IACOS PROJECT

Gonçalo Moreira[1190709], Henrique Teixeira[1200883], and Rodrigo Moreira[1201268]

Instituto Superior de Engenharia do Porto, Portugal

Abstract. This report explores the challenge of enabling autonomous vehicles to interpret and respond to hand signals from Authorized Traffic Controllers (ATCs) in roadwork and traffic control settings. While human drivers can intuitively follow these temporary directions, autonomous systems require advanced perception and interpretation capabilities to adapt. Developing this functionality is essential for enhancing safety and seamless interaction in mixed-traffic environments where human controllers guide both human-driven and autonomous vehicles.

Keywords: Autonomous Vehicles · ATC · Gesture Recognition · Hand Gesture Recognition · ROS2.

1 Introduction

In various roadwork and traffic control situations, human Authorized Traffic Controllers (ATCs) use hand signals to direct vehicles. These signals can override standard traffic rules, ensuring safety and order during temporary disruptions. While human drivers understand these instructions, autonomous vehicles face challenges in interpreting such unstructured inputs. Allowing vehicles to perceive, interpret, and act upon ATC gestures is crucial for improving the adaptability and safety of autonomous systems in mixed-traffic environments where humans and machines must coexist. This report addresses one such challenge: enabling autonomous vehicles to understand and respond to gestures from ATCs to ensure safe and smooth navigation through these scenarios.

2 State of the Art

2.1 Gesture Recognition

Gesture recognition plays an important role in computer science [1] which refers to the process of interpreting human gestures through mathematical algorithms and computer vision techniques by the computer [1], [2]. It aims to create a natural interface between humans and machines, allowing devices to understand and react to various body movements, especially hand gestures [3], [4]. The entire process of hand gesture recognition is divided into three phases: hand detection, hand tracking, and recognition [5]. Modern gesture recognition systems often rely on Artificial Intelligence, such as machine learning and deep learning techniques [6] to accurately identify and classify gestures from images or video streams [7].

2.2 Autonomous Vehicles

Autonomous vehicles are equipped with advanced technologies that allow them to perceive their environment, make decisions, and navigate without human intervention. This capability is achieved through a combination of sensors such as LiDAR, cameras, radar, and GPS. The development of autonomous vehicles has been significantly advanced by the adoption of frameworks like the Robot Operating System (ROS) and its newer version, ROS2. These frameworks facilitate the integration of hardware and software components in a modular fashion, which is crucial for the complex needs of autonomous driving.

3 Existent Approaches

There are distinct approaches to implementing gesture recognition, with technologies that can be broadly classified into vision-based [8] and sensor-based [9] methods.

3.1 Vision-Based

Automotive Industry: Gesture recognition is widely used in vehicles, allowing drivers to control features like music, navigation, and temperature without looking away from the road [10].

Healthcare and Medical Operations: Gesture recognition systems facilitate touchless control over medical equipment, enhancing hygiene and ease of use. Surgeons, for instance, can manipulate on-screen visuals during surgery without physical contact [11].

3.2 Sensor-Based

Consumer Electronics and Smart Homes: Hand gestures can control various household devices, from TVs to lighting systems, allowing for seamless home automation. Advanced gesture-based remote controls simplify interactions by enabling users to manage multiple devices with natural hand motions [12] [13].

4 Technologies

4.1 ROS2

ROS2 is a powerful middleware [14] and it was created as an open-source software development toolkit for robotic applications [15], [16], [17]. ROS2's integrated and thoroughly tested messaging system utilizes a publish-subscribe architecture, which streamlines communication between distributed nodes [18], [19]. This method allows multiple nodes to transmit and receive messages without needing to be aware of each other's existence, fostering a more flexible and decoupled design that is particularly beneficial in complex robotic systems [19].

