YAVER: An Autonomous Waiter Robot with Dynamic Human Mapping and Gesture-Based Service Rejection

Çetin Fidan[1], Emre Çelik[1], Eren Köklü[1], Ömer Mutlu Türk Kaya[1], Erkan Uslu[1] and Gökhan Erdemir[2]

***Abstract*— This paper presents YAVER, an autonomous waiter robot for indoor hospitality scenarios. The implemented system combines ROS2 navigation stack, camera–LiDAR fusion, a pretrained YOLOv8n person detector, and a YASMIN-based finite state machine to search for people, project valid detections onto the occupancy-grid map, build a dynamic service queue, and approach each target sequentially. A second YOLOv8n model, fine-tuned for stop-hand detection on a dedicated hand dataset, runs only during approach and serving to interpret explicit task rejection and allow the controller to blacklist the current target and continue the mission without operator intervention. The study also includes a PyQt5-based desktop operator application that combines a wizard-driven map and configuration workflow with live mission supervision, visual feedback, and runtime parameter adjustment. Experiments on multiple distinct environments across 22 runs showed that the system maintained human-detection recall and precision at or above 96.0% and 89.3%, respectively, while sustaining an average processing rate of 15.8 FPS, demonstrating that the proposed architecture can support practical, real-time autonomous service with operator supervision.**

I. INTRODUCTION

The global service robot market is projected to exceed $37 billion by 2030, with hospitality and food-service applications among the fastest-growing segments [1]. Autonomous service robots are increasingly expected to operate in restaurants, clinics, offices, and other indoor spaces populated by humans. In these settings, reliable point-to-point navigation is necessary, but insufficient. A practical waiter robot must also decide whom to approach, when an interaction is no longer desirable, and how to continue the mission without repeated manual intervention. This requirement shifts the problem from pure motion planning to human-aware mission-level autonomy.

The proposed ROS2-based [2] autonomous waiter robot, YAVER, addresses this need by integrating perception, state-based decision-making, and operator-facing supervision tools. Our system architecture couples a mobile base with a forward-facing RGB camera, 2D LiDAR, the Nav2 stack, a pretrained YOLOv8n model for person detection, and a YASMIN finite state machine. During patrol, the robot detects people, estimates their position in the map-frame, filters out irrelevant or duplicate candidates, and stores valid targets in a queue. When the robot approaches a person or reaches a serving range, it activates a second YOLOv8n model that has been fine-tuned for stop-hand detection. A confirmed gesture is interpreted as a task rejection, causing the system to remove the current target and proceed to the next candidate. Experiments conducted across multiple indoor evaluation scenarios show that this integrated workflow supports reliable target discovery, real-time supervision, and mission-level adaptation during indoor service operation. The evaluation scenarios include two indoor configurations, namely `living_room` and `office`, to test the proposed workflow across different layouts.

The main contribution of this work is not a new detector in isolation, but an end-to-end service workflow in which perception directly modifies mission execution. The implemented contributions are as follows:

- A dynamic human-mapping pipeline that uses a pretrained YOLOv8n person detector and projects RGB detections into the global map by fusing camera bearing, filtered LiDAR range, and TF2 transformations.
- A YASMIN state machine that coordinates map selection, configuration loading, patrol search, navigation, serving, pause-resume, battery checking, return-to-dock, and charging.
- A lightweight gesture-based rejection mechanism built on a fine-tuned YOLOv8n hand detector that allows a user to refuse service and forces the queue manager to adapt immediately.
- A PyQt5-based desktop operator application for supervision and control, complemented by the YASMIN runtime visualization interface for state-machine inspection.

The paper is organized as follows. Section II reviews closely related work. Section III describes the implemented architecture together with the supporting runtime figures. Section IV presents the experimental setup and results. Section V concludes the paper and provides directions for future work.

II. RELATED WORK

Prior work relevant to YAVER spans three overlapping dimensions: indoor navigation and sensing architecture, mission-level control, and human-robot interaction through gesture. Examining these together reveals that no prior system integrates all three capabilities simultaneously within a service workflow of the type considered in this work.

