Implementation of a ROS-Based LiDAR Target Following System for Smart Cart

Xin Li[1], Lei Fang[3], Xiaojia Yu[3], Bolin Wu[2], Wenjing He[3]
[1]Nanchong Electronic Information Industry Technology Research Institute, Nanchong 637000, Sichuan, China
[2]Lunqu Technology (Dongguan) Co., Ltd., Dongguan 523808, Guangdong, China
[3]School of Medical Imaging, North Sichuan Medical College, Nanchong 637000, Sichuan, China

Corresponding author: Wenjing He (e-mail: 18990715902@163.com)

ABSTRACT As an important branch of robotics, intelligent vehicles have become a major research focus for improving the flexibility, adaptability, and autonomy of material handling in modern manufacturing facilities. Benefiting from advances in electromagnetic sensing and real-time signal processing, LiDAR-based perception technologies provide reliable environmental information for autonomous navigation and target tracking. This paper investigates the development of a LiDAR target-following intelligent vehicle system based on the Robot Operating System (ROS), with particular emphasis on the design and implementation of node communication and data transmission mechanisms within the ROS framework. The system accomplishes target-following tasks through multiple functionally distinct yet collaboratively operating ROS nodes, including a LiDAR driver node, a target detection node, a tracking node, and an Extended Kalman Filter (EKF)-based pose estimation node. Experimental results demonstrate stable and reliable target-tracking performance across diverse environments. The proposed system provides both a theoretical basis and a practical engineering solution for intelligent vehicles in autonomous navigation, medical applications, and automated material transport, while offering useful insights into electromagnetic perception and real-time sensing systems.

INDEX TERMS ROS system, LiDAR, target following, electromagnetic sensing, data transmission

I. INTRODUCTION

THE next phase of the robotics revolution is well underway, driving the integration of modular, scalable, and reliable architectures into production environments to meet the sector's specific demands for sensing, mobility, and autonomous material handling. The Robot Operating System (ROS) significantly accelerates the pace of robotics research by providing free components and a modular framework [1].As an open-source robot software framework, ROS realizes inter-program communication through nodes, topics, and services. Developers can split robot functions into independent modules, develop them separately and integrate them through ROS, which greatly simplifies the development process of robots and improves development efficiency. In addition, ROS also provides a full set of development tools, covering the whole process from simulation to deployment, which makes ROS widely used in robot development. For example, DS Sanjaya, P Hullikuppi et al. proposed an effective method for simulating the communication between edge radar sensors and electronic control units (ECU) based on ROS, which can process the perceived data more reliably, providing practical experience for the development of autonomous driving technology [2]. ET Baek et al. proposed a ROS-based unmanned mobile robot platform for agriculture, with a new type of autonomous mobile robot with a two-wheel drive chassis to solve the problem of limited indoor workspace and complex environment. This method is applicable to all greenhouse environments and is expected to improve production efficiency and reduce labor costs in agricultural working environments [3]. More directly relevant to industrial manufacturing, D'Avella et al. developed a ROS-Industrial based robotic cell for Industry 4.0, utilizing an eye-in-hand stereo camera and visual servoing to achieve flexible, fast, and accurate picking and hooking operations in a jewelry component production line [4]. It can be seen that ROS and various sensors not only greatly enrich the functions of the robot, but also make the robot more intelligent.

This research focuses on the ROS-based lidar target following intelligent car system, integrating optimized tracking algorithms and motion control strategies to facilitate autonomous material transport and obstacle avoidance within the densely structured machinery layout of manufacturing plants. The aim of this paper is to design and implement an intelligent car system that can stably follow the target by studying the node communication and data transmission mechanism under the ROS architecture. Unlike sensor fusion schemes [5,6] that pursue high-precision multimodal perception with real-time control, this study is dedicated to constructing a real-time target-following solution based

on low-cost single LiDAR for a specific mobile robot platform. Its core concerns lie in the reliability of hardware integration under limited resources, the stability of real-time communication under the ROS architecture, and the effectiveness of the engineering deployment of PID control algorithms. It provides a theoretical basis and technical solution with practical value for the practical application of intelligent cars in industrial logistics scenarios, and is expected to promote the development of this field.