4.2 Autoware

Autoware is an open-source software platform designed for autonomous driving [20], [21], [22], [23]. It was built on the ROS framework to enable the commercial deployment of autonomous driving across a wide range of vehicles and applications. Autoware also supports a variety of sensors, including radars, LiDARs, and cameras, allowing it to perceive and interpret the vehicle's surroundings effectively. Additionally, it integrates seamlessly with simulation environments like Gazebo, which will be used in this project to simulate various scenarios and validate the autonomous vehicle's behaviour in response to ATC signals and commands.

4.3 OpenCV

The Open Source Computer Vision Library (OpenCV), is a powerful and widely utilised open-source library [24], [25], [26] that provides a comprehensive suite of tools for computer vision and machine learning [27], [28]. It also offers support for several programming languages, such as C++, Python, Java, and MATLAB, making it accessible to a diverse range of developers and researchers [29]. Coupled with AI techniques where needed, this approach enables accurate interpretation of gestures or signs that an ATC may use, translating them into commands for a vehicle. With OpenCV's high performance and cross-platform support, this system can integrate seamlessly with ROS2 to create a dynamic and responsive node within a simulation, enabling autonomous decision-making and vehicle control.

4.4 YOLO

YOLO (You Only Look Once) is a deep learning-based algorithm for real-time object detection that frames object detection as a single regression problem. Developed to optimize both detection speed and accuracy, YOLO differs from traditional approaches that rely on a sequence of regional proposals and subsequent classifications. By treating the entire detection process as a single convolutional neural network (CNN) pass, YOLO's architecture is highly efficient and performs detection significantly faster than multi-stage approaches [30], [31], [32].

4.5 Gazebo

Gazebo is a free set of open-source software frameworks designed to simplify the development of high-performance 3D robotic simulations compatible with all major platforms such as Linux, macOS, Windows, IOS and Android [33], [34]. Gazebo offers a comprehensive API for creating custom plugins and simulations, making it a highly adaptable platform for a variety of robotic applications [35]. Also, Gazebo is an excellent option when access to physical robotic hardware is limited or when there is a need to test multiple robots at the same time [36].

5 System Architecture

The system architecture is divided into four primary subsystems: Perception, Decision-Making, Control and Gazebo Simulation. Each subsystem plays a critical role in ensuring the vehicle can navigate safely and accurately in response to the ATC instructions.

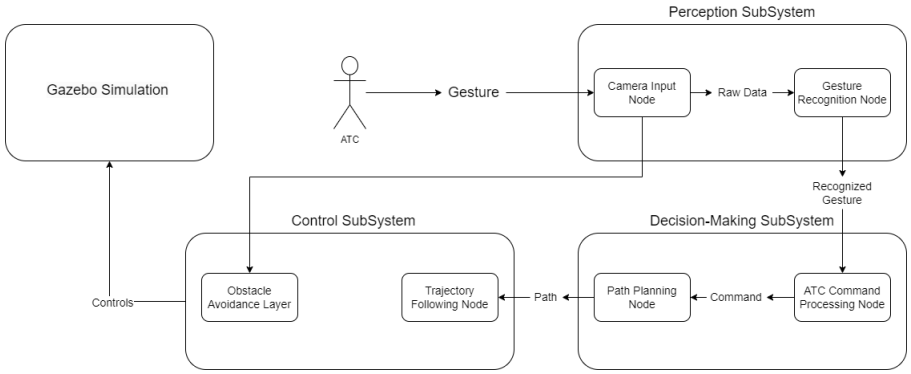

Fig. 1. System Architecture.

Furthermore, it is essential to emphasize that the overall system architecture, as currently designed, remains flexible and subject to future adjustments. Such changes may be necessary to enhance and optimize the system's performance as development progresses and additional insights are gained.

5.1 Perception Subsystem

The Perception Subsystem is designed to gather and interpret visual information from the surrounding environment, with a focus on detecting gestures from an Authorized Traffic Controller (ATC). It utilizes a camera to capture a continuous live video feed, which is subsequently processed to recognize hand gestures or other relevant signals. This subsystem is composed of two primary components, each playing a specific role in the detection and interpretation of these visual cues, enabling the system to respond effectively to ATC commands.