[1]Ç. Fidan, E. Çelik, E. Köklü, Ö. M. T. Kaya and E. Uslu are with Dep. of Computer Engineering, Yıldız Technical University, Istanbul, Türkiye {cetin.fidan, emre.celik5, eren.koklu}@std.yildiz.edu.tr, {omer.kaya, euslu}@yildiz.edu.tr

[2]G. Erdemir is with the Dep. of Engineering Management and Technology, University of Tennessee, Chattanooga, TN, USA gokhan-erdemir@utc.edu

Indoor Navigation and Sensing Architecture

Lightweight mobile platforms running ROS2 have demonstrated reliable indoor navigation for service tasks. Daison *et al.* built an autonomous medicine transporter on TurtleBot3 Burger under ROS2 Humble [3], confirming that differential-drive class robots can sustain practical service missions. On the sensing side, Sanboontho *et al.* fused monocular vision with 2D LiDAR for localization and object measurement [4], and Riordan and O'Donnell validated a similar LiDAR–vision fusion for autonomous Gazebo navigation [5]. Despite these sensing capabilities, both approaches treat humans purely as obstacles or passive observations. Neither system evaluates whether a detected person is a valid service target, nor does human intent alter the mission execution. YAVER differs precisely here: the same camera–LiDAR fusion that estimates a person's map-frame position also feeds a queue-management layer that decides whether, and in what order, to serve each candidate.

Waiter and Restaurant Service Robots

Several systems have focused specifically on autonomous waiter robots, but each makes a different simplifying assumption that limits generalization. Yu *et al.* achieved high-accuracy delivery by fixing service targets to known table positions [6]; their system excels at precise delivery but cannot dynamically discover targets from perception. Thanh *et al.* followed predefined line-sensor paths, removing the navigation challenge entirely at the cost of environmental flexibility [7]. Wan *et al.* focused on spill-free beverage transport through jerk-limited motion [8], a complementary contribution that addresses physical stability rather than target selection or interaction. The most relevant prior work is Yano *et al.*, whose robot recognizes waving customers and takes orders using human and action recognition [9]. However, this system inherently assumes affirmative intent: a customer who waves is always served. It provides no mechanism for a user to refuse service once the robot has committed to an approach, and it does not describe how the mission continues after such an event. YAVER is designed to bridge this gap by treating users' refusal as a primary operational event that must be detected, acted upon, and recovered from without operator intervention.

Mission-Level Control

At the control architecture level, YASMIN provides a ROS2-oriented state-machine framework with blackboard-based data exchange and execution tracing [10]. YAVER adopts this interpretable mission-control style, but extends it with logic absent from the base framework: map-based person queuing, duplicate suppression, table-proximity filtering, gesture-triggered blacklisting, and battery-aware autonomous docking. These additions reflect the difference between a general-purpose state machine and one designed to handle the specific contingencies of a human-populated service environment.

Gesture-Based Human-Robot Interaction and Service Rejection

Gesture recognition has become a standard mechanism for directing robots without physical contact. Herbaz *et al.* showed that YOLO-family detectors achieve real-time hand gesture recognition at high accuracy under resource-constrained conditions [11], and García *et al.* validated a YOLO-based pipeline for gesture-driven HRI commands [12]. Both works focus on affirmative command channels—gestures that direct the robot to act. YAVER repurposes the same detection paradigm in the complementary direction, interpreting a stop-hand as an explicit refusal. The importance of this distinction is underscored by Chang *et al.*, who found that when a service robot fails to respond to user intent, trust erodes. Recovery strategies become necessary [13]. YAVER sidesteps this recovery problem entirely by detecting intentional rejection before service is delivered. Urakami and Seaborn further support this design choice, noting that explicit hand gestures are among the most unambiguous nonverbal signals available in HRI [14], making them preferable to subtler cues such as gaze or posture for a rejection trigger.

