2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

# Towards fault-tolerant deployment of mobile robot navigation in the edge: an experimental study

Florian Mirus[1,*], Frederik Pasch[1,*] and Kay-Ulrich Scholl [1]

***Abstract*— Modern algorithms allow robots to reach a greater level of autonomy and fulfill more challenging tasks. However, on-board limitations regarding computational and battery resources are hindering factors regarding the deployment of such algorithms particularly on mobile robots. Although offloading a majority of the algorithmic components to the edge or even cloud offers an attractive option to leverage massive computing power in robotics applications, safety and reliability remain critical issues. This paper presents a minimalistic safety fallback mechanism when offloading mobile robot navigation to the edge, that ensures safe and collision-free navigation even in the presence of failures in the connection between the on-board and edge-devices. We show the effectiveness of our approach through extensive testing in three different relevant scenarios in a simulated warehouse environment. Our experiments demonstrate the effects of different fallback strategies and show how our proposed approach is able to ensure safety while allowing the robot to continue its mission during an interrupted connection and thus avoiding unnecessary downtime.**

## I. INTRODUCTION

Mobile robot systems are an integral component of the factories of the future. Due to the mobile systems' limitations regarding on-board compute and power resources, there is an increasing interest in edge computing solutions for Autonomous Mobile Robots (AMRs) [1], i.e., employing remote compute platforms (edge) within the same local network to reduce the communication latency compared to cloud computing. Edge and even cloud robotics [2] have emerged as an attractive option to leverage high computing power and large storage spaces in the robotics domain. However, transferring computational workloads from the robots' on-board computing platform to the edge-/cloud-continuum introduces additional challenges such as limited network bandwidth, increased latency and unreliable wireless communication. Although these issues are investigated in current research, safety considerations and their implications on task performance are still important open issues [3].

In this paper, we focus on the safety aspects of transferring algorithmic components from the robot's on-board compute to a remote edge compute platform. As example application, we consider AMR navigation employing the modern Navigation2 stack [4] included with the Robot Operating System (ROS2) [5]. Therefore, we containerize the navigation stack and use Kubernetes to deploy the containers in an edge-server with only the low-level sensor and actuation nodes remaining in the robot's on-board container as depicted in Fig. 1. That is, the robot's perceived raw sensor data is transmitted to the navigation container running on an edge-server, where the navigation modules (i.e., localization, global and local path planning) are executed and only the final velocity command is sent back to the robot to control the actuators. From a safety perspective, the weak spot in a real-world deployment of such a system is the wireless connection between the robot's on-board compute and the navigation components running in the edge. In case this connection gets interrupted, a correct and collision-free performance of the navigation task can not be guaranteed anymore. The simplest and still safe solution would be to detect a broken communication, stop the robot and wait until the communication is reestablished. However, in a large-scale manufacturing environment, even a short downtime of one robot could lead to significantly increased costs [6], [7]. Another option could be to leave the calculation of the velocity commands on the robot to ensure collision-free navigation. However, for scalability reasons, the goal is to keep the robot hardware and software setup as simple and lightweight as possible.

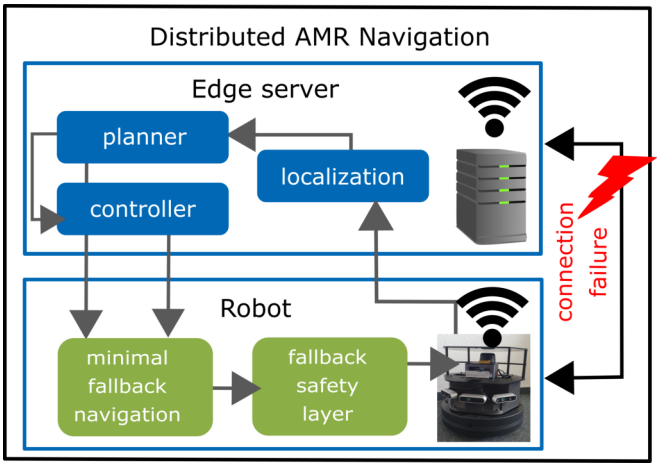



Fig. 1. System overview.

In this paper, we propose a minimalistic fallback mechanism for edge-based mobile robot navigation, that ensures safe, collision-free navigation without stopping the navigation task unless a safety-critical situation occurs, which keeps the impact on the navigation task performance at a minimum level. We demonstrate the feasibility of our approach by extensive evaluation in a simulated warehouse environment in three different relevant navigation scenarios.

