ATC Signs Detection for Autonomous Vehicles in Roadwork Scenarios

Carlos Rijo[1101626], João Fernandes[1221973], and Mário Carneiro[1170699]

Instituto Superior de Engenharia do Porto, Portugal

Abstract. This report complements the previous delivery by explaining the implementations made and by analyzing both the results obtained and the system limitations.

Keywords: Autonomous Vehicles · Roadwork · ROS2 · OpenCV · Computer Vision · Gazebo · ATC Signs.

1 Implementation Overview

During the development of this system, the architecture, proposed previously, was slightly changed. These modifications can be seen in the updated system architecture design, in figure 1.

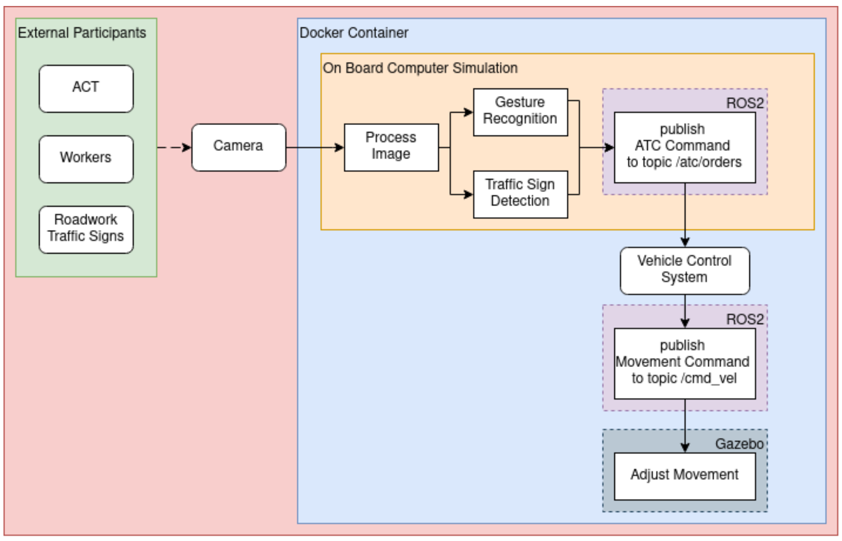


Fig. 1: Updated System Architecture Diagram

Instead of running the system in Autoware, it was decided to run everything in a docker container with the Linux Ubuntu 20.04 operating system as its base. This change helped centralize the individual work and keep the implementation, as a whole, updated.

As for the second change, it was implemented a second `ROS 2` topic called `/cmd_vel` with a movement command. This way, after detecting the ATC order and processing it, a topic with the ATC order (`/atc/order`) is sent to the Vehicle Control System that, after a decision-making process, sends movement command, with linear and angular velocities, thought the topic `/cmd_vel` to the Vehicle Actuators and thus adjust the vehicle movement.

The ATC hand gesture orders implemented:

- Stop - `/atc/stop`
- Forward - `/atc/forward`
- Turn Left - `/atc/turn left`
- Turn Right - `/atc/turn right`
- Slow Down - `/atc/slow down`
- Invert - `/atc/invert`

The code developed in this implementation is stored in a GitHub repository: `https://github.com/carlosrijo9/MESCC_IACOS/tree/main`. This repository also has the result videos and a README.md file with instructions to prepare the Docker Container, compile our setup and run our implementation.