Summary

Taken together, prior waiter-robot systems each address a subset of the challenges involved in autonomous indoor service: some excel at navigation precision, others at stable delivery, and one at customer recognition. To the best of our knowledge, no prior system integrates dynamic target discovery from perception, gesture-based rejection as a first-class mission event, and battery-aware autonomous recovery within a single unified workflow. YAVER is designed to close this gap, connecting low-level camera–LiDAR fusion to mission-level queue adaptation and operator supervision in an end-to-end architecture.

III. SYSTEM ARCHITECTURE AND IMPLEMENTATION

A. Overall System Structure

The implemented system has four tightly coupled layers. First, a setup workflow selects a map or imports a new YAML map, then either loads or creates a map-specific JSON configuration containing the docking station, table locations, and search waypoints. Second, the main ROS2 node performs perception, marker publication, motion commands, and Nav2 action handling. Third, a YASMIN state machine drives the mission logic on a shared blackboard that contains the current queue, target, served list, rejected list, battery state, and selected configuration. Fourth, operator interfaces subscribe to the same live data streams for supervision and control.

At the software level, the core ROS2 communication and state-machine logic are cleanly separated from the user interface. The desktop application follows a widget-based and event-driven design in which setup, monitoring, and operator commands are grouped into dedicated interface components for map selection, configuration editing, live

visualization, and mission control [15]. The setup-stage map editor used before mission start is shown in Fig. 1.

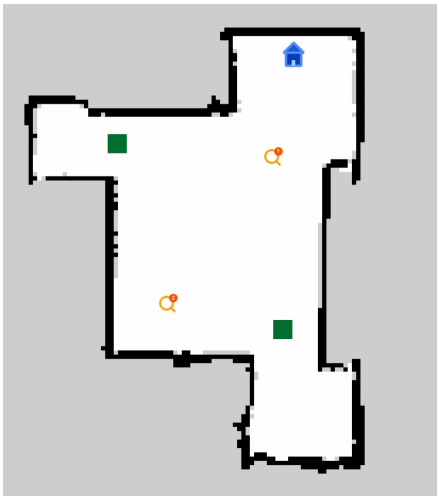


Fig. 1. Setup-stage map configuration view showing the selected occupancy map together with the docking point, table markers, and search waypoints.

The current prototype follows a TurtleBot3-class differential-drive architecture and uses the TurtleBot3 Navigation2 launch workflow in software [16], [17]. RGB perception is provided by a Logitech Brio 100 USB webcam operating at 1920x1080 resolution and 30 FPS, and the image-to-bearing projection model uses its 58° horizontal field of view as a parameter. The camera is mounted 0.03 m ahead of the base frame and 0.10 m above it, while the platform also carries a planar LiDAR processed over the 0.12–3.5 m range. The default mission parameters are a 360° search sweep at 0.3 rad/s, a human-approach offset of 0.5 m, a navigation timeout of 60 s, and a low-battery threshold of 20%.

B. Perception Models and Training Data

The perception stack uses two compact YOLOv8n-based models with different roles in the mission pipeline. For human discovery during patrol, YAVER uses a pretrained YOLOv8n person detector without task-specific retraining in the current deployment. This model is active in the SEARCHING state and provides the image-space detections that are later fused with LiDAR range and TF2 pose information to estimate person locations on the map.

For explicit service rejection, the system uses a second YOLOv8n model fine-tuned for stop-hand detection on the Roboflow *hand-dtozq* dataset [18]. During dataset preparation, images were center-cropped and resized to 640×640 pixels. The augmentation pipeline generated three outputs per training example and included horizontal flipping, random rotation between -15° and $+15^\circ$, brightness variation between -25% and $+25\%$, blur up to 2 px, and noise affecting up to 2% of pixels. This separation between a general pretrained person detector and a task-specific fine-tuned hand detector keeps the search stage lightweight while still allowing a specialized rejection cue during close interaction.