Camera Input Node: The camera input node is designed to capture a continuous video stream, which can come from various sources: a live camera feed, a simulated environment, or pre-recorded footage. This node processes the incoming frames and publishes the raw image data to a designated ROS2 topic, /camera/imageraw. The published raw data is then used as the primary input for gesture recognition modules, enabling real-time processing and interpretation of gestures captured within the video stream.

Gesture Recognition Node: The gesture recognition node leverages computer vision techniques, using OpenCV, to analyze the camera feed and identify specific gestures made by an Authorized Traffic Controller (ATC). The system is trained to recognize key commands, such as "stop," "go," and directional gestures like "turn left" or "turn right." Upon detecting a gesture, the node publishes the gesture information on the ROS2 topic /atc/gesture, including both the type of signal detected and a confidence score. This data is then made available for downstream processing, allowing other system components to respond appropriately to the recognized ATC instructions.

5.2 Decision-Making Subsystem

After the Perception Subsystem successfully identifies a gesture from the Authorized Traffic Controller (ATC), the Decision-Making Subsystem takes over to interpret this information and decide on the most appropriate course of action. This subsystem plays a critical role, as it integrates data from various sensory inputs, not just from gesture recognition, to build a comprehensive understanding of the current environment and context. The Decision-Making Subsystem prioritizes received commands, ensuring that the most immediate and important instructions are addressed first, while also making high-level navigation decisions based on the recognized gestures. This coordinated approach enables the system to respond accurately and efficiently to ATC signals, ensuring safe and compliant navigation in dynamic settings.

ATC Command Processing Node: This node subscribes to the /atc/gesture topic, where it receives and processes each incoming gesture signal. For every gesture, it interprets the gesture type—such as "Stop," "Go," or directional gestures like "Turn Left", to determine the most appropriate response. To maintain safety and adhere to traffic control guidelines, the Decision-Making Subsystem follows a set of predefined prioritization rules, ensuring that certain commands, like "Stop," are given precedence over others, such as "Go." Once the correct command is selected, it is published as an actionable output on the /atc/orders topic, allowing the system to execute the intended response effectively and safely.

Path Planning Node: The path planning node is responsible for transforming the decision-making output into a concrete, navigable trajectory for the vehicle. By subscribing to the /atc/orders topic, it receives directives based on the ATC's commands and uses this information to generate a safe and efficient route. This node takes into account various environmental factors, such as the presence of obstacles or restricted areas, to ensure that the planned path is both feasible and compliant with the ATC's instructions. Once the trajectory is calculated, the node publishes the detailed path, including precise waypoints, to the /plannedpath topic. These waypoints serve as guidance for the vehicle to accurately follow the designated route.

5.3 Control Subsystem

The Control Subsystem is responsible for executing the planned trajectory and adjusting the vehicle's path and speed according to the planned instructions. This subsystem receives the path plan and translates it into motion commands, ensuring that the vehicle navigates smoothly through the environment.

Trajectory Following Node: The trajectory following node is tasked with ensuring that the vehicle accurately adheres to the designated route. By subscribing to the planned path published on the /plannedpath topic, this node continuously adjusts the vehicle's speed and steering to maintain alignment with the specified trajectory. Operating within the Gazebo simulation environment, the node directly interfaces with the vehicle model, enabling it to implement real-time control adjustments. Through this connection, the trajectory following node can seamlessly translate the planned route into precise movements, ensuring smooth and responsive navigation along the path defined by the path planning node.

Obstacle Avoidance Layer: To enhance safety, the system incorporates an obstacle avoidance layer that continuously monitors the camera feed for any unexpected obstacles that might appear along the vehicle's planned route. This layer adds an extra level of adaptability by allowing the system to make real-time adjustments to the vehicle's path, ensuring safe navigation even in dynamic or uncertain environments. If an unplanned obstacle is detected, the obstacle avoidance layer can initiate immediate corrective actions, such as dynamically rerouting the vehicle or executing an emergency stop. This proactive approach significantly boosts the system's robustness, allowing it to respond effectively to sudden changes or unforeseen hazards in the vehicle's surroundings.

