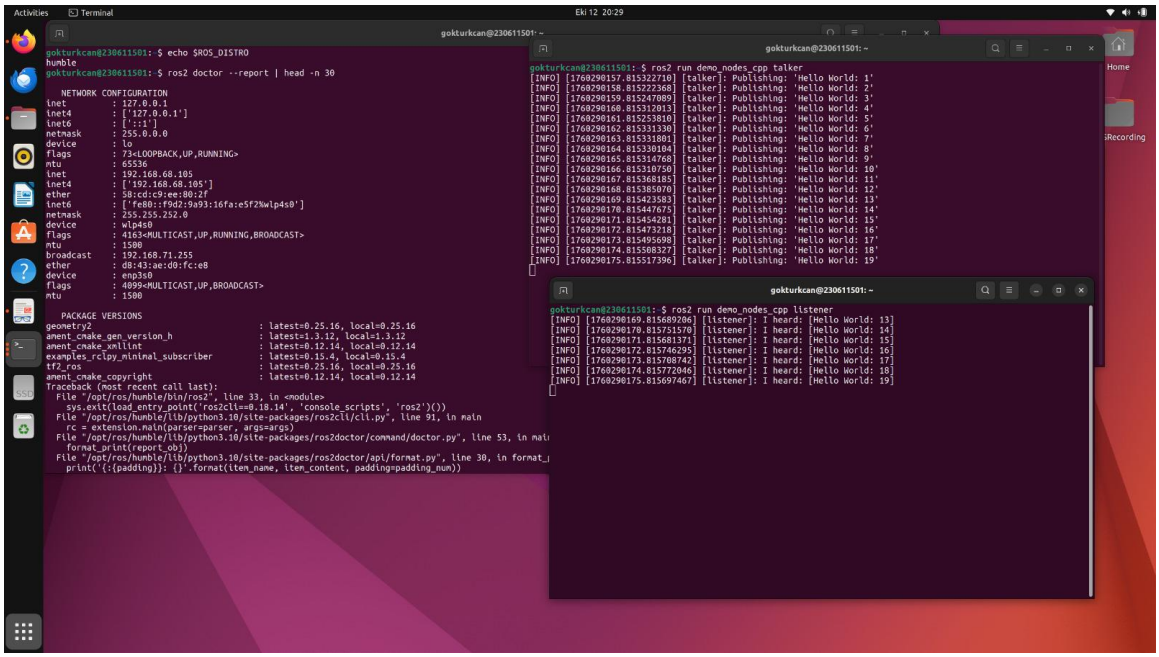
Assignment 2 Report — Build Your Own Robot (Diff Drive + Camera + TF/URDF + Ball Chaser)

Name Surname: Göktürk Can
Student No: 230611501
Course: CSE412 – Robotics (Fall 2025)
Programming Language: Python
Submission Date: 31 October 2025

1. System Information

The development and testing were performed on a computer running Ubuntu 22.04 LTS, equipped with ROS 2 Humble Hawksbill and Gazebo Harmonic.
The ROS 2 Humble distribution provides middleware abstraction via DDS (Data Distribution Service), ensuring real-time message passing and deterministic control behavior — a crucial aspect for robot motion and perception synchronization.
Gazebo Harmonic, in combination with the ROS-Gazebo bridge, simulates real-world physics and sensor feedback (IMU, camera, joint state).
Hardware specifications and system environment are verified in the demonstration video, including terminal outputs from 'hostnamectl', 'printenv | grep ROS_DISTRO', and 'gz version'.