C. Dynamic Human Mapping and Queue Generation

The perception pipeline is centered on the `PersonLocalizer` ROS2 node. In the SEARCHING state, it uses the pretrained YOLOv8n person detector described above on resized 640×480 RGB frames. For a detection center at pixel coordinate u in an image of width W, the horizontal bearing is estimated as

$$\theta = -\left(\frac{u}{W} - \frac{1}{2}\right)\phi, \tag{1}$$

where ϕ is the camera horizontal field of view. In the current implementation, $\phi = 58^\circ$ and the corresponding LiDAR distance is extracted with a median filter over a small angular window. The resulting local coordinates are

$$x_l = r\cos\theta, \qquad y_l = r\sin\theta, \tag{2}$$

which are then transformed from `base_link` into the global `map` frame through TF2, which maintains the relationships between coordinate frames over time and supports transformation between them [19]. The resulting SEARCHING overlay, in which newly discovered candidates appear on the occupancy map, is shown in Fig. 2.

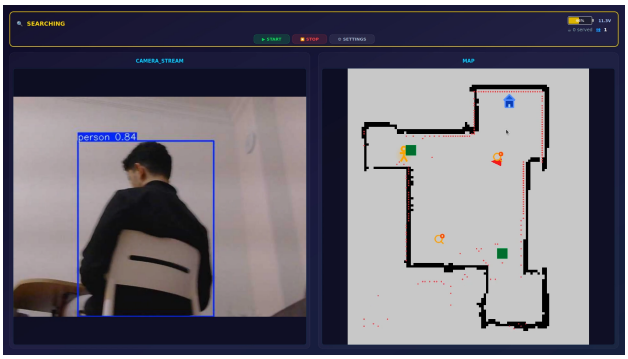


Fig. 2. SEARCHING state after new people are detected and projected onto the map. Newly observed service candidates appear in orange.

Several application-specific filters are applied before a candidate becomes a service target. First, LiDAR samples outside the valid range are discarded. Second, recently observed detections are stored for a short time window to prevent near-identical detections from being inserted repeatedly. Third, a table-proximity rule requires the detected person to be sufficiently close to one of the operator-defined table markers, which matches the waiter-robot use case better than servicing every visible person in the room. Finally, targets that are already served or rejected are discarded before queue insertion. Valid targets are appended to a service queue and visualized on the map, so the configuration stage in Fig. 1 naturally leads into the live detections shown in Fig. 2.

D. YASMIN State Machine and Gesture-Aware Logic

The mission controller is implemented as a YASMIN state machine with the following states: `MAP_SELECTION`, `CONFIG_SELECTION`, `PAUSED`, `SEARCHING`, `NAVIGATING`, `SERVING`, `IDLE`, `BATTERY_CHECK`,

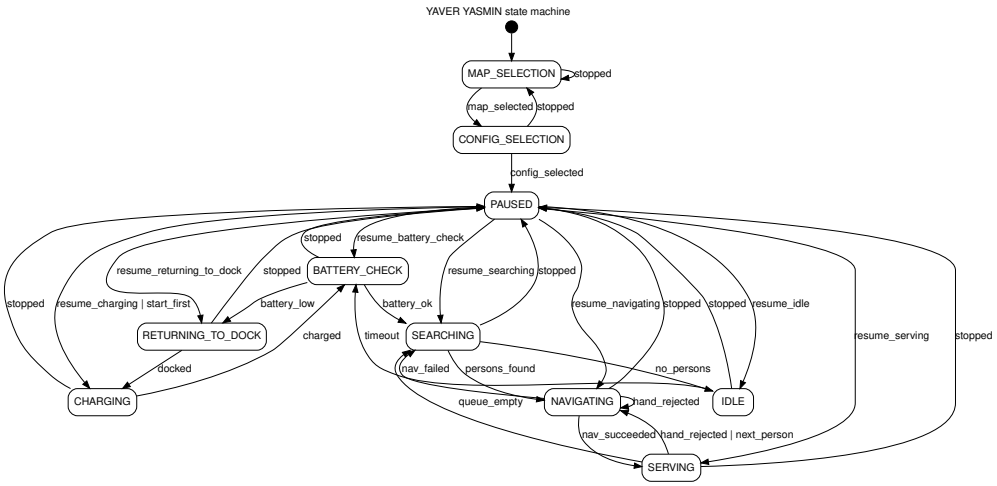


Fig. 3. YASMIN execution view showing the active state machine and runtime transitions during mission execution.