6 Code

All the code developed to achieve the final objective was developed in Python, as ROS2 supports this language and the learning curve is easier than C++, which was an advantage considering the deadline of the project. There are two main files in the project that execute the core functions of the system, the "sign_detector.py" and the "vehicle_controller.py". The first one is responsible for detecting the gesture made by the ATC towards the vehicle, analyze it, and then post the command to the topic "/atc/orders", whereas the latter is responsible for subscribing to this same topic and control the car given the commands received.

6.1 sign_detector.py

This project integrates ROS2, OpenCV, and MediaPipe to implement a real-time gesture recognition system. The system detects predefined gestures using

a webcam and publishes the corresponding commands to a ROS 2 topic. The gestures are classified into commands such as "turn_left," "turn_right," "stop," and "advance," which are used to instruct the vehicle control system on how to act upon encountering an ATC. This code consist of three main components, the MediaPipe Pose Detector, the Gesture Classification, and the ROS2 Node Publishing.

The classify_posture function extracts specific body landmarks and evaluates their relative positions to classify gestures:

– **Advance**: Both wrists are below their respective elbows signaling a neutral or forward movement.

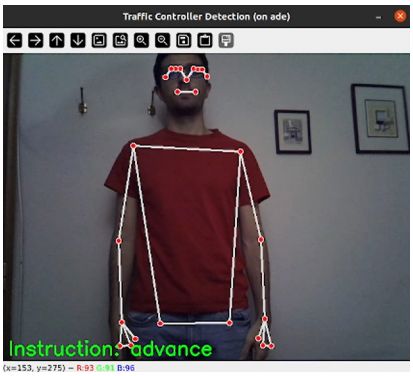

Fig. 2. Advance.

– **Stop**: Both wrists are above their respective elbows with open palms indicating a stop signal.

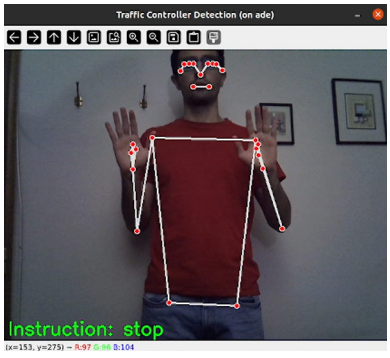

Fig. 3. Stop.

– **Turn Left**: The left wrist is below the left shoulder and the right wrist is positioned to the left of the left wrist.

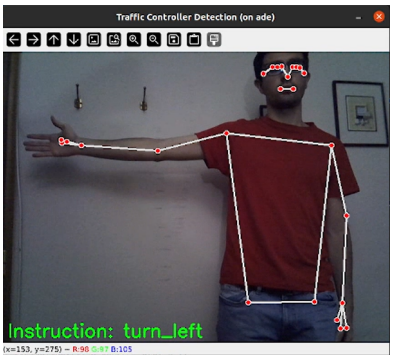

Fig. 4. Turn left.

– **Turn Right**: The right wrist is below the right shoulder and the left wrist is positioned to the right of the right wrist.

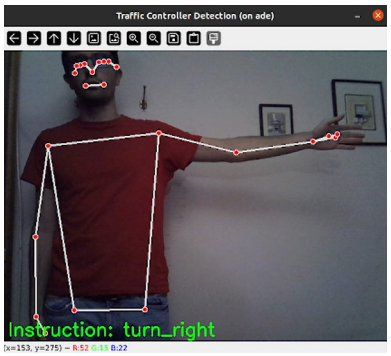

Fig. 5. Turn right.

– **Unknown or No Detection**: If the detected pose does not match any predefined conditions. In this example there's no detection of the predefined conditions.