* Equal contribution

[1]Florian Mirus, Frederik Pasch and Kay-Ulrich Scholl are with Intel Labs, Karlsruhe, Baden-Württemberg, Germany {florian.mirus, frederik.pasch, kay-ulrich.scholl}@intel.com

## II. RELATED WORK

There is a large variety of research directions focusing on leveraging the enormous potential offered by the edge-/cloud-continuum for robotics [2], [8], [9]. Particularly, employing edge-/cloud-systems in robotics enables exchange of knowledge [10] between individual robots or, going even further, centralized algorithms [11], sophisticated multi-robot collaboration [12] and the creation of a digital twin [13]. Another research direction is offloading compute-heavy and resource-hungry algorithmic components to the edge to reduce the robots' on-board power consumption and thus increase their battery-lifetime [14]. However, transferring computational workloads to a remote compute platform always comes at the cost of additional latency for data transmission and thus there is always a trade-off between network bandwidth and latency vs. on-board compute [14]. Hence, some research has focused on investigating the optimal allocation of algorithms in edge-/cloud-robotics systems [15] as well as the orchestration problem including mixed-criticality considerations [16]. [16] focuses on extending the Kubernetes scheduling system with real-time criticality specifications such that the orchestration scheme prefers to deploy real-time critical containers on on-board devices. Other recent efforts have emerged to combine the de-facto standard robotics framework, the ROS2, with cloud-native tools like Kubernetes [17], [18].

Recently, there also have recent efforts focusing on offloading the navigation components of AMRs and deploying them in the edge [19] or even the cloud [20]. In their real-world experiments, [19] particularly focus on investigating the communication latency when the robot moves from one access point the environment to another one. However, safety remains an important issue when offloading algorithmic workloads such as the entire AMR navigation stack to a remote compute platform in the edge or even cloud [3], which has received less attention compared to other aspects of edge-/cloud-robotics.

## III. FAULT TOLERANT NAVIGATION DISTRIBUTED BETWEEN ROBOT AND EDGE

### A. Offloaded robot navigation

Over the last decade, the Robot Operating System (ROS) [21] has become the de-facto standard for software and application development as well as academic research in the robotics domain with its successor ROS2 [5] being targeted towards industrial applications and product-level readiness. ROS2 comes with its own navigation framework, namely Nav2 [4], a highly configurable and flexible stack for mobile robot navigation. In this paper, we containerize the Nav2 stack with the help of Docker [22] and we use Kubernetes [23] to deploy these containers on different computation platforms within one Kubernetes cluster. We run the navigation stack in one container deployed in an edge-server separated from the robot's low-level control, which is still executed on the robot's on-board compute platform. The navigation part consists of Adaptive Monte Carlo Localization (AMCL) [4], [24] for localizing the robot in combination with the NavFn global planner implementing Dijkstra's algorithm [4], [25] for global path planning and the DWB [4], [26] controller for local path planning as well as Nav2's behavior tree navigator for coordination.

| ID | Short name | Edge | Robot |
|---|---|---|---|
| 1 | no countermeasures | Nav2 | low-level control |
| 2 | network safestop | Nav2 | network safestop before low-level control |
| 3 | safety fallback | Nav2 | safety fallback before low-level control |
| 4 | navigation & safety fallback | Nav2 | fallback controller and safety fallback before low-level control |

TABLE I
OVERVIEW OF THE FOUR DIFFERENT ON-BOARD FALLBACK STRATEGIES.

### B. Potential failures

In case the connection between the robot and the navigation components running in the edge gets interrupted, a correct and collision-free performance on the navigation task can not be guaranteed anymore. There are multiple issues that can cause such an interrupted connection. In this paper, we focus on two possible failure cases, namely a communication error and a failure of the navigation components in the edge including a stateful recovery mechanism. From the robot's perspective, both failure cases are identical with no more velocity commands being received from the edge server. The difference is in the edge's perspective: in case of a communication failure, the edge container stops receiving sensory data and thus the navigation components in the edge are unable to update their internal belief about the robot's state and, once the communication is restored, need to handle discrepancies between new incoming sensory data and their outdated internal belief, which leads to high uncertainty. In case of an application failure, the robot continues to transmit sensor and odometry data, which could be used by the orchestration to recover from the application failure by restarting the failed component and initializing it with the last known healthy state and the robot data received since then.