II. HARDWARE DESIGN

In order to meet the requirements of a scalable, modular software architecture and support the processing of

real-time data streams as well as control needs [7], the hardware design architecture is divided into two

layers, the decision-making layer and the execution layer. The decision-making core is the Raspberry Pi,

responsible for receiving environmental data and generating decisions. The execution core is the STM32,

responsible for executing commands and providing motion feedback. The Raspberry Pi is connected to LiDAR

and a camera as environmental perception sensors, which stream real-time point cloud and image data to it

respectively.. ROS (Robot Operating System) is a robot software framework that works with Raspberry Pi to

process perception data and execute target tracking algorithms.

In the execution architecture, the quadrature encoder feeds back the motor speed and wheel mileage

information, providing a closed-loop basis for motion control. The OLED displays the system status

(e.g., battery level, error code). The STM32 module adopts an integrated drive and control design

equipped with IMU. It fuses encoder data and IMU (inertial measurement unit) information, feeds back

three-axis speed, acceleration and angular velocity through the serial port. Simultaneously, it receives ROS

control data frames and generates motor control signals to drive DC motors for propelling the wheels,

calculates the required front wheel steering angle via the Ackermann kinematics model, converts it into a corresponding PWM duty cycle signal, and outputs it to the steering gear to adjust the turning angle. The main hardware connection diagram of the car is shown in Figure 1:

III. KEY NODE IMPLEMENTATION AND DATA TRANSMISSION

Referring to Service Robot Software Development Plan [8], the system's ROS node design includes lidar node (lslidar_driver_node), lidar message processing node (laser_track), control message generation node (follower), chassis control node (wheeltec_ robot) and pose node (robot_pose_ekf). The design of their nodes and topics is illustrated in Figure 2, with the program flow depicted in Figure 3.

A. LIDAR NODE

The lidar node's main functions encompass three stages: data validation, message construction, and data publishing.

Data Validation: The node obtains environmental perception data from the lidar via a serial port, then verifies the transmission reliability over the communication link using the CRC8 checksum algorithm through the following steps:
① Data Length Check: Determines whether the length of the serially received data is greater than 1 byte (must contain at least valid data and a CRC checksum).
② Checksum Extraction: Retrieves the received CRC checksum (assumed to be the last byte) from the data.
③ Checksum Calculation: Performs a modulo 2 division on the first n-1 bytes of data to generate the calculated CRC checksum.
④ Bidirectional Comparison: If the received value matches the calculated value, the process proceeds to data processing; otherwise, a warning log is triggered.

Message Construction: After validation, the system parses the raw data into a ROS-standard sensor_msgs::LaserScan message with key parameters configured as follows:
① Time and Coordinate Frame: The timestamp is the current system time, and the coordinate frame ID is set to "laser_frame".
② Scanning Range: The angular range is from $ -/2 $ to $ /2 $, with an angular increment of $ /180 $ (1° per point), and a single scan duration of 0.1 seconds.
③ Range Thresholds: The valid distance range is from 0.1 to 10.0 meters.
④ Data Structure: The data structure consists of 360 data points, containing a distance value and a signal strength (reflecting measurement reliability).

Data Publishing: The processed laser scan data is published through /scan topic at a frequency of 10Hz for Laser_track, lidar_driver_node and other nodes to subscribe.

B. LIDAR MESSAGE PROCESSING NODE

In lasertrack.py, the message_filters library for multi-sensor data time synchronization is imported. Through this library, the /scan topic is subscribed to receive LaserScan messages. The sliding window noise filtering algorithm is used to detect and track the target object in real time, comparing its distance with the lidar scanning range to find the nearest target. The position of this target is then calculated by performing the following operations:
① Distance Validity Verification: Checks whether the target distance is within the effective range (range_max) of the sensor. If it exceeds this range, a warning indicating that no target is detected is published.
② Angle Calculation: Calculates the polar coordinate angle of the target based on the target point index, starting angle (angle_min) and angle resolution (angle_increment).