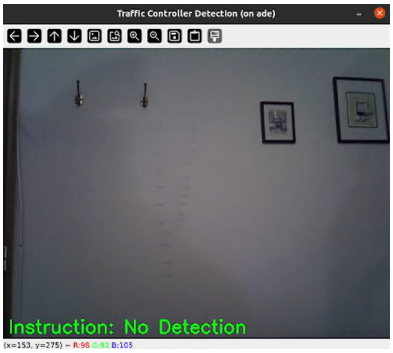

Fig. 6. Unknown or No Detection.

Then the SignDetector class implements the initialization of a node named "sign_detector" and a publisher on the topic "/atc/orders" using the message type "std_msgs.msg.String", and finally sets up a timer to process video frames and detect gestures at a frequency of 10 Hz.

The system then captures the video frames using OpenCV from the webcam and preprocesses the frame (resizing and color conversion) for MediaPipe compatibility. After this it processes the frame using MediaPipe to identify body landmarks and uses them to classify the gestures made by the ATC and publishes the command.

MediaPipe Pose Detector: MediaPipe is used to detect and analyze human body pose landmarks in real-time. The pose detection model identifies key points of the human body, such as the wrists, elbows, and shoulders, and provides their normalized x,y coordinates.

Gesture Classification: Includes a custom function that classifies gestures based on the relative positions of the detected landmarks. Each gesture corresponds to a predefined command.

ROS2 Node Publishing: A ROS2 node processes the video feed, detects gestures, and publishes the classified commands on a ROS2 topic (/atc/orders).

```
[INFO] [1733335309.482379971] [sign_detector]: Published Instruction: advance
[INFO] [1733335309.585180638] [sign_detector]: Published Instruction: advance
[INFO] [1733335309.679826754] [sign_detector]: Published Instruction: advance
[INFO] [1733335309.782662317] [sign_detector]: Published Instruction: advance
[INFO] [1733335309.880590936] [sign_detector]: Published Instruction: advance
```

Fig. 7. Instructions published to /atc/orders topic (e.g. advance).

6.2 vehicle_controller.py

This project implements a ROS2 node named VehicleController that subscribes to a topic (/atc/orders) to receive commands. These commands, generated by a gesture recognition system, are processed by the node to simulate actions such as stopping, advancing, and turning a vehicle. The "VehicleController" node listens for messages on a ROS2 topic and executes predefined actions based on the received commands. It acts as a simulation of a vehicle controller, showcasing how ROS2 nodes can interact to achieve a coordinated system.

The VehicleController node starts by subscribing to the topic "/atc/orders", where commands are published as std_msgs.msg.String messages. The "listener_callback" method is triggered whenever a new message is received on the topic and then passes the command to the "execute_command" method, which logs the corresponding vehicle action.

```
[INFO] [1733335391.827738573] [vehicle_controller]:
Received Command: advance
[INFO] [1733335391.828327375] [vehicle_controller]: Advancing...
[INFO] [1733335391.828799871] [vehicle_controller]: Published Twist: geometry_msgs.msg.Twist(linear=geometry_msg
s.msg.Vector3(x=1.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
```

Fig. 8. Subscription to topic /atc/orders and publish to topic /cmd_vel (e.g. Instruction: advance).

To simulate the vehicle's movement, the node publishes geometry_msgs.msg.Twist messages to the /cmd_vel topic. These messages control the vehicle's linear and angular velocities, allowing it to move in the Gazebo simulation.

6.3 gazebo.launch.py

This launch file sets up the simulation environment in Gazebo and spawns a vehicle to interact with the VehicleController node. It begins by launching Gazebo with a specified custom world file, creating the environment for the simulation. The vehicle, defined by its URDF model [8], is then spawned at a designated position within this environment.

With the simulation running, the VehicleController node can publish velocity commands to the /cmd_vel topic, enabling the vehicle to move in response to

commands such as stop, advance, turn_left, and turn_right. This setup demonstrates how ROS2 nodes can interact with a simulated environment to control a vehicle's movements in real time, integrating high-level commands with the physical behavior of the simulation.