RETURNING_TO_DOCK, and CHARGING. On the first start, the robot performs an in-place localization rotation, then begins waypoint patrol. Search waypoints are traversed in a forward-backward pattern so that the patrol direction flips at the end of the route instead of forcing a full reset. At each waypoint, the robot rotates in place and waits for the perception node to populate the queue. The live YASMIN execution used to inspect the transitions is shown in Fig. 3.

Once the queue is non-empty, the NAVIGATING state computes an approach pose that stops the robot before the target rather than directly at the target coordinates. The remaining distance from Nav2 feedback is stored on the blackboard and reused by the hand-rejection logic. The fine-tuned hand detector is only activated in two cases: while navigating to a person when the remaining distance is below 1.5 m, and while already serving. To reduce false triggers at runtime, the detector also uses a warm-up delay after each state transition and requires 15 consecutive positive frames before raising a rejection flag.

If a rejection gesture is confirmed, the controller blocks the current target, removes nearby duplicates from the queue, publishes a dedicated rejected-person marker, and either continues with the next target or returns to the search cycle. If the service completes normally, the current target is added to the served list and can later be rendered in green in the monitoring interface. The rejection is shown in Fig. 4.

The battery branch is equally explicit: when the charge level falls below threshold, the robot transitions to RETURNING_TO_DOCK, navigates to a pre-docking pose, and then performs a slow reverse docking maneuver before entering CHARGING.

E. Operator Interfaces

The primary desktop operator application is implemented in PyQt5 as a supervisory interface for mission setup and live robot monitoring. The workflow begins with a setup stage where the operator selects a map, loads or creates a configuration, places the docking point, defines table markers, and inserts search waypoints directly on the occupancy-grid image [20]. After setup, the application switches to a full-screen control dashboard that presents the annotated camera stream, live map, robot pose, LiDAR overlay, queue status, battery state, service count, and mission status in real time. The same interface also exposes operator actions such as START, STOP, and the initial-pose publication, along with editable runtime parameters, including duplicate-distance threshold, search speed, serving duration, navigation timeout, and table-proximity threshold.

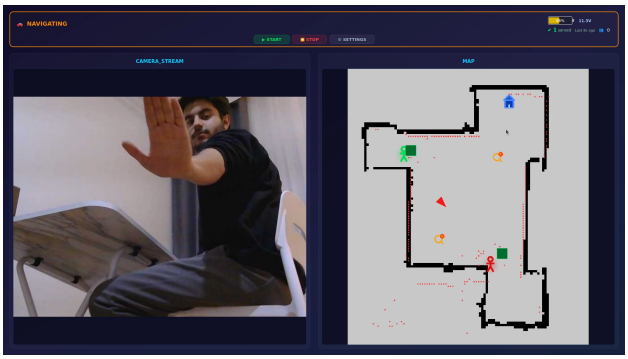
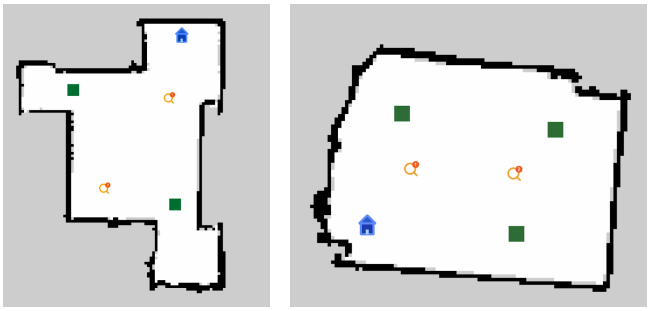


Fig. 4. Map view after a rejection event. Rejected targets are marked explicitly and removed from the active service flow.