On the other hand, to prevent a machine part or wall from becoming the nearest object, visual information needs to be incorporated. For this purpose, the camera driver node usb_cam is installed, which is a commonly used USB

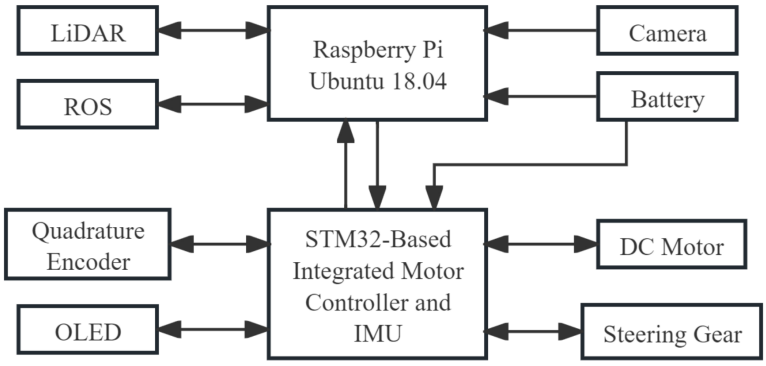

FIGURE 1. The main hardware connection diagram of the trolley

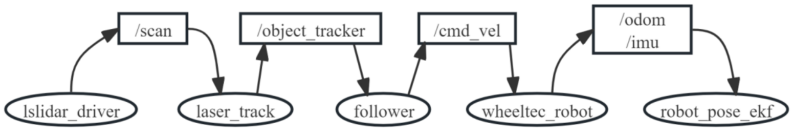

FIGURE 2. Topics and nodes of the system

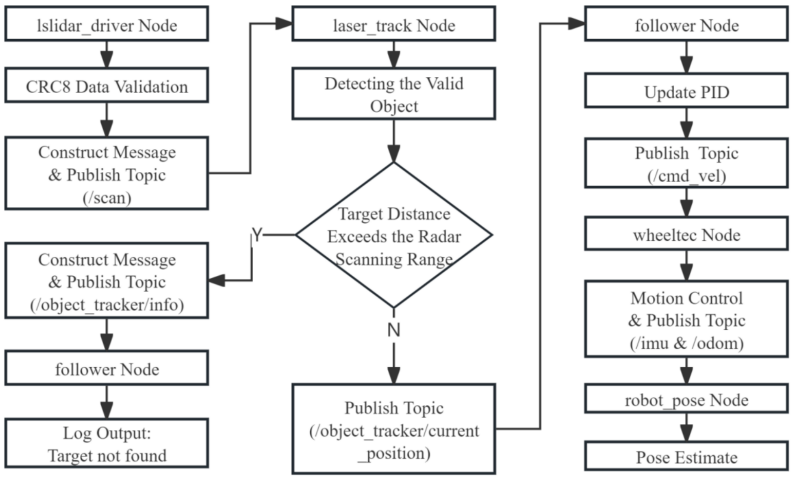

FIGURE 3. Flow chart of lidar target following program

camera driver that can be installed directly using "apt-get". The message_filters library is used to subscribe to the /usb_cam/image_raw topic to obtain sensor_msgs/Image messages. Based on the distance and angle of the nearest target, its position is calculated, and an HSV threshold is set to check if the color near that position falls within the HSV threshold range. If the target is not within the camera's field of view, or if the target's color is outside the threshold range, a warning indicating that no target is detected is published to the /object_tracker/info topic. Otherwise, the angle and distance information of the target is output, a message of type StringMsg is constructed, and it is published to the /object_tracker/current_position topic.

To address issues such as target occlusion and signal noise affecting data validity for LiDAR in real-world environments, this node adopts a sliding window noise filtering mechanism based on temporal analysis to enhance the reliability of target detection. It uses the sliding window algorithm of time series comparison to identify valid targets, and the specific steps are as follows:

① Historical Data Verification: Checks whether there is a previous frame scan data (LastScan) to ensure the validity of time series comparison.