However, due to problems with the packages and dependencies, it wasn't possible to spawn the vehicle in the .world file and watch the results in the Gazebo.

7 Conclusion

In conclusion, this project successfully addresses a critical challenge in autonomous driving, enabling vehicles to interpret and respond to hand gestures from Authorized Traffic Controllers (ATCs). By integrating advanced technologies such as ROS2, OpenCV, and MediaPipe, the system demonstrates a robust pipeline for real-time gesture recognition, decision-making, and control. The three-layer architecture, including perception, decision making, and control, ensures seamless integration of data flow and system responsiveness.

The use of Python to implement ROS2 nodes and gesture recognition modules simplifies development while achieving reliable performance. Through a structured approach to gesture classification and vehicle control, the project highlights the potential for autonomous vehicles to adapt effectively to mixed-traffic environments where human and machine interactions are crucial for safety and efficiency.

Although the current implementation displays promising results, it wasn't possible to simulate the results in the Gazebo simulator and use the YOLO algorithm as said in the State of the Art. However, further refinements, such as improving the accuracy of gesture detection under varied lighting conditions or testing in diverse real-world scenarios, would be beneficial. This system represents a significant step towards enhancing the adaptability and safety of autonomous vehicles in dynamic real-world settings.

References

1. Wang, Xianghan, et al. "Research on gesture recognition method based on computer vision." MATEC Web of Conferences. Vol. 232. EDP Sciences, 2018.
2. Sonkusare, Jayesh S., et al. "A review on hand gesture recognition system." 2015 International Conference on Computing Communication Control and Automation. IEEE, 2015.
3. Biswas, Kanad K., and Saurav Kumar Basu. "Gesture recognition using microsoft kinect®." The 5th international conference on automation, robotics and applications. IEEE, 2011.
4. Saxena, Ankita, Deepak Kumar Jain, and Ananya Singhal. "Hand gesture recognition using an android device." 2014 fourth international conference on communication systems and network technologies. IEEE, 2014.

5. Anwar, Shamama, et al. "Hand gesture recognition: a survey." Nanoelectronics, Circuits and Communication Systems: Proceeding of NCCS 2017. Springer Singapore, 2019.
6. Garg, Pragati, Naveen Aggarwal, and Sanjeev Sofat. "Vision based hand gesture recognition." International Journal of Computer and Information Engineering 3.1 (2009): 186-191.
7. Xu, Pei. "A real-time hand gesture recognition and human-computer interaction system." arXiv preprint arXiv:1704.07296 (2017).
8. Sinha, Keshav, et al. "A computer vision-based gesture recognition using hidden Markov model." Innovations in Soft Computing and Information Technology: Proceedings of ICEMIT 2017, Volume 3. Springer Singapore, 2019.
9. Kim, Minwoo, et al. "IMU sensor-based hand gesture recognition for human-machine interfaces." Sensors 19.18 (2019): 3827.
10. BMW Blog, "BMW Gesture Control: The Quick How-To Guide." Accessed: Oct. 31, 2024. [Online]. Available: https://www.bmwblog.com/2022/10/07/bmw-gesture-control-how-to-guide/
11. GestSure, "GestSure | Take control of your operating room" Accessed: Oct. 31, 2024. [Online]. Available: https://www.gestsure.com/
12. Alabdullah, Bayan Ibrahimm, et al. "Smart Home Automation-Based Hand Gesture Recognition Using Feature Fusion and Recurrent Neural Network." Sensors 23.17 (2023): 7523.
13. Alemuda, Fariz, and Fuchun Joseph Lin. "Gesture-based control in a smart home environment." 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE, 2017.
14. Bonci, Andrea, et al. "Robot Operating System 2 (ROS2)-based frameworks for increasing robot autonomy: A survey." applied sciences 13.23 (2023): 12796.
15. Casado Navarro, Kenneth. ROS2 versus AUTOSAR: Automated Parking System case-study. MS thesis. Universitat Politècnica de Catalunya, 2022.
16. Mehrooz, Golizheh, Emad Ebeid, and Peter Schneider-Kamp. "System design of an open-source cloud-based framework for internet of drones application." 2019 22nd Euromicro Conference on Digital System Design (DSD). IEEE, 2019.
17. Li, Jian, et al. "Smart railways: the design and construction of an autonomous inspection and maintenance vehicle." Procedia CIRP 128 (2024): 61-65.
18. Pandya, Nishit V., et al. "Decentralized Information-Flow Control for ROS2." Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS'24). 2024.
19. Open Robotics, "Understanding nodes - ROS2 Documentation: Foxy Documentation" Accessed: Oct. 19, 2024. [Online]. Available: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html
20. Konda, Krishna Chaitanya. Integration of vehicle interface to Autoware. Diss. Technische Hochschule Ingolstadt, 2022.
21. Tsukada, Manabu, et al. "AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles." 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall). IEEE, 2020.
22. Iyer, Nalini C., et al. "Virtual simulation and testing platform for self-driving cars." ICT Analysis and Applications: Proceedings of ICT4SD 2020, Volume 2. Springer Singapore, 2021.