1.1 OpenCV

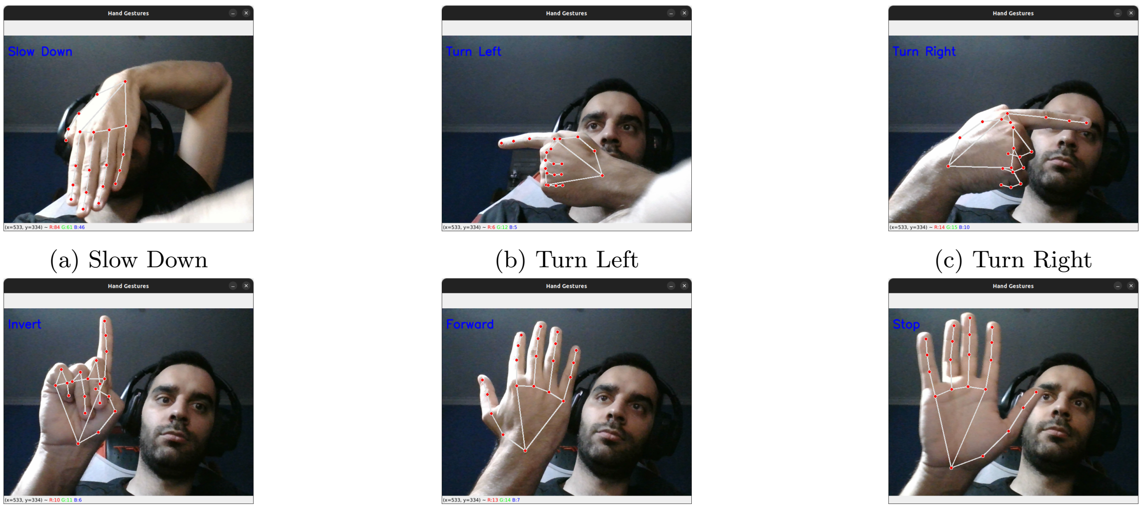
The OpenCV implementation for hand gesture detection utilizes the MediaPipe library for hand tracking and gesture recognition.

Initially, video frames from the webcam are captured and processed to detect hands using the MediaPipe hands model. A function calculates hand orientation based on detected landmarks, ensuring accurate gesture recognition.

To refine the gesture recognition process, the system identifies whether the detected hand is left or right using the `identify_hand_type` function, which is crucial for interpreting certain gestures correctly, such as determining the direction (left or right).

The `recognize_gesture` function interprets hand landmarks, orientation, and hand type to recognize specific gestures. The implemented gestures include commands for stopping, moving forward, turning left, turning right, slowing down, and inverting the vehicle's movement. The recognized gestures are then displayed on the video feed using OpenCV's drawing functionalities.

Finally, the `imshow` function in OpenCV is employed to present everything on the video screen, as it is possible to see in figures 2. We also display the messages received in our ros2 middleware node before we send them to Gazebo.



(a) Slow Down (b) Turn Left (c) Turn Right

(d) Invert (e) Forward (f) Stop

Fig. 2: Compilation of Hand Gestures

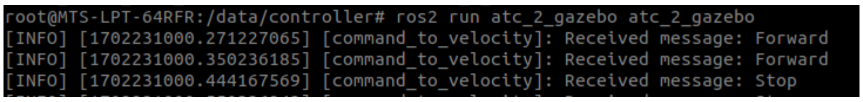
1.2 ROS 2 Communication

The ROS 2 system facilitates communication between the gesture detection module and the vehicle actuation in the simulation. A publisher node sends recognized gesture commands, and a subscriber/publisher node relays this information for Gazebo to control the vehicle. We based our implementations for ros2 publisher/subscriber nodes from the PL lectures and the documentation in [7].

Hand Gestures Node: Detects gestures using the webcam and publishes commands on the `/atc` topic.

Middleware Node: Subscribes to the `/atc` topic and publishes control commands via the `/cmd_vel` topic to actuate the vehicle in Gazebo.

Result: ROS2 integration enabled real-time control of the vehicle in the simulation, demonstrating an efficient method for processing autonomous vehicle commands (shown in figure 3).



```
root@MTS-LPT-64RFR:/data/controller# ros2 run atc_2_gazebo atc_2_gazebo
[INFO] [1702231000.271227065] [command_to_velocity]: Received message: Forward
[INFO] [1702231000.350236185] [command_to_velocity]: Received message: Forward
[INFO] [1702231000.444167569] [command_to_velocity]: Received message: Stop
```

Fig. 3: Message received from ROS2 topic `/atc`

1.3 Gazebo Simulation

The simulation in Gazebo was divided into two parts: modulation of a vehicle and modulation of a world.