② Laser Point Sorting Traversal: Traverses the laser points

in the current frame from small to large according to the distance, and gives priority to the near target.
③ Window Extraction and Comparison: Firstly, constructs a sliding window around the current point, If a sufficient number of historical points exist whose distance is close to the current point (with a difference less than deltaDist), the current point is judged as potentially belonging to a persistent real target, rather than random noise or transient obstructions such as floating fabric or a passing pedestrian. This judgment, based on multi-frame consistency, effectively improves the integrity of target data in complex dynamic environments.
④ Target Lock and Update: Records the index and distance value of the first valid target, and updates the historical data to the current frame.

C. CONTROL MESSAGE GENERATION NODE

The follower node subscribes to the /object_tracker topic via the laser_follow.py file to receive topic messages. Upon receiving a Position-type target position message, it updates the PID control and subsequently publishes a Twist-type speed message to the /cmd_vel topic to control the cart's speed, thereby achieving target following. If the target is directly in front of the car and the distance between them is the set following distance, the car will remain stationary.

After the system obtains the target location, it processes and normalizes the location information, mainly including: ① Data Extraction: Analyzes the horizontal angle and straight-line distance of the target from the topic message. ② Angle Normalization: Converts the front view to the rear view to adapt to the robot control logic.

In terms of PID speed control, the system uses two PID controllers to calculate linear velocity and angular velocity respectively. Firstly, calculates the angle error and distance error. The angular error is calculated by normalizing the difference between the angle and the angle of the target; The distance error is the difference between the current distance and the target following distance. Secondly, calculates the output speed by PID algorithm based on the angle error and distance error. Thirdly, carries out two-way limiting on angular velocity and linear velocity to prevent motor overload, and inverts the linear velocity to control the symbol change caused by the change of viewing angle.

Encapsulate the data processed by the PID controller into Twist messages and publish them. Set the X-axis component to the calculated linear velocity, and keep the Y/ Z-axis at 0. Set the Z-axis component to the calculated angular velocity, and keep the X/ Y-axis at 0. Finally, publish the speed command through the / cmd_vel topic to drive the robot to move.

D. WHEELTEC_ROBOT NODE

Wheeltec_robot node receives speed instructions by subscribing /cmd_vel topic, and then sends a downlink frame (Table 1) to the STM32 all-in-one driver controller through the serial port to control the movement of the trolley, and receives the uplink frame of the STM32 controller to obtain the speed, acceleration and angular velocity of the three axes of the trolley, and posts /imu, /odom topics. Its core functional modules include speed command parsing, data frame encapsulation, and serial communication.

The node receives the Twist message from the /cmd_vel topic, converting the floating-point speed data into an integer format suitable for serial transmission. It extracts the linear velocity components (linear.x, linear.y) and angular velocity component (angular.z) from the message. The speed values are multiplied by 1000 to convert m/s and rad/s into mm/s and mrad/s, thereby enhancing data transmission precision. Finally the 16-bit integer values are then split into high and low bytes for byte-order transmission. Table 1 shows the downlink data frame structure of serial communication. The header and end of the frame are defined by fixed identification bytes (0x7B and 0x7D) to define the data frame boundary. Two reserved bytes (indices 1-2) are allocated for future functionality expansion. Bytes 3-8 store the linear velocity in the X and Y directions and the angular velocity in the Z axis in sequence, with each component occupying 2 bytes. Finally, BCC is used to ensure the reliability of data transmission.

TABLE 1. Downstream data frame content and description

Byte index	Content	Illustrate
0	0x7B	Frame header
1-2	0x0000	Reserved
3-4	x speed	X-axis linear velocity
5-6	y speed	Y-axis linear velocity
7-8	z speed	Z-axis linear velocity (mrad/s)
9	BCC	XOR
10	0x7D	Frame end

The STM32 controller parses the received linear velocity (x speed, y speed) and angular velocity (z speed). The angular velocity is encoded into a target position via a PWM signal and sent to the steering gear. The steering gear integrates closed-loop feedback internally, enabling it to automatically correct positional deviations and adjust itself to the target angle. As long as the rotational speed corresponding to this angular velocity command is lower than the steering gear's maximum response speed (selected as 428°/s for this system), it ensures that the commanded angle can be accurately reached within the control cycle, thereby achieving precise, real-time control of the steering angle.