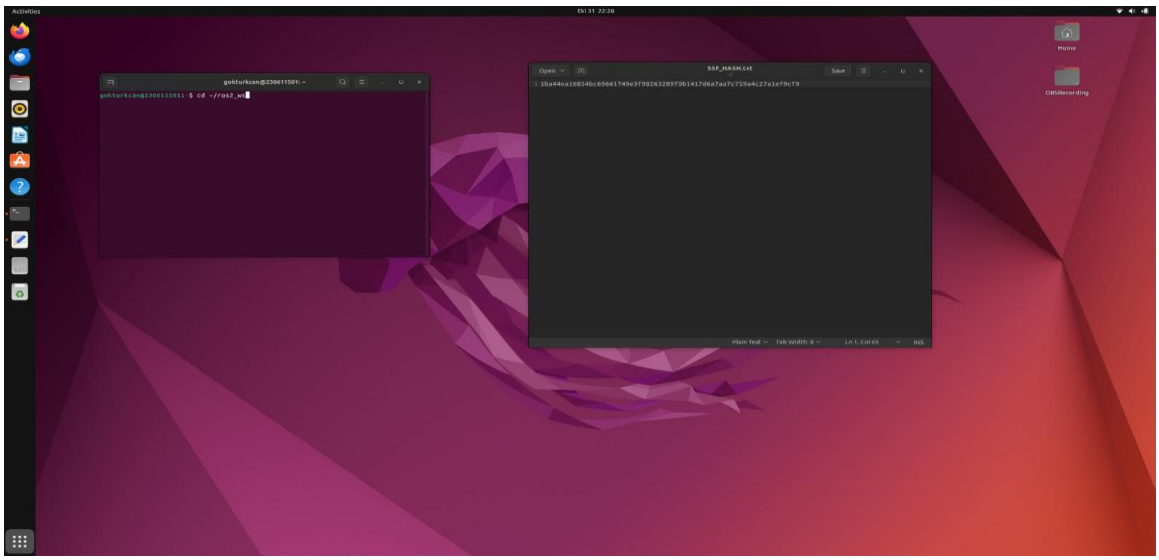


2. SSF Output

Before launching the system, the provided ssf.sh script was executed to generate the submission hash (RAW + SHA256).

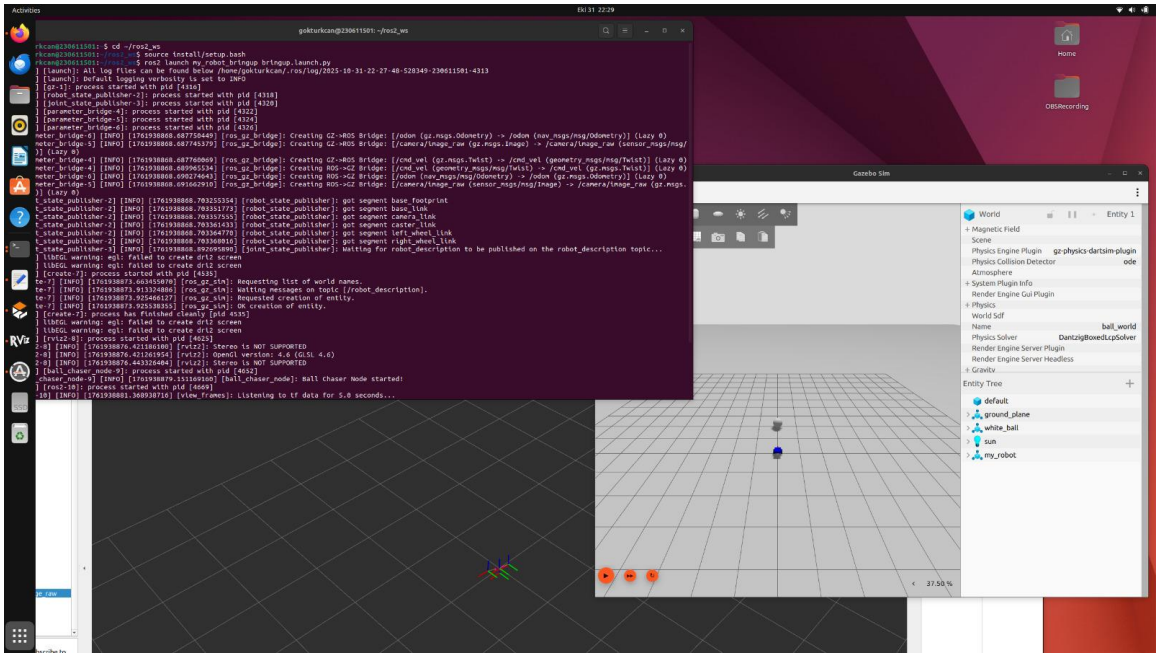
This ensures the authenticity and traceability of the project submission. The output was saved to 'SSF_HASH.txt' located in the project's root directory (~/ros2_ws/SSF_HASH.txt).

In the video, the complete terminal log is displayed prior to starting ROS 2 launch sequences. This serves as both version and identity validation according to assignment requirements.



