UK Atomic Energy Authority

UKAEA-RACE-PR(25)05

Michal Staniaszek, Tobit Flatscher, Joseph Rowell, Hanlin Niu, Wenxing Liu, Yang You, Matthew Gadd, Matıas Mattamala, Alex Schutz, Daniele De Martini, Luke Pitt, Robert Skilton, Maurice Fallon, Nick Hawes

AutoInspect: Towards Long-Term Autonomous Inspection and Monitoring

AutoInspect: Towards Long-Term Autonomous Inspection and Monitoring

Michal Staniaszek, Tobit Flatscher, Joseph Rowell, Hanlin Niu, Wenxing Liu, Yang You, Matthew Gadd, Matıas Mattamala, Alex Schutz, Daniele De Martini, Luke Pitt, Robert Skilton, Maurice Fallon, Nick Hawes

This is a preprint of a paper submitted for publication in IEEE Transactions on Field Robotics

Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version XX Month, XXXX.

Digital Object Identifier 10.1109/XXXX.2022.1234567

AutoInspect: Towards Long-Term Autonomous Inspection and Monitoring

Michal Staniaszek [1], Tobit Flatscher [1], Joseph Rowell[1], Hanlin Niu [2], Wenxing Liu[2], Yang You[2], Matthew Gadd[1], Matías Mattamala[1], Alex Schutz[1], Daniele De Martini[1], Luke Pitt[1], Robert Skilton[2], Maurice Fallon[1], Nick Hawes[1]

[1]Oxford Robotics Institute, Department of Engineering Science, University of Oxford, Oxford, UK
[2]Remote Applications in Challenging Environments (RACE), United Kingdom Atomic Energy Authority, Culham, UK

Corresponding author: Nick Hawes (email: nickh@robots.ox.ac.uk).

Initial work was supported by the Innovate UK AutoInspect grant (1004416). Further work was supported by the EPSRC Programme Grant "From Sensing to Collaboration" (EP/V000748/1), the UKAEA/EPSRC Fusion Grant (EP/W006839/1), and part of the work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are those of the author(s) and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

ABSTRACT This paper presents a technical overview of *AutoInspect*, a ROS-based software system for robust and extensible mission-level autonomy. Over the past three years we have deployed AutoInspect on multiple platforms in a variety of environments, from forests to fusion reactors, for durations ranging from hours to weeks. The AutoInspect system combines robust mapping and localisation with graph-based autonomous navigation, mission execution, and scheduling, into a complete autonomous inspection and monitoring system. Code for the graph-based autonomy component will be made available at ori.ox.ac.uk/projects/autoinspect. In this article we describe in detail our most commonly-used configuration of AutoInspect: a Boston Dynamics Spot fitted with a custom compute and sensing payload, called *Frontier*. To highlight the flexibility of the AutoInspect system, we also describe its deployment with different hardware and software configurations. We evaluate AutoInspect's performance in two long-term deployments on Spot at the Culham Centre for Fusion Energy in Oxfordshire, UK. The first deployment took place at a robotics test facility, spanning 49 days, including 14 uninterrupted days of autonomous operation. The second deployment covered 35 days in the torus hall of the Joint European Torus (JET) fusion reactor, and achieved 15 days of uninterrupted operation. This was the first ever deployment of a fully autonomous mobile robot in a fusion facility.

INDEX TERMS Inspection, Simultaneous Localisation and Mapping, Long-term Autonomy

I. Introduction

The increasing physical capabilities of commercially-available robot platforms has opened up new applications for mobile autonomy, and also allows research in planning and mapping to be applied in a much wider range of industrial settings than was previous possible. A use-case of major industrial importance for this technology is *autonomous inspection*, where a robot has to monitor a set of sites of interest over a long period of time. Creating a robotic system capable of performing autonomous inspection requires a generic, modular and scalable software system that can integrate multiple components such as perception, decision-making, planning and control. To meet this need we have developed *AutoInspect*, a flexible software system for autonomous inspection and monitoring.