E. ROBOT_POSE_EKF NODE

Robot_pose_ekf node subscriptions to /imu, /odom, etc., to get IMU and odometer data. It utilizes the Extended Kalman Filter (EKF) algorithm to fuse these data and estimate the smart cart's pose. The results are published as a PoseWithCovarianceStamped message type to the /robot_pose_ekf/odom_combined topic. The results contain the pose information of the car and the corresponding

covariance matrix to represent the uncertainty of the pose estimation.

The Extended Kalman Filter fusion algorithm is the core fusion engine. It first predicts the current state based on the system motion model, using the previous pose and current control input. It then corrects the predicted state by incorporating the angle measurement from IMU and the position measurement from odometer through the Kalman gain. Finally, the covariance matrix of state estimation is maintained, and the weights of each sensor are dynamically adjusted to ensure the stability of the system.

IV. FUNCTIONAL TESTING AND ANALYSIS

A. TARGET DETECTION ACCURACY TEST

The test was conducted in an indoor standardized environment, using lidar for multi-range detection of static targets. The target object was a circular calibration plate with a diameter of 30cm, placed on a flat ground, and

fixed at four calibration positions of 0.5m, 1.0m, 2.0m and 3.0m respectively. The cart remained stationary while the lidar continuously collected 20 sets of data at a frequency of 10 Hz. The arithmetic mean of these sets was taken as the detection result to eliminate random errors. The following table presents four sets of data.

TABLE 2. Experimental data of object detection

Number of experiments	Actual distance to the target (m)	Target detection distance (m)	Detection Error (m)
1	0.50	0.48	0.02
2	1.00	0.99	0.01
3	2.00	2.02	0.02
4	3.00	3.01	0.01

The cart demonstrates high detection accuracy and stability within the 0.5-3.0 meter range. This indicates that the lidar target detection module of the cart has reliable ranging capabilities in near-to-medium distances, with minimal error fluctuations. It provides stable input data for subsequent target tracking and path planning, meeting the precision requirements of real-time following scenarios for smart carts.

B. FOLLOWING PERFORMANCE TEST

Regarding the evaluation of moving object following performance, related studies typically employ highprecision motion capture systems or real-time positioning systems as the "ground truth" to obtain precise motion parameters or absolute position coordinates of dynamic targets [9,10]. However, such systems are costly and have specific environmental requirements. To complete system verification under resource-limited conditions, this paper adopts a vision-based measurement method using external reference objects.In the experiment, the car was made to follow a remote-controlled trolley with fixed speeds of 0.5, 1.0, 1.5, 2.0, and 2.5m/s respectively. The linear speed and angular velocity of the car can be directly displayed by the OLED screen connected to the STM32. To measure the instantaneous actual following distance, a method involving setting up a reference scale in the scene and taking photographs was employed. A tape measure with an accuracy of $\pm$1cm served as the reference scale. One person held one end, attempting to follow the target, while another held the opposite end, attempting to follow the smart car. A third person was responsible for taking a photograph every five seconds during each test, within the motion plane of the target and the smart car and perpendicular to the direction of motion. The straight-line distance between the smart car's LiDAR position and the target was then read from the pictures based on the tape measure. The difference between

the maximum and minimum values of this distance across five tests was taken as the "following distance change". The results are recorded in Table 3.