### C. Fallback strategies

Table I gives an overview of the different fallback strategies investigated in this paper. In setup 1, there is no on-board fallback enabled and the robot relies on the velocity commands from the edge container to be correct and received on time. We describe the other setups employing on-board fallback strategies in following paragraphs in detail.

*1) On-board network safestop:* In its simplest form, our on-board fallback approach to mitigate an interrupted connection between robot and edge permanently checks that the last velocity command is valid, i.e., sufficiently up to date. In case the delta $\delta_t(v) = t_{now} - t_{last\_vel}$ between the current time $t_{now}$ and the time the last velocity command has been received surpasses a certain threshold $\varepsilon$, i.e., $\delta_t(v) > \varepsilon$, we

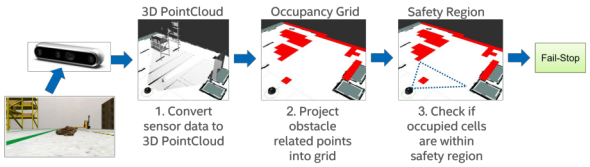
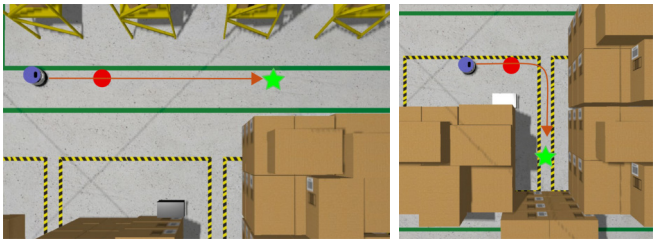



Fig. 2. Overview of the on-board fallback obstacle safety-layer



(a) (b)

Fig. 3. Visualization of our simulation scenarios with the robot being located at its start location. The green star indicates the navigation goal of the robot whereas the red circle indicates the area where the simulated communication loss is triggered.

set the robot's velocity to 0 until an new, up-to-date velocity command is received from the edge container. Although this strategy ensures that the robot is safe and not actively colliding with anything, it impairs the robot's navigation task performance, which could be undesirable.

*2) On-board fallback obstacle safety layer:* To improve the availability of the robot during network connection failures, it is necessary that the robot can continue its driving mission, even if no new velocity commands arrive at the robot. However, if the robot continues to follow an outdated velocity command, collisions with obstacles in the environment can occur. Therefore, we foresee that a safety layer runs locally on the robot. The objective of this safety layer is to detect obstacles within a safety critical distance $d_{safe}$ to the robot and in this case enforce a stop of the robot. For this purpose, we use the safe concept depicted in Fig. 2, that is based on the perception approach proposed in [27], which is available as sample application within Intel's Robotics SDK[1]. On our robots used for evaluation, the perception module uses a RealSense depth camera and creates a costmap containing all safety relevant obstacles. Then, for each obstacle an evaluation is conducted, during which the distance of each object towards the robot is compared against a user-defined safety region around the robot (e.g., defined by the robots stopping distance). If the object is located within this safety region, we enforce a stop of the robot.

As a result of this safety concept, the robot can continue to drive with the last received velocity command until its safety region is violated. In other words, there is no need to stop the robot immediately upon detection of a network failure, instead the robot can continue to operate in a safe manner until it is too close to an object in the environment.

*3) On-board fallback navigation and obstacle safety:* The fallback obstacle safety layer as described in Sec. III-C.2 allows the robot to continue driving even in the presence of a failure of the connection between the robot's on-board and navigation container in the edge. However, without additional on-board navigation capabilities, the robot is doomed to drive with the last know velocity command. Therefore, we introduce a minimalistic on-board fallback controller, that is running in addition and prior to the fallback obstacle safety layer on-board. Similar to network safestop in Sec. III-C.1, we check the temporal difference $\delta_t(v) = t_{now} - t_{last\_vel}$ since the last received velocity command. If this delta is below a certain threshold, i.e., $\delta_t(v) <= \varepsilon$, the incoming velocity command is considered sufficiently up-to-date and is forwarded unaltered to the fallback obstacle safety module. Note that this approach does not increase the robot's on-board computational load and thus energy consumption during normal mode, as the fallback navigation is only activated in case of connection failures. If however, the last velocity command becomes outdated, i.e., $\delta_t(v) > \varepsilon$, the fallback navigation module switches to an on-board navigation mode. During normal operation, the fallback navigation module, in addition to the last velocity command, also stores the last plan, i.e., a sequence of poses $P = (p_i)_{i=1}^{n}$ with $p_i = (x_i, y_i, \theta_i)$, which is transformed from the map frame into the robot's coordinate frame. This transformation is necessary, since once the connection to edge is interrupted no more global localization information are available to the robot. To follow this plan, we employ a variant of the simple pure pursuit controller [28], which determines the closest point far enough from the current robot position $p_{robot}$ using a given lookahead distance $d_L$, i.e., $p_L = p_i \in P$ with $\|p_i - p_{robot}\| \geq d_L$ and $\|p_{i-1} - p_{robot}\| < d_L$. With a given desired linear velocity $v_{lin}$, we can now set the velocity command to $\left(v_x = v_{lin}, v_y = 0, v_\theta = \frac{2 \cdot y_i}{\|p_i - p_{robot}\|}\right)$ and send it to the fallback obstacle safety layer for the collision safety check. Note that for this formulation to work, all coordinates need to be in the robot's base frame. However, while the connection to the edge and thus the robot's localization module is interrupted, only localization based on dead reckoning can be performed. That is, while the wireless communication between the robot and the edge is interrupted, the robot's navigation performance is heavily dependent on the quality of the odometry.