Our goal with AutoInspect was to create an autonomy system which can perform a wide variety of inspection and monitoring tasks, for long periods of time, on a range of robot platforms. Requirements for the system were driven by interactions with industry parters as well as test deployments in a variety of environments, including: an operational chemical plant, a decommissioned nuclear power plant, a mine,

FIGURE 1. **Spot deployments during development of AutoInspect. *Top left*: October 2021, climbing stairs at Fire Service College mock oil rig during first fully autonomous mission. *Top right*: May 2022, short-term construction site deployment. *Bottom left*: May 2023, change detection testing at Fire Service College urban search and rescue facility. *Bottom right*: January 2024, measuring gamma radiation during long-term deployment at the Joint European Torus.**

a mock oil rig, and around partially demolished buildings at an outdoor training ground for firefighters. Some of these are shown in Figure I.

To meet these requirements AutoInspect integrates two major elements: *Frontier*, a mapping and localisation system packaged in a fixed hardware payload; and *Toplogical Autonomy* (TA), a framework for spatial abstraction and mission planning. The mapping subsystem that runs on the Frontier brings together our modular odometry system VILENS [1] with pose-graph LiDAR SLAM (see Section III). The TA spatial abstraction and mission planning subsystem is a robot-agnostic graph-based autonomy framework (Section IV). Code for TA will be made available through our website[1]. Figure 2 shows an overview of AutoInspect as a whole, which is implemented using ROS Noetic [2].

A key enabler of AutoInspect is our *Frontier* hardware, shown in Figure 3. Weighing approximately 1.5kg, It combines sensors for mapping and localisation with a small form-factor computer in a 3D printed case, and runs the entire AutoInspect system. With Frontier, AutoInspect can be integrated with any platform which provides power and a connection to control its motion. This platform neutrality is a key benefit of AutoInspect. The graph-based navigation system within the TA subsystem provides a standard interface to integrate robot-specific navigation commands. This inteface can be used to define a variety of motion behaviours for use during navigation between graph nodes, e.g. door traveral or stair climbing. TA's mission planning component provides interfaces for integration of custom actions to be performed by the robot, such as capturing images or making sensor measurements. Although our work is agnostic of the hardware platform being controlled by AutoInspect, the majority of the work presented in this article uses the Boston Dynamics Spot[2] platform. We selected Spot as our primary development platform for its robust navigation, stair climbing, and autonomous charging capabilities.

In the following sections, we describe AutoInspect in detail, covering mapping and localisation in Section III and the TA system in Section IV. In Sections VI and VII we descibe how we integrate AutoInspect with Spot, and present