TABLE 3. Target-following performance experiment data

Number of trials	Target velocity (m/s)	Trolley linear speed (m/s)	Trolley angular velocity (rad/s)	Follow distance change (m)
1	0.5	0.49	0.05	$\pm$0.05
2	1.0	1.03	0.11	$\pm$0.08
3	1.5	1.51	0.14	$\pm$0.04
4	2.0	2.01	0.20	$\pm$0.10
5	2.5	2.54	0.28	$\pm$0.15

Based on the analysis of experimental data, the intelligent car performs well in the process of target following. The linear speed control accuracy is high. In the range of target speed of 0.5~2.5m/s, the maximum linear speed error of the trolley is only 0.04m/s, and the minimum is 0.01m/s. It indicates that its speed tracking ability is stable and reliable. At the same time, the angular velocity increases with the increase of target velocity. This reflects that the system can still effectively maintain the target azimuth alignment when the steering adjustment demand is enhanced. The following distance fluctuation is controlled between $\pm$0.04m and $\pm$0.15m, and the overall fluctuation range is small, which verifies the range stability of the system in dynamic following.

C. SYSTEM STABILITY TEST

The experiment was carried out on the Ubuntu platform. The hardware configuration was Intel i5-1135G7 processor (4 cores and 8 threads) and 16GB memory. The lidar used was the Lsn10p lidar. The target moved at a speed of 0.5~2.5m/s by following a remote-controlled trolley at a fixed speed to simulate a dynamic following scene. The system used the rosbag tool to record the CPU, memory usage, and message flow data during the running period at a sampling period of one second. The top command was used to collect CPU and memory data in real time, while rostopic hz was used to count message frequency and rosnode info to detect node status. The following table presents the resulting data.

TABLE 4. Experimental data of system stability performance

Number of trials	CPU Usage (%)	Memory Usage (%)	Message Frequency (HZ)	Message loss rate (%)	Errors and exceptions
1	31	41	10	0.1	None
2	33	42	10	0.1	None
3	37	38	10	0.2	None
4	34	44	10	0.1	None

The experimental data indicates that CPU usage remained within the range of 31%-37%, and memory usage was stable between 38% and 44%. This demonstrates balanced system resource utilization without issues of resource contention or leakage. The message release frequency is strictly maintained at 10Hz, which is consistent with the design parameters, and the message loss rate is only 0.1%~0.2%, so the real-time and integrity of data transmission are effectively guaranteed. None of the tests showed any error or exception logs, verifying the robustness of the software architecture. The experimental results show that the ROS-based system design can operate stably for a long time in the lidar target following scenario.

V. CONCLUSION

This design adopts a hierarchical hardware architecture and ROS-based node software design. The system hardware is divided into decision-making and execution layers. The decision-making layer is centered around a Raspberry Pi (Ubuntu 18.04 + ROS Melodic), processing data from the lidar and camera, and runs a distance and color recognition-based nearest target detection and tracking algorithm. The execution layer is dominated by the STM32 integrated drive-control module, which integrates motor control, IMU data acquisition and encoder feedback. It sends data frames bidirectionally through the serial port to communicate with the Raspberry Pi to form a closed-loop control. The software designs five key ROS nodes, and uses key data processing methods such as sliding window noise filtering algorithm, dual PID control, and multi-sensor fusion localization based on extended Kalman filter.

Experiments show that the target ranging error is $\leq \pm 0.02$m (accuracy$\pm 2\%$) in the range of 0.5–3m. When the target velocity is $\leq$2.5m/s, the tracking linear velocity error is $\leq$0.04m/s. And the following distance fluctuation is controlled within $\pm$0.15m. The Raspberry Pi has a low resource usage (CPU<37%, memory<44%). The synchronization of 10Hz topics is stable with a message loss rate $\leq$0.2%. Compared to existing research, 2D LiDAR-based systems often report centimeter-level tracking accuracy at typical working distances of several meters. For example, in static or quasi-static high-precision measurements of ankle positions, one study using an RPLiDAR A3 sensor reported an average absolute error of about 0.046 m for axial tracking [11]. In

dynamic pedestrian-tracking scenarios, a study based on a convolutional neural network showed an average localization error ranging from about 0.17 m to 0.43 m across various test environments [10], reflecting the practical distribution of errors in such tasks.