From an application perspective, the PyQt5 interface provides four main properties: structured setup, real-time supervision, runtime configurability, and modular operator interaction. Structured setup is achieved through the staged configuration workflow; synchronized visual and state displays support real-time supervision; runtime configurability allows mission thresholds to be adjusted without modifying the autonomy code; and modular operator interaction keeps user commands separate from the ROS2 mission logic. These properties are consistent with Qt's event-driven component model based on signals and slots for inter-object communication [15]. A representative desktop supervision and configuration screen is shown in Fig. 5.

Fig. 5. PyQt5 desktop operator application used for wizard-based setup, live robot supervision, and runtime parameter tuning.



(a) `living_room` (b) `office`

Fig. 6. Evaluation scenarios with different map configurations.

IV. EXPERIMENTAL SETUP AND RESULTS

A. Environment and Procedure

The evaluation was conducted in two indoor scenarios, `living_room` and `office`, to assess the proposed workflow across different map configurations. These scenarios differ in spatial layout and service-point arrangement, allowing the system to be tested beyond a single indoor setup. The corresponding scenario maps are shown in Fig. 6.

Each scenario used an occupancy-grid map with a resolution of 0.05 m and a map-specific configuration containing the docking station, table regions, and search waypoints. Before each run, the robot was initialized at the docking pose defined in the configuration and then executed its waypoint-based patrol in the configured forward–backward order.

A total of 22 complete runs were performed across the evaluation scenarios. Depending on the scenario, multiple participants were placed near the predefined table regions, yielding 69 intended human-service targets in total across the full experiment set. During each run, the pretrained YOLOv8n person detector operated in the SEARCHING state to identify candidate people, project valid detections onto the map frame, and populate the service queue. The fine-tuned stop-hand detector was activated during close approach and served to evaluate gesture-based service rejection. The evaluation focused on person-detection reliability, gesture-rejection recognition, and runtime processing performance.

B. Results and Observations

Table I summarizes the measured results across the evaluated scenarios. In the `office` scenario, 12 runs were performed to evaluate the proposed workflow in a different map layout; the system detected 44 of 44 intended human targets, corresponding to a human-detection recall of 100.0% and a precision of 91.7%. The 4 false-positive detections in the `office` scenario were mainly associated with partial human views, background clutter, and viewpoint-dependent localization shifts during target projection. Across the 10 runs in the `living_room` scenario, the system correctly detected 24 of the 25 intended human targets. In three cases, one additional human target was inserted into the queue, resulting in 3 false-positive person detections overall. This corresponds to a human-detection recall of 96.0% and a precision of 89.3%.

For gesture-based rejection, 8 stop-hand trials were conducted in the `living_room` scenario. The system correctly recognized 6 of these events, missed 2 true rejections, and triggered 1 false rejection. Under this test set, the gesture-rejection recall was 75.0%, and the precision was 85.7%. For the `office` scenario, 16 stop-hand trials were conducted. The system correctly recognized 15 of these events, missed 1 true rejection, and triggered 3 false rejections. This corresponds to a gesture-rejection recall of 93.8% and a precision of 83.3%. The missed rejection events were observed under visually less favorable interaction conditions, including partial hand occlusion, non-frontal hand orientation, and variations in illumination or viewing distance during the robot's approach. These observations suggest that gesture perception can be sensitive to natural interaction conditions and motivate further evaluation under more diverse restaurant-like scenarios. The average runtime processing rates measured during the experiments were 15.2 FPS in the `office` environment and 16.3 FPS in the `living_room` environment, indicating that the perception and supervision pipeline sustained real-time operation throughout the evaluation.