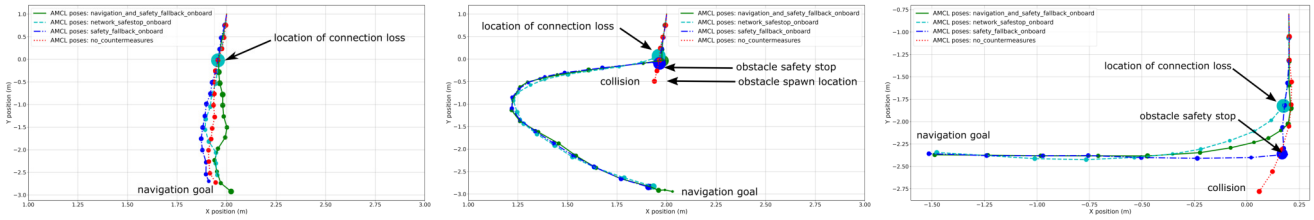
This fallback strategy allows the robot to continue driving by following the last received global plan until its safety perimeter is violated, and thus continue its navigation mission until either the communication disruption is resolved or an unknown obstacle blocks its path.

## IV. EXPERIMENTS

### A. Simulation scenarios

We evaluated our approach in three simulation scenarios within the AWS warehouse environment in Gazebo Ignition

[1]https://amrdocs.intel.com/docs/2.0/index.html




(a) Scenario A (b) Scenario B (c) Scenario C

Fig. 4. Visualization of the robot's trajectory during selected runs with our 4 setups in each scenario. Filled circles indicate the robot's position as estimated by the AMCL localization module. The size of the circles indicates the time since the last AMCL update, i.e., circles with bigger radius show positions where the robot stopped moving.

| ID | Task | Description | Img |
|---|---|---|---|
| A | straight drive 4 m forward | communication loss after 1 m | Fig. 3a |
| B | straight drive 4 m forward | communication loss after 1 m, then spawn box in front of robot | Fig. 3a |
| C | drive around corner into corridor between boxes | communication loss right before corner | Fig. 3b |

TABLE II

OVERVIEW AND DESCRIPTION OF ALL 3 SCENARIOS.

[29], [30]. Fig. 3 depicts the scenarios in a top-down view with the robot (blue circle) being located at its starting position, the green star and red circle indicating the robot's navigation goal and the robot's location when the communication interruption occurred. Fig. 3a shows scenarios A and B, where the robot needs to drive in straight line towards its goal when the communication loss happens (scenario A). In scenario B, the robot has to reach the same goal location as in scenario A, but shortly after the communication is interrupted an obstacle is created in front of the robot. Fig. 3b shows scenario C, where the robot has to drive around a corner into a narrow corridor to reach its navigation goal with the communication interruption being triggered right before the corner.