[1] ori.ox.ac.uk/projects/autoinspect

[2] bostondynamics.com/products/spot

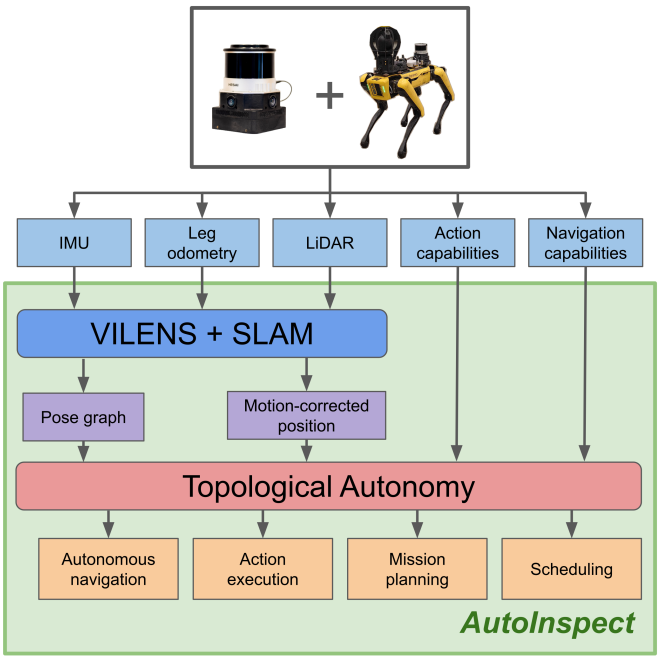

FIGURE 2. **Overview of the *AutoInspect* system deployed on Spot, using the Frontier payload. IMU measurements and LiDAR data from Frontier, as well as leg odometry from Spot are used by VILENS to generate a continuous pose estimate of the robot and a SLAM pose graph. These inputs, was well as Spot's action and navigation capabilities, are used by the topological autonomy system to provide autonomous navigation, action execution, mission planning, and scheduling capabilities to the operator.**

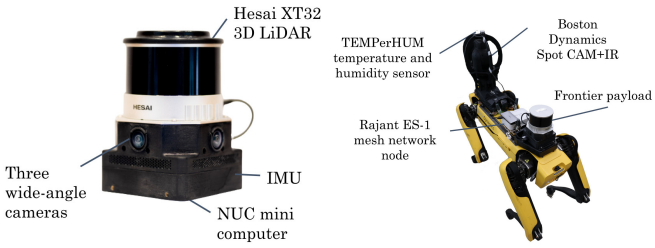

FIGURE 3. ***Left*: Our autonomy payload, *Frontier*, consisting of a Hesai LiDAR, three Sevensense Alphasense cameras, and an IMU. *Right*: Our primary deployment platform, Boston Dynamics Spot, with the payloads used for deployments at UKAEA. The Frontier runs the entire *AutoInspect* software stack and sensor drivers.**

results from two long-term deployments of this system. Section VIII gives an overview of AutoInspect integrations and deployments with other robot platforms.

II. Related Work

A. Robotic Inspection Systems

Early surveys on robotic systems for transmission line [3] and pipe [4] inspection, detail the different approaches to robot morphologies required for these tasks. This highlights the degree of hardware specialisation and complexity required for certain inspection tasks. This is in contrast to more recent surveys on the robotic inspection of infrastructure [5, 6] which show a trend for systems based on general-purpose mobile platforms such as aerial vehicles (UAVs) and unmanned ground vehicles (UGVs). In these cases the complexity comes from the design and integration of the sensor suite, rather than the design of the platform itself. These surveys also note the limited autonomy capabilities provided by existing systems, something AutoInspect aims to address.

AutoInspect may appear superficially similar to the *platform-specific* inspection solutions implemented for the ANYbotics ANYmal[3][4] [7, 8] and Spot[5]. This is because its development was guided by similar requirements. However, in contrast to these and other inspection systems [5, 9, 10], AutoInspect is not committed to any particular robot or inspection objective.

B. Odometry, Mapping, and Localisation

Mobile robots use a variety of sensors to position themselves within their operational environment and to perceive obstacles. Here we broadly distinguish between the three different positioning tasks of odometry, localization and SLAM (Simultaneous Localization and Mapping) and give a brief summary of relevant systems.

Odometry is concerned with tracking the continuous relative pose (position and orientation) of the robot. This is usually with respect to an arbitrary fixed origin and used for the purpose of closed loop control and local path following. Initial odometry systems used wheel odometry for dead reckoning in 2D. Legged robots are prominently used in this work and example legged robot motion estimators include TSIF [11] and Pronto [12] which fuse foot and joint sensing as well as IMU readings often using an Extended Kalman Filter (EKF) in full 3D. These systems aim to achieve low-latency, low-drift proprioceptive motion estimates.

