Authorized Traffic Controlle Sign Detection System for Autonomous Vehicles

Pedro Barbosa 1231922, Rúben Seabra 1181865, and Tiago Costa 1201329

Instituto Superior de Engenharia do Porto

Abstract. Navigating complex environments such as roadwork zones remains a significant challenge for autonomous vehicles due to the dynamic and unforeseen events of these environments. This project explores the development of a ROS2 node capable of processing and classifying basic Authorized Traffic Controllers (ATCs) signs that could be used in such situations. The system integrates ROS2, computer vision and AI as its key technologies.

Keywords: autonomous vehicle · ROS2 · Computer vision · OpenCV · Gazebo.

1 Introduction

Autonomous vehicles (AVs) made significant progress in recent years, with many advancements in sensing, mapping, and decision-making technologies, allowing them to navigate complex environments with increasingly less human intervention. However, despite this recent progress, scenarios such as roadwork zones remain a challenge for AVs. These environments are characterized by unpredictable and temporary layouts that overlap the standard infrastructure, with lane markings and traffic signals replaced by temporary signals, cones, barriers, and authorized traffic controllers (ATCs) [1]. These dynamic conditions can be easily interpreted by humans, but for AVs, these new elements can lead to navigation failures if they aren't correctly handled.

Advancements in AI, and computer vision more particularly in gesture recognition [2], could enable AVs to better interpret non-standardized, human-controlled signals in roadwork zones.

This project develops a system for real-time interpretation of ATC gestures using ROS2 and computer vision. The gesture recognition node classifies commands like stop, forward, left, and right. A control node then calculates the vehicle's velocities and the Gazebo simulator adjusts the TurtleBot3's trajectory in real-time based on these commands.

2 System architecture

The system architecture (Figure 1) follows a sequential process. It starts with real-time image capturing through a camera, followed by processing and interpretation of these images through OpenCV [7] and MediaPipe, to identify the gestures. After the identification, ROS node publishes the corresponding gesture to the "atc/orders" topic. Subscribed to this topic is the control node that for each gesture publishes the corresponding command in the "cmd_vel" topic with TurtleBot3 being subscribed to this topic and adjusting its trajectory based on the recognized gestures.

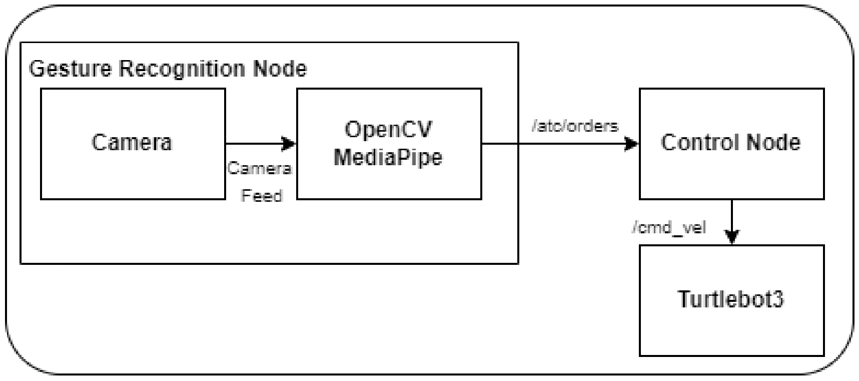

Fig. 1. System architecture

Components:

– Gesture Recognition Node: This node processes the video input using AI or OpenCV techniques to detect and classify gestures. Once a gesture is recognized, it publishes the corresponding sign to the topic /atc/orders.
– Control Node: This node subscribes to the /atc/orders. Based on the recognized sign, it calculates the appropriate linear and angular velocities for the vehicle. These commands are then published to the /cmd_vel topic in the form of a Twist message.
– Gazebo: The simulator is subscribed to the /cmd_vel topic. It adjusts the trajectory of the TurtleBot3 in real-time according to the velocity commands received from the Control Node.