### B. Experimental Setup

To compare and evaluate the different on-board fallback strategies proposed in Sec. III, we conducted extensive testing with the simulation scenarios defined in Sec. IV-A. We use ROS2 [5] Galactic as middleware and chose the AWS warehouse [30] in Gazebo Ignition Fortress [29], [31] as simulation environment. As described in Sec. III, we focus our experiments on a connection loss of variable length as failure to be investigated. From the robot's perspective, no incoming velocity commands could be either a loss of the communication channel or even an application failure in the edge. To emulate these failures, we deploy an additional ROS2 proxy node inside the robot container that subscribes to a set of user-specified topics (here, sensor messages to be sent from robot to edge and velocity command and plan messages to be sent from edge to robot) and, in normal mode, simply forwards them unaltered. In communication loss mode (triggered via ROS2 service call), this node pauses the transmission for the user-specified duration and thus emulates a communication loss. By only blocking the message sent from the edge-container to the robot, we emulate an application error in the edge including some (non-existing) stateful recovery mechanism where all navigation modules are in an healthy and up-to-date state at the end of the failure. By interrupting the communication in both directions, we emulate a communication failure, where the edge modules continue to run without receiving sensory data and need to handle discrepancies between new incoming sensory data and their outdated internal belief once the communication is restored.

We compare the performance of each of our fallback strategies for a communication loss with a duration of 1 s, 2 s, 8 s and 15 s and conduct 100 runs per experiment to achieve statistical significance.

### C. Qualitative Results

Fig. 4 visualizes one trajectory as estimated by the AMCL module from one selected evaluation run for each of our three scenarios and all four fallback strategy setups with a fixed connection loss duration of 8 s. During the connection loss, the AMCL module in the edge continued to receive sensor and odometry data from the robot during the interrupted connection. Each circle indicates the location where AMCL performed an update, while the size of the circles visualizes the time that has passed since AMCL performed its last update. That is, circles with larger size correspond to locations where the robot did not move. Without any on-board fallback strategies (no countermeasures), the robot continues to drive with the last known velocity command once the failure occurs, whereas with safety fallback enabled, the robot continues driving similarly but stops in case of an imminent collision. In contrast, the network safestop strategy makes the robot stop moving and thus pauses the navigation task once the last velocity command is outdated. Finally, with the navigation and safety fallback strategy, the robot continues following the last known global plan while being protected by the safety fallback avoiding collisions. Although we observe the expected behavior of the fallback strategies in these selected runs, we need to confirm this on a statistically significant amount experiments. Additionally, we observe that when the connection between robot and edge is interrupted in both directions, the localization module in the edge only receives sensory and odometry data after the connection is restored and thus "jumps" to the new position

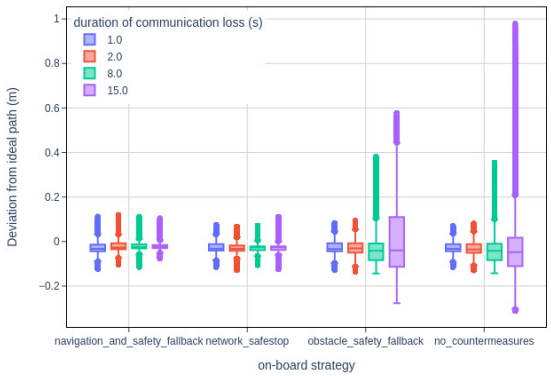



Fig. 5. Evaluation of the deviation from the ideal path in scenario A.

| duration of connection loss | 1 s | | | 2 s | | | 8 s | | | 15 s | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scenario → Setup ↓ | A | B | C | A | B | C | A | B | C | A | B | C |
| 1 | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 100 % | 100 % | 12 % | 100 % | 100 % |
| 2 | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % |
| 3 | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % |
| 4 | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % | 0 % |

TABLE III

AMOUNT OF COLLISIONS IN 100 RUNS PER FALLBACK SETUP AND SCENARIO DEPENDING ON THE DURATION OF THE CONNECTION LOSS.

with high uncertainty and needs some time to recover. This effect occurs particularly in scenario A where the incoming sensory data before and after the lost connection is very similar and thus ambiguous.

### D. Quantitative Results

*1) Collision avoidance:* Firstly, we investigate if the on-board safety fallback strategies are reliably able to protect the robot from colliding with obstacles. Table IV-C shows the amount of collisions for each scenario and setup over 100 experimental runs. As expected, only the setup without on-board countermeasures leads to collisions, particularly for longer connection interruptions, where the robot crashes in every trial in scenario C for loss duration 8 s and 15 s. In contrast, all safety fallback strategies are successfully able to protect the robot from collisions. The 12 % collisions in scenario A for a connection loss of 15 s arise from the robot significantly deviating from the ideal path leading to collisions with shelves in the warehouse (visible at the top of Fig. 3a).

*2) Navigation task performance:* Next, we are also interested in the robot's ability to continue its navigation mission during an unexpected communication failure. Therefore, we evaluate the overall navigation time $T_{nav} = t_{goal} - t_{start}$, where $t_{start}$ and $t_{goal}$ denote the time when the robot started navigating and the time when it reached its navigation goal respectively. Additionally for scenario A (straight driving without unknown obstacles), we also evaluate the deviation from the ideal path, which is a straight line with $y = 2$.