3. Robot Model and URDF

The robot model was constructed from first principles using URDF/Xacro, defining a hierarchical chain of links and joints.
The kinematic structure follows the required convention: world → odom → base_link → (left/right/caster)_link → camera_link.
Each wheel uses a continuous joint type, while the caster wheel is fixed, allowing free rolling without torque transmission.
The robot_state_publisher and joint_state_publisher nodes were used to broadcast transforms (TF) in real time.
This modular URDF design enables straightforward TF tree visualization and consistent frame alignment for localization tasks.

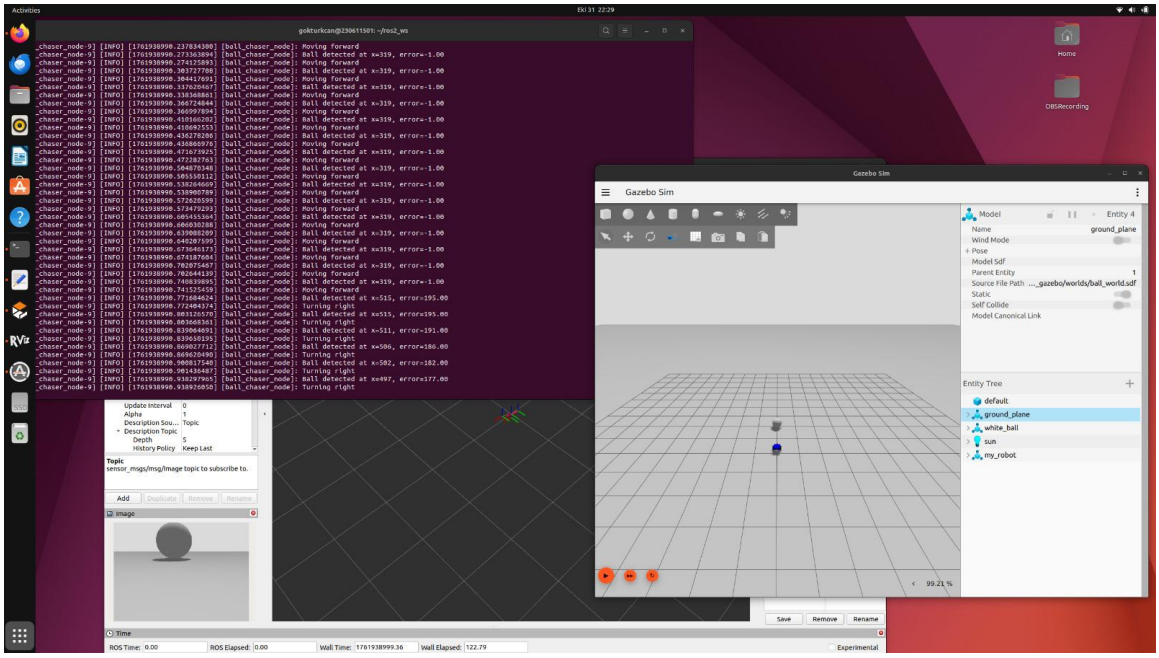


4. Diff Drive Integration

Differential drive motion was implemented via the Gazebo diff_drive plugin, which computes wheel velocities based on linear and angular commands from the /cmd_vel topic.
The plugin reads parameters such as wheel separation, wheel radius, and update rate, publishing corresponding joint states for Gazebo physics simulation.
This integration provides a foundational control architecture analogous to real mobile robots, where velocity commands (Twist messages) are transformed into differential angular velocities for each wheel.