This system, through sliding-window filtering and visual-assisted verification, effectively handles common environmental noise and transient occlusions, improving data validity and tracking robustness. However, in extreme scenarios such as sustained full occlusion or highly structured noise, the current perception scheme may still have limitations. Future work will focus on fusing depth-camera information or adopting learning-based perception models to further enhance data integrity and target-tracking performance in complex environments.

AUTHOR CONTRIBUTIONS

Conceptualization –Xin Li, Lei Fang, XiaojiaYu, BolinWu and Wenjing He; methodology – Xin Li, Lei Fang, BolinWu and Wenjing He; investigation – Xin Li, Lei Fang, XiaojiaYu, BolinWu and Wenjing He; writing-original draft preparation – Xin Li, Lei Fang, XiaojiaYu, BolinWu and Wenjing He. All authors have read and agreed to the published version of the manuscript.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

FUNDING

This research was supported by, This research was funded by the NanChong Science and Technology Planning Project Fund(Grant No.22YYJCYJ0068).

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, Art. no. eabm6074, 2022, doi: 10.1126/scirobotics.abm6074.

[2] D. S. Sanjaya, P. Hullikuppi, A. Joshi, N. D. Kulkarni, and Basawaraj, "Automotive Radar Data Communication and Simulation Using ROS," in International Conference on Information and Communication Technology for Intelligent Systems. Singapore: Springer Nature Singapore, 2024, pp. 35-46, doi: 10.1007/978-981-97-6684-0_4.

[3] E. T. Baek and D. Y. Im, "ROS-Based Unmanned Mobile Robot Platform for Agriculture," Appl Sci, vol. 12, pp. 4335, 2022, doi: 10.3390/app12094335.

[4] S. D'Avella, C. A. Avizzano, and P. Tripicchio, "ROS-Industrial based robotic cell for Industry 4.0: Eye-in-hand stereo camera and visual servoing for flexible, fast, and accurate picking and hooking in the production line," Robotics and Computer-Integrated Manufacturing, vol. 80, 102453, 2023, doi: 10.1016/j.rcim.2022.102453.

[5] J. Bai, S. Li, H. Zhang, L. Huang, and P. Wang, "Robust Target Detection and Tracking Algorithm Based on Roadside Radar and Camera," Sensors, vol. 21, no. 4, pp. 1116, 2021, doi: 10.3390/s21041116.

[6] A. M. Annaswamy, A. Guha, Y. Cui, S. Tang, P. A. Fisher, and J. E. Gaudio, "Integration of Adaptive Control and Reinforcement Learning for Real-Time Control and Learning," IEEE Transactions on Automatic Control, vol. 68, no. 12, pp. 7740-7755, 2023, doi: 10.1109/tac.2023.3290037.

[7] Y. Jiang, N. Walker, M. Kim, et al., "LAAIR: A Layered Architecture for Autonomous Interactive Robots," arXiv preprint arXiv:1811.03563v2, 2018, doi: 10.48550/arXiv.1811.03563.

[8] Y. H. Jo, S. Y. Cho, and B. W. Choi, "Towards a ROS2-based software architecture for service robots," Bulletin of Electrical Engineering and Informatics, vol. 12, no. 5, pp. 3027-3038, 2023, doi: 10.11591/eei.v12i5.5590.

[9] A. Harchowdhury, L. Kleeman, and L. Vachhani, "3D Sensing of a Moving Object with a Nodding 2D LIDAR and Reconfigurable Mirrors," doi: 10.48550/arXiv.1912.13461.

[10] Á. M. Guerrero-Higueras, C. Álvarez-Aparicio, M. C. Calvo Olivera, et al., "Tracking People in a Mobile Robot From 2D LIDAR Scans Using Full Convolutional Neural Networks for Security in Cluttered Environments," Front Neurorobot, vol. 12, pp. 85, 2019, doi: 10.3389/fnbot.2018.00085.

[11] Y. Zeng, H. Yu, H. Dai, et al., "An Improved Calibration Method for a Rotating 2D LIDAR System," Sensors, vol. 18, no. 2, pp. 497, 2018, doi: 10.3390/s18020497.