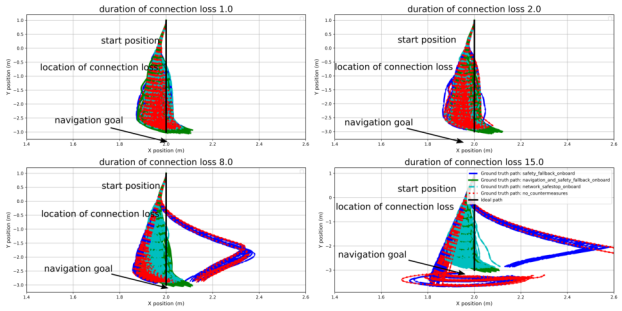
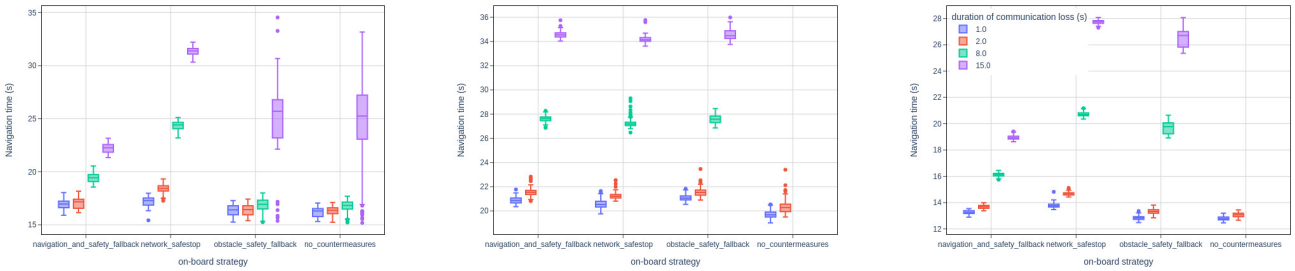



Fig. 6. Visualization of all 100 ground truth trajectories in Scenario A for all evaluated setups in comparison to the ideal path depicted as a black line.

Therefore, we project the (ground truth) 2D positions of the robot's trajectory $P_{robot} = (x_i, y_i)_{i=1}^{n}$ to the ideal path, i.e., $P_{ideal} = (x_i, 2)_{i=1}^{n}$ and calculate the deviation as $d = \hat{p}_i - p_i$ with $\hat{p}_i \in P_{ideal}$ and $p_i \in P_{robot}$.

Fig. 5 shows the deviation from the ideal path in scenario A over 100 experimental runs for each setup, whereas Fig. 7 shows the navigation time for all on-board setups in each scenario. For both, the network safestop and the fallback navigation and safety strategy, we observe a constantly low deviation from the ideal path independent from the duration of the connection loss, whereas the deviation increases significantly for the other strategies (1 and 3) for longer connection interrupts. This behavior is due to the fact, that these strategies continue driving during the interrupted connection with the last known velocity command. If this contains an angular component or if the robot is slightly rotated away from the navigation goal pose, the robot constantly moves away from the ideal path and thus the deviation accumulates over the duration of the interrupted connection. Fig. 6 visualizes this behavior showing all 100 (ground truth) trajectories for scenario A with significant deviations from the ideal path for strategies 1 and 3 while having to drive for longer duration without valid velocity commands from the edge-container. In contrast, the network safestop strategy stops once the connection is interrupted and only continues to drive once the robot receives up-to-date velocity commands from the edge container again, which results in a low path deviation but also increased navigation times (cf., Fig. 7a). Similarly, the fallback navigation and safety strategy continues to drive along the last known global plan using odometry-based localization and is thus able to achieve constantly low deviations from the ideal path combined with smaller navigation times compared to the network safestop strategy.