The vehicle modulation was based on a GitHub project[6]. After preparing a workspace, we start by updating the vehicle URDF file, that describes the vehicle (`robot.urdf.xacro`). The configurations on this file will be passed to `robot_state_publisher` making the data available on the `/robot_description` topic.

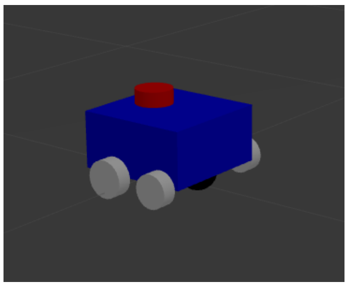
Then we created a new file called robot_core.xacro mainly with Chassis, two Driving wheels, collision and inertia parameters.

Finally, it was created a new xacro file called gazebo_control.xacro with references to both wheel joints and that includes the control plugin, `libgazebo_ros_diff_drive.so`, responsible that will interact with the core Gazebo code which simulates the motors and physics.

With ROS2 topic called /cmd_vel, the Gazebo simulation can get the linear speed in x (driving forward or backward), and angular speed in z (turning left or right).

For the world modulation, we simply added a road that works as a reference to the vehicle movement. Not directly related to the simulation. but helpful during the development, it was added a file called `launch_sim.launch.py` that launches the Gazebo simulation with all parameters and necessary flags.

Result: As it is possible to see in figure 4a and 4b it was possible to create a vehicle module and a world module to simulate the vehicle's movement and the adjustment according to the ATC order.


(a) Vehicle Simulation

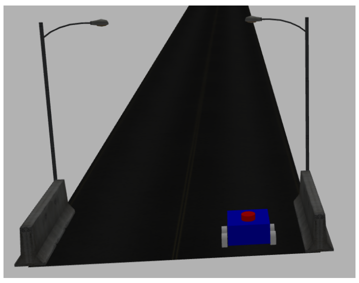

(b) World Simulation

Fig. 4: Simulation in Gazebo

2 Conclusion

The developed system can perform the tasks intended: detect an ATC sign, publish the orders, through a ROS2 topic, and adjust the vehicle movement accordingly.

Even so, some aspects can be improved and some limitations that need to be addressed. In terms of limitations, the trained model set still lacks accuracy in detecting hand gestures and is not trained to detect all the hand gestures defined in the law. The implemented control of the vehicle (linear and angular movement) is not very accurate in comparison to the real movement of a car. And the Gazebo world is almost empty and far from a real scenario, making it unsuitable for testing and simulation.

As for improvements, the model set should be trained with more images and add more hand gestures and road signs to make the vehicle follow all road rules. The control of the vehicle movement should be analyzed in depth to copy the movement of a real car and follow the physics rules. The simulation in Gazebo should improve the vehicle model to simulate a real vehicle (with real size, weight, center-of-mass, etc), should improve the world scenario by adding more roads and obstacles (houses, pedestrians, other vehicles, road work objects, etc) and should integrate more sensors like Radar, Lidar, and Cameras to perform obstacle detection and avoidance actions.

These improvements would increase the simulation's resemblance to a real scenario and ensure that the vehicle follows all road rules.

References

1. OpenCV Documentation Homepage, `https://docs.opencv.org/4.x/d1/dfb/intro.html`, Last access: November 15, 2023
2. Author, Bradski, G., & Kaehler, A.: Learning OpenCV: Computer vision with the OpenCV library.
3. Author, Steven Macenski and Tully Foote and Brian Gerkey and Chris Lalancette and William Woodall: Robot Operating System 2: Design, architecture, and uses in the wild, `https://www.science.org/doi/10.1126/scirobotics.abm6074`, Last access: November 16, 2023
4. GAZEBO Homepage, `https://gazebosim.org/about`, Last access: November 15, 2023
5. GAZEBO Tutorial, `https://classic.gazebosim.org/`, Last access: December 08, 2023
6. Josh Newans Github Repository, `https://github.com/joshnewans/articubot\_one/tree/545acac87ae215d80ef6b28abe6097eb7281d9ff`, Last access: December 08, 2023
7. ROS2 Foxy Repository, `https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html`, Last access: December 08, 2023