The qualitative figures are consistent with these results and summarize the main stages of the implemented system. Figure 6 illustrates the evaluation scenarios. Figure 1 presents the setup-stage configuration workflow, Fig. 2 shows valid target discovery during SEARCHING, and Fig. 4 documents the rejection branch. In addition, Fig. 3 shows the YASMIN execution view used for runtime state inspection, while Fig. 5 summarizes the PyQt5-based desktop supervision interface used during the experiments.

V. CONCLUSION

This paper presented YAVER, a ROS2 waiter robot that combines dynamic human mapping, YASMIN-based mission control, gesture-aware service rejection, autonomous return-to-dock behavior, and PyQt5-based operator supervision within a single service workflow. The main strength of the system is its end-to-end integration: person detections are

TABLE I
EXPERIMENTAL RESULTS ACROSS EVALUATION SCENARIOS

Metric	`living_room`	`office`
Experimental runs	10	12
Search waypoints	2	2
Intended human targets	25	44
Correct human detections	24	44
False positive person detections	3	4
Human-detection recall	96.0%	100.0%
Human-detection precision	89.3%	91.7%
Gesture-rejection trials	8	16
Correct rejection detections	6	15
Missed rejection events	2	1
False rejection triggers	1	3
Gesture-rejection recall	75.0%	93.8%
Gesture-rejection precision	85.7%	83.3%
Processing rate	16.3 FPS	15.2 FPS

converted into map-grounded service targets, mission execution adapts online through queue management and rejection handling, and the full process remains observable through interpretable state transitions and desktop-based supervision.

The experimental results obtained across different indoor evaluation scenarios show that YAVER can operate as a practical indoor service platform with reliable target discovery, real-time perception performance, and effective mission-level coordination across searching, serving, rejection, and docking stages. Taken together, the proposed architecture provides a solid foundation for autonomous hospitality scenarios that require both robot autonomy and operator oversight.

Future work can extend the interaction flow to support second-order requests after the initial service interaction. Another natural extension is to integrate a kitchen-oriented task branch so that the robot can travel to a preparation area, receive items, and then resume table-service execution. In addition, broader evaluations with more test runs, different map layouts, and diverse real-world operating conditions will be conducted to further examine the system under varied scenarios, including gesture-perception challenges such as partial occlusions, lighting variations, and ambiguous hand poses. False rejection cases will also be addressed by incorporating an additional confirmation step through speech or the operator GUI and by refining the temporal consistency requirement before a target is blacklisted.

ACKNOWLEDGMENT

The authors thank Assoc. Prof. Dr. Ali Can Karaca from the Department of Computer Engineering at Yıldız Technical University for his valuable support.

REFERENCES

[1] Grand View Research, "Service Robotics Market Size, Share & Trends Analysis Report," Market Research Report, 2024. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/service-robotics-industry. Accessed: Mar. 8, 2026.

[2] 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, p. eabm6074, 2022, doi: 10.1126/scirobotics.abm6074.

[3] A. Daison, C. Varghese, S. Sajeev, S. Yohannan, G. R. King, and R. P. S, "Autonomous Mobile Medicine Transporter for Bedridden Patients Using ROS2 Humble and TurtleBot3 Burger," in *2025 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, Nov. 2025, pp. 1–5, doi: 10.1109/RAICS66191.2025.11332601.

[4] W. Sanboontho, A. Pichitkul, S. Tantrairatn, T. Phiboon, and A. Ariyarit, "Indoor Localization and Measurement Object in Field for TurtleBot Using Combined 2D-LiDAR and Monocular Camera," *Journal of Physics: Conference Series*, vol. 3022, no. 1, p. 012004, May 2025, doi: 10.1088/1742-6596/3022/1/012004.

[5] C. Riordan and C. O'Donnell, "Fusion of LiDAR and Computer Vision for Autonomous Navigation in Gazebo," in *SoutheastCon 2023*, 2023, pp. 553–558, doi: 10.1109/SoutheastCon51012.2023.10115172.