23. Reddy, D. Santhosh, et al. "Robust Obstacle Detection and Collision Warning for Autonomous Vehicles Using Autoware Universe." 2024 16th International Conference on Computer and Automation Engineering (ICCAE). IEEE, 2024.
24. Kumar, Sudhanshu, et al. "A Comprehensive Review on Sentiment Analysis: Tasks, Approaches and Applications." arXiv preprint arXiv:2311.11250 (2023).
25. Vamsi, Kodeti Surya, Sashank Gangadharabhotla, and Vallabhaneni Sri Harsha Sai. "A deep learning approach to solve sudoku puzzle." 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2021.
26. Jarali, Nikith M., et al. "Assistive Aid for Visually Impaired People." ICDSMLA 2020: Proceedings of the 2nd International Conference on Data Science, Machine Learning and Applications. Springer Singapore, 2022.
27. Sharma, Ayushi, et al. "Object detection using OpenCV and python." 2021 3rd international conference on advances in computing, communication control and networking (ICAC3N). IEEE, 2021.
28. Mishra, Shubham, et al. "An intelligent motion detection using OpenCV." International Journal of Scientific Research in Science, Engineering, and Technology 9.2 (2022): 51-63.
29. OpenCV team, "About - OpenCV." Accessed: Oct. 29, 2024. [Online]. Available: https://opencv.org/about/
30. Wu, Zhihuan, et al. "Rapid target detection in high resolution remote sensing images using YOLO model." The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 42 (2018): 1915-1920.
31. Sankaraiah, Y. Ravi, et al. "Artificial Intelligence approach for helmet detection with number plate extraction using YOLO." (2022)
32. Borra, Subba Reddy, et al. "Deep CNN Framework for Object Detection and Classification System from Real Time Videos." Journal of Science and Technology (JST) 8.12 (2023): 78-93.
33. Mingo Hoffman, Enrico, et al. "Yarp based plugins for gazebo simulator." Modelling and Simulation for Autonomous Systems: First International Workshop, MESAS 2014, Rome, Italy, May 5-6, 2014, Revised Selected Papers 1. Springer International Publishing, 2014.
34. Tukhtamanov, Nikita, et al. "Open Source Library of Human Models for Gazebo Simulator." 2022 International Siberian Conference on Control and Communications (SIBCON). IEEE, 2022.
35. Baratta, Alessio, et al. "Digital twin for human-robot collaboration enhancement in manufacturing systems: Literature review and direction for future developments." Computers and Industrial Engineering (2023): 109764.
36. FS Studio, "ROS Gazebo: Everything You Need To Know - Robotic Simulation Services." Accessed: Oct. 18, 2024. [Online]. Available: https://roboticsimulationservices.com/ros-gazebo-everything-you-need-to-know/
37. GitHub Open Robotics, "car_demo" Accessed: Dec. 4, 2024. [Online]. Available: https://github.com/osrf/car_demo