3 Environment setup

The environment used includes the Ubuntu 22.04.5 LTS (Jammy Jellyfish) operating system, together with a ROS2 Humble distribution and Gazebo 11. To initiate the simulation, the following command was executed:

```
./launch_simulation
```

The command launches the simulation environment for the TurtleBot3 [9] using waffle model and the empty world examples.

The ROS2 node with the gesture recognition can be launched with the following command:

```
./launch_gestureRecognition
```

The command sets the Python environment necessary for the node to function, and starts the ROS2 node.

The Control node can be launched with the following command:

```
./launch_gestureToTwist
```

The command sets the Python environment necessary for the node to function, and starts the ROS2 node.

4 Gesture Recognition

The process begins with analyzing a real-time camera feed, which serves as the primary input. Using OpenCV, the video stream is divided into manageable image frames, forming the basis for further processing. From here MediaPipe [8] framework is used to identify 21 key landmarks on the hand, providing precise coordinates for gesture analysis.

4.1 Landmark Extraction and Normalization

For each gesture the respective landmarks coordinates were extracted and normalized in relation to the wrist. This normalization ensures consistency across different hand positions and orientations, allowing the system to generalize effectively. These processed landmark coordinates were then used to construct a labeled dataset to train a neural network to classify the gestures.

4.2 Gesture Classification

The classifier was trained to recognize four key gestures required for roadwork scenarios:

– Forward: The hand is closed, with all fingers flexed and touching the palm, forming a tight fist.
– Stop: The hand is open, the palm is extended with all fingers fully spread apart.
– Left: The hand is pointing to the left, with the index finger extended.
– Right: The hand is pointing to the right, with the index finger extended.

The use of an AI-based approach for classification significantly simplifies the implementation, as it avoids the need for explicit logic to analyze angles or distances between landmarks. This modularity allows easy adaptation, with the addition of new gestures only requiring the extraction of landmarks to expand the dataset and retraining the classifier.

4.3 ROS2 Node Integration

The gesture recognition functionality was implemented in a ROS2 node. After the processing of the incoming camera feed and classifying the gesture, the node publishes a message with the gesture detected to the /atc/orders topic.

5 Results

The primary goal of the project was achieved successfully, the final implemented system effectively detects and classifies hand gestures while controlling the trajectory of a simulated vehicle. The neural network demonstrated accurate gesture recognition, enabling the Control Node to adjust TurtleBot3 trajectory in accordance with the identified gestures.

The AI approach used, enabled an agile employment of the detection capabilities, since we don't have to define logic for each gesture, with this approach desired gestures could easily be added. This agility is a notable advantage over traditional methods, which could rely on calculating distances and angles between landmarks.

However, the gesture recognition system has some limitations and challenges, such as hand occlusion or partial gestures, that occasionally confuses the model due to the limited gesture set. Additionally, variations in lighting conditions also led to occasional incorrect classifications. These limitations underscore the need for future improvements, including enhancing the model's robustness with larger training datasets and improving the camera.

References

1. Mishra A, Kim J, Cha J, Kim D, Kim S. Authorized Traffic Controller Hand Gesture Recognition for Situation-Aware Autonomous Driving. Sensors. 2021; 21(23):7914. https://doi.org/10.3390/s21237914
2. J. Wiederer, A. Bouazizi, U. Kressel and V. Belagiannis, "Traffic Control Gesture Recognition for Autonomous Vehicles," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 10676-10683, doi: 10.1109/IROS45743.2020.9341214.
3. W. Shi and R. R. Rajkumar, "Work Zone Detection For Autonomous Vehicles," 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 2021, pp. 1585-1591, doi: 10.1109/ITSC48978.2021.9565073.
4. Gupta, S., Vasardani, M., Winter, S.(n.d.). Conventionalized gestures for the interaction of people in traffic with autonomous vehicles. https://doi.org/10.1145/3003965.3003967
5. ROS: Home,https://ros.org/, last accessed 31/11/2024
6. Gazebo, https://gazebosim.org/home, last accessed 31/11/2024
7. OpenCv - Open Computer Vision Library, https://opencv.org/, last accessed 31/11/2024
8. MediaPipe, https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer
9. TurtleBot3, https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/