5. Camera and RViz Verification

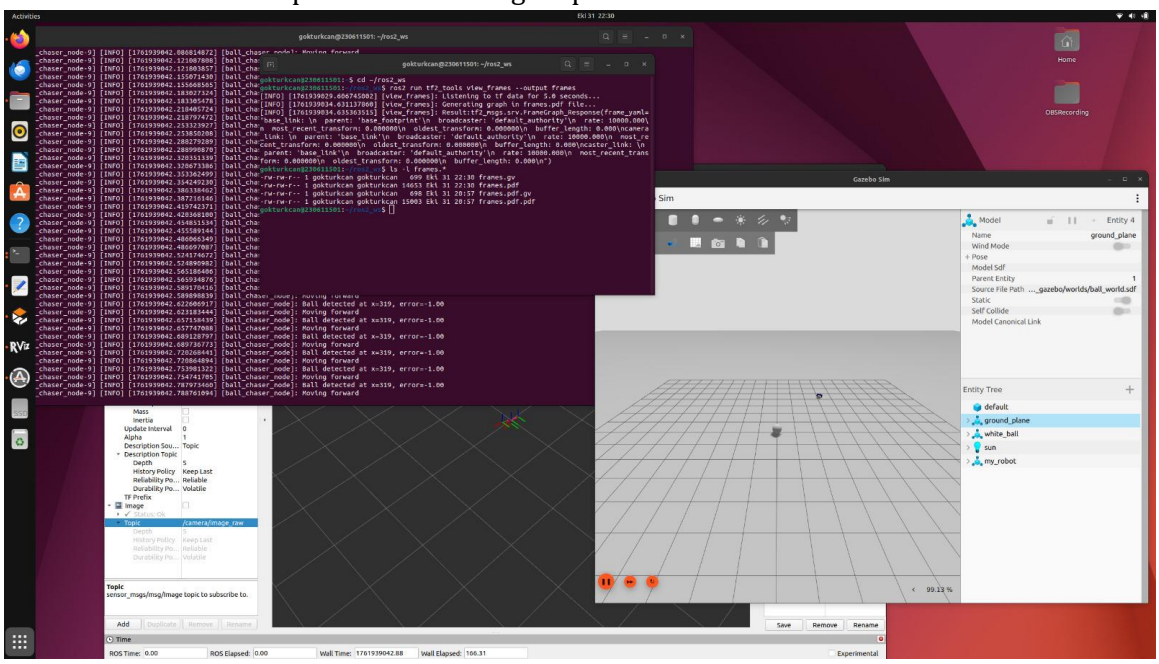
A simulated camera sensor was added to the URDF and bridged to ROS 2 through ros_gz_bridge, producing sensor_msgs/Image messages on the /camera/image_raw topic. RViz visualization confirmed correct sensor alignment and streaming frequency. The TF Tree and robot model were jointly visualized, validating all spatial transformations.
The command 'ros2 run tf2_tools view_frames --output frames.pdf' generated a graphical representation of the TF relationships, which was also displayed and verified in the video.



6. Ball Chaser (Behavior Analysis)

The Ball Chaser node was implemented in Python using rclpy. It subscribes to /camera/image_raw, processes image frames via OpenCV, and publishes geometry_msgs/Twist messages to /cmd_vel.
The algorithm segments the white ball through color thresholding in HSV space and calculates its centroid position. Based on the centroid's horizontal offset from the image center, the node adjusts the robot's angular velocity to recenter the target.
If the ball is not detected, both linear and angular velocities are set to zero. This control

scheme emulates a simple visual servoing loop.



7. Experiments and Results

Two main experimental scenarios were evaluated: (1) when the ball is visible, and (2) when the ball disappears.

In the visible case, the robot continuously tracks and approaches the ball until it is centered in the frame. In the absence of the ball, the robot stops immediately, demonstrating robust conditional control logic.

Performance remained stable without oscillations, and the visual feedback latency was negligible, confirming successful real-time operation of the ROS 2-Gazebo ecosystem.

8. Launch Procedure

All system components are launched through a single ROS 2 launch file: ros2 launch my_robot_bringup bringup.launch.py.

This file initializes the Gazebo world (robot + white ball), starts robot_state_publisher, joint_state_publisher, RViz, bridges, and the ball_chaser node in a coordinated sequence.

The single-command architecture ensures reproducibility and simplicity of deployment. All concurrent processes were shown functioning together in the demonstration video.

9. Conclusion and Evaluation

This project achieved full integration of robot modeling, TF broadcasting, camera perception, and autonomous ball chasing under ROS 2 Humble.

The system demonstrates modular design, real-time responsiveness, and accurate motion behavior.

The results validate the effectiveness of ROS 2 middleware and Gazebo Harmonic

simulation for robotics education.
Future work could extend to implementing PID controllers for smoother velocity control and testing on a physical differential-drive platform.

10. Links

GitHub Repository: https://github.com/GokturkCan/CSE412_Assignment2_GokturkCan
YouTube (Unlisted): https://www.youtube.com/watch?v=lx7NfIekOpU
All system verifications, outputs, and experiments are demonstrated within the accompanying video.