Fig. 7 depicts the time the robot takes to reach its navigation goal as box-plots over 100 evaluation runs for each scenario (Fig. 7a, 7b, and 7c show the results for scenario A, B and C) and fallback strategy ($y$-axis of the plots in Fig. 7). For scenario B, we observe almost identical task performance for all fallback strategies, as they all need to stop the navigation task either due to the connection loss or due to the unknown obstacle spawned in front of the robot after the connection to




(a) Scenario A (b) Scenario B (c) Scenario C

Fig. 7. Evaluation of the navigation time for varying durations of the connection interruption for our on-board fallback strategies over 100 experimental runs per strategy.

the edge is interrupted. Without a safety fallback, the robot collides with the unknown obstacle for the longer durations of the connection loss. For scenario A, the navigation time without a safety fallback and the obstacle safety fallback strategy remain almost constant across varying times of the connection loss, expect for the largest duration. We already observed, that this increased navigation time corresponds to the aforementioned large deviation from the ideal path. The navigation time of the network safestop strategy increases corresponding to the duration of the connection loss. The navigation time of the fallback navigation and obstacle safety strategy increases slightly (proportionally to the duration of the connection loss) due to the robot driving with a slightly lower velocity during the connection loss. Finally for scenario C, we observe that the navigation time for the network safestop and obstacle safety fallback strategies increases proportionally to the duration of the connection loss as the robot again needs to stop the navigation task either due to the connection loss or due to an imminent collision with the boxes in front of the robot after the connection to the edge is interrupted. Only the navigation and obstacle safety fallback strategy is able to fulfill its navigation mission in consistently low time even for the longer duration of the interrupted connection. As mentioned before, reason for the slightly increased navigation time for a longer duration of the drop is the (user-specified) lower velocity with which the robot navigates while the connection is interrupted.

Note, that all results and discussions presented in Sec. IV-D referred to an interrupted connection in both directions. Although we observed a difference in the robot's behavior in selected runs of scenario A, there was no statistically significant difference regarding navigation time or deviation from the ideal path when interrupting the connection only from robot to edge or in both directions.

## V. DISCUSSION

### A. Conclusion

In this paper, we have introduced three different on-board fallback safety strategies to handle connection failures when deploying AMR navigation in the edge instead of the robot's on-board compute. Such deployment schemes are prone to failures in the (mostly wireless) connection between the on-board and offloaded algorithmic components. Through extensive testing in simulation, we have demonstrated that with a minimal on-board fallback navigation component combined with an obstacle safety layer, the robot is reliably able to mitigate failures in the connection between robot and edge/cloud. Even over a longer duration of an interrupted connection, the robot is able successfully avoid collisions and continue its navigation mission. We demonstrated the feasibility of our approach through extensive testing in a simulated warehouse environment in three relevant use-case scenarios. It is worth noting however, that the improvements in terms of reduced navigation time compared to the simpler network safestop during long running navigation tasks heavily depends on how often a communication failure actually occurs. For instance, if the communication fails only once for 10 s during a 8 h navigation task, the percentage by which the navigation time increases using the simple network safestop over the fallback navigation is less significant compared to twenty 10 s communication failures during a 5 min navigation task.

### B. Future Work

In this paper, we have shown how to mitigate failures in the connection between the low-level control in the on-board robot container and the navigation stack in the edge container. In future work, we aim to mitigate other failures such as errors of the algorithmic components running in the edge and how to mitigate such failures through a stateful handover, i.e., continuously storing the last healthy state of the edge-components. The failing component can then be recovered by either triggering a complete restart or running a fallback variant in the background that takes over upon failure and providing the stored state of the failed component. Such a stateful handover in combination with the on-board fallback strategies proposed in this paper will increase safety, availability and stability of edge-based AMR navigation. Another option for future work would be to incorporate the safety grid of the obstacle safety-layer into the fallback controller to avoid unmapped obstacles to further improve the on-board fallback navigation during failures. Finally, we aim to investigate the safety and orchestration aspects of scaling up to a multi-robot system with collaborative algorithms in the edge and/or supporting the AMRs through static sensors in the infrastructure.

## REFERENCES


[1] A. Baxi, M. Eisen, S. Sudhakaran, F. Oboril, G. S. Murthy, V. S. Mageshkumar, M. Paulitsch, and M. Huang, "Towards factory-scale edge robotic systems: Challenges and research directions," *IEEE Internet of Things Magazine*, vol. 5, no. 3, pp. 26–31, 2022.

[2] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A Survey of Research on Cloud Robotics and Automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[3] J. Gao and D. Wang, "Robot Cloud Computing and Al Services - State-of-the-art Solutions, Challenges, and Needs," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2022, pp. 105–116.

[4] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The Marathon 2: A Navigation System," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020. [Online]. Available: https://navigation.ros.org/

[5] 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, May 2022.