[6] Q. Yu, C. Yuan, Z. Fu, and Y. Zhao, "An autonomous restaurant service robot with high positioning accuracy," *Industrial Robot: An International Journal*, vol. 39, no. 3, pp. 271–281, 2012, doi: 10.1108/01439911211217107.

[7] V. N. Thanh, D. P. Vinh, N. T. Nghi, L. H. Nam, and D. L. H. Toan, "Restaurant Serving Robot with Double Line Sensors Following Approach," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2019, pp. 235–239, doi: 10.1109/ICMA.2019.8816404.

[8] A. Y. S. Wan, Y. D. Soong, E. Foo, W. L. E. Wong, and W. S. M. Lau, "Waiter Robots Conveying Drinks," *Technologies*, vol. 8, no. 3, p. 44, 2020, doi: 10.3390/technologies8030044.

[9] Y. Yano, K. Isomoto, T. Ono, and H. Tamukoh, "Autonomous Waiter Robot System for Recognizing Customers, Taking Orders, and Serving Food," in *RoboCup 2023: Robot World Cup XXVI*, vol. 14140, Lecture Notes in Computer Science, C. Buche, A. Rossi, M. Simões, and U. Visser, Eds. Cham, Switzerland: Springer, 2024, pp. 252–261, doi: 10.1007/978-3-031-55015-7_21.

[10] M. A. Gonzalez-Santamarta, F. J. Rodriguez-Lera, C. F. Llamas, F. Martin Rico, and V. Matellan-Olivera, "YASMIN: Yet Another State MachINe library for ROS2," arXiv:2205.13284, 2022.

[11] N. Herbaz, H. El Idrissi, and A. Badri, "Deep Learning Empowered Hand Gesture Recognition: Using YOLO Techniques," in *2023 14th International Conference on Intelligent Systems: Theories and Applications (SITA)*, IEEE, 2023, pp. 1–7, doi: 10.1109/SITA60746.2023.10373734.

[12] I. García, P. Vilcapoma, and J. P. Vásconez, "Human-Robot Interaction Based on Hand Gesture Detection Using YOLO Algorithm," in *Advanced Research in Technologies, Information, Innovation and Sustainability (ARTIIS 2024)*, Communications in Computer and Information Science, vol. 2346, T. Guarda, F. Portela, and G. Gatica, Eds. Cham, Switzerland: Springer, 2025, pp. 333–347, doi: 10.1007/978-3-031-83210-9_26.

[13] X. Chang, Y. Li, S. Liu, L. Ma, and R. LC, "'Sorry to Keep You Waiting': Recovering from Negative Consequences Resulting from Service Robot Unintended Rejection," in *HRI '24: Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, Boulder, CO, USA, Mar. 2024, pp. 96–105, doi: 10.1145/3610977.3634959.

[14] J. Urakami and K. Seaborn, "Nonverbal Cues in Human–Robot Interaction: A Communication Studies Perspective," *ACM Transactions on Human-Robot Interaction*, vol. 12, no. 2, Article 22, Mar. 2023, doi: 10.1145/3570169.

[15] Qt Group, "Signals and Slots – Qt for Python," official documentation. [Online]. Available: https://doc.qt.io/qtforpython-6/tutorials/basictutorial/signals_and_slots.html. Accessed: Mar. 8, 2026.

[16] ROBOTIS, "TurtleBot3 Features," e-Manual. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/. Accessed: Mar. 7, 2026.

[17] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[18] Hand-2pcal, "hand-dtozq," Roboflow Universe dataset. [Online]. Available: https://universe.roboflow.com/hand-2pcal/hand-dtozq. Accessed: Mar. 8, 2026.

[19] T. Foote, "tf: The transform library," in *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013, pp. 1–6, doi: 10.1109/TePRA.2013.6556373.

[20] Qt Group, "QWizard Class – Qt Widgets," official documentation. [Online]. Available: https://doc.qt.io/qt-6/qwizard.html. Accessed: Mar. 8, 2026.