Exteroceptive odometry systems incorporate environment sensing with vision and LiDAR being the most common modalities. Both visual and LiDAR odometry have been extensively studied. Our system uses LiDAR inertial odometry (LIO) to construct building-scale 3D point cloud maps. A key breakthrough in this field was the LOAM system by Zhang *et al.* [13] which leveraged inertial sensing to correct motion effects during scanning as well as detecting features in LiDAR scans to achieve accurate frame-rate motion tracking. Subsequent systems such as Fast-LIO2 [14] and WildCat [15] have demonstrated impressively accurate performance, e.g. 1m drift per 1km travelled in benign conditions. A common way to achieve robustness is to fuse these various modalities. In our case we use the odometry estimate of the plaform being controlled by AutoInspect (e.g. the leg odometry from Spot) as a motion prior for a second stage LiDAR odometry system. This helps to reduce the risk of LiDAR scan degeneracy in constrained environments.

[3] anybotics.com/robotics/anymal
[4] anybotics.com/robotics/robot-capabilities
[5] bostondynamics.com/products/orbit

Inevitably, odometry systems will accumulate incremental drift which will lead to the map representation becoming inconsistent. To address this problem, simultaneous localization and mapping (SLAM) systems aim to correct for odometry drift by building a single consistent global map which relies on recognising when a robot returns to a previous location. A common approach to solve this problem is pose graph optimization [16] which builds a graph of nodes corresponding to samples of the robot's odometry state. When a location is re-recognised, the graph can be adjusted in real-time to maintain map consistency and to correct for odometry drift. A summary paper for the 2021 DARPA Subterranean challenge presents a detailed overview of the research frontiers of multi-sensor odometry and SLAM more broadly [17]. The systems of the winning teams in that competition are representative of most mature LiDAR-focused SLAM systems.

Localization involves positioning the robot within a prior map of a (largely static) environment built using SLAM. This topic can be divided between global and local localization. In our work we focus on local localisation, and refer the reader to exemplar systems for global localization: NetVLAD [18] and ScanContext++ [19] Local localization aims to update a precise running position estimate in the map at sensor frame rate which can be directly used to make mission planning decisions. Early 2D systems using sonar and single-beam LiDARs, lacked the ability to uniquely resolve a full position estimate from a single LiDAR scan — especially in the presence of scene change and dynamics around the robot. To overcome this problem, common systems including the default ROS navigation stack, rely on Monte Carlo localization [20] to fuse their platform odometry estimate with partial position measurements into a recursive Bayesian position estimate. Many modern inspection robots now use 3D multi-beam LiDARs to achieve full 6 DOF localisation within a 3D point cloud map using a single LiDAR scan and ICP registration without needing to rely on a Bayesian estimator to fuse measurements [8]. These systems often assume that there is sufficient structure close to robot to ensure localization reliability but may also rely on introspection and global place recognition to recover from registration failures.

C. Mobile Robot Autonomy

Mobile robot autonomy often consists of a variety of systems for obstacle avoidance, local and global motion planning, task execution, and mission planning. We assume access to low-level local motion planning and obstacle avoidance, and focus on high-level autonomy for navigating and acting. Topological map representations are good at representing large physical spaces [21], and since their introduction in this context by Brooks [22] have been used extensively on mobile robots as an abstraction for laser-based [23–26] and visual [27–30] navigation and mapping. The topological map is well suited for incorporating domain knowledge during deployments and can be used to adapt the system's behaviour. It is also a natural representation for advanced planning and resource allocation algorithms, which is important for our research. Topological maps have been used as a basis for planning and navigation in field deployments in offices [26, 31–33] and agriculture [34]. In addition to its technical benefits, a topological representation is also an intuitive model for end users, providing a clear visualisation of the structure and bounds of the robot's operational area, and the ability to label locations in the environment is useful when communicating about missions. Topological maps can be constructed in many ways, such as from aerial images [35] and 2D or 3D maps [36–38]. We use a hybrid approach, automatically generating an initial map based on the SLAM pose graph from global map construction, then use a graphical user interface (GUI) to manually adapt it to the environment and mission specification.

Long-Term Autonomy (LTA) has long been recognised as a challenge for mobile robots [39], and is a target for our work on AutoInspect too. Much literature on LTA describes tour guide robots [40–44], which navigate the environment and present information to tour participants. The most significant recent LTA projects in terms of