[6] Pingdom. (2023) Average Cost of Downtime per Industry. [Online]. Available: https://www.pingdom.com/outages/average-cost-of-downtime-per-industry/#:~:text=In%20the%20manufacturing%20industry%2C%20the,tool%20breaks%2C%20adjustments%2C%20etc.

[7] IIoT-World. (2023) The actual cost of downtime in the manufacturing industry. [Online]. Available: https://www.iiot-world.com/predictive-analytics/predictive-maintenance/the-actual-cost-of-downtime-in-the-manufacturing-industry/

[8] K. Cao, S. Hu, Y. Shi, A. W. Colombo, S. Karnouskos, and X. Li, "A survey on edge and edge-cloud computing assisted cyber-physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7806–7819, 2021.

[9] J. Zhang, F. Keramat, X. Yu, D. M. Hernández, J. P. Queralta, and T. Westerlund, "Distributed Robotic Systems in the Edge-Cloud Continuum with ROS 2: a Review on Novel Architectures and Technology Readiness," in *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*, 2022, pp. 1–8.

[10] A. K. Bozcuoglu, G. Kazhoyan, Y. Furuta, S. Stelter, M. Beetz, K. Okada, and M. Inaba, "The Exchange of Knowledge Using Cloud Robotics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1072–1079, Apr. 2018.

[11] P. Huang, L. Zeng, K. Luo, J. Guo, Z. Zhou, and X. Chen, "CollaSLAM: Real-Time Multi-Robot Collaborative Laser SLAM via Edge Computing," in *2021 IEEE/CIC International Conference on Communications in China (ICCC)*, 2021, pp. 242–247.

[12] B. Hu and R. Yang, "A KubeEdge-based Multi-Robot Collaboration Framework for Perception, Planning and Navigation," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 1186–1191.

[13] L. Girletti, M. Groshev, C. Guimaraes, C. J. Bernardos, and A. de la Oliva, "An Intelligent Edge-based Digital Twin for Robotics," in *2020 IEEE Globecom Workshops GC Wkshps*. IEEE, Dec. 2020.

[14] S. Chinchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, and M. Pavone, "Network offloading policies for cloud robotics: a learning-based approach," *Autonomous Robots*, vol. 45, no. 7, pp. 997–1012, July 2021.

[15] S. Alirezazadeh and L. A. Alexandre, "Optimal Algorithm Allocation for Single Robot Cloud Systems," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 324–335, 2023.

[16] F. Lumpp, F. Fummi, H. D. Patel, and N. Bombieri, "Containerization and orchestration of software for autonomous mobile robots: a case study of mixed-criticality tasks across edge-cloud computing platforms," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 9708–9713.

[17] Y. Zhang, C. Wurll, and B. Hein, "KubeROS: A Unified Platform for Automated and Scalable Deployment of ROS2-based Multi-Robot Applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9097–9103.

[18] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. Llontop, D. Xu, C. Buscaron, J. Kubiatowicz, I. Stoica, J. Gonzalez, and K. Goldberg, "FogROS2: An Adaptive Platform for Cloud and Fog Robotics Using ROS 2," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5493–5500.

[19] M. Groshev, G. Baldoni, L. Cominardi, A. de la Oliva, and R. Gazda, "Edge robotics: are we ready? an experimental evaluation of current vision and future directions," *Digital Communications and Networks*, May 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864822000888

[20] M. Balogh, A. Vidács, G. Fehér, M. Maliosz, M. Á. Horváth, N. Reider, and S. Rácz, "Cloud-Controlled Autonomous Mobile Robot Platform," in *2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2021, pp. 1–6.

[21] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[22] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, mar 2014. [Online]. Available: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

[23] A. Jeffery, H. Howard, and R. Mortier, "Rearchitecting Kubernetes for the Edge," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 7–12.

[24] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*, ser. Intelligent robotics and autonomous agents. The MIT Press, 2005.

[25] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[26] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[27] C. Buerkle, F. Oboril, J. Burr, and K.-U. Scholl, "Safe Perception-A Hierarchical Monitor Approach," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 71–78.

[28] S. Macenski, S. Singh, F. Martín, and J. Ginés, "Regulated pure pursuit for robot path tracking," *Autonomous Robots*, vol. 47, no. 6, pp. 685–694, 2023.

[29] Open Robotics. (2023) Gazebo Ignition. [Online]. Available: http://gazebosim.org/home

[30] AWS Robomaker. (2023) AWS Robomaker Small Warehouse Gazebo World. [Online]. Available: https://github.com/aws-robotics/aws-robomaker-small-warehouse-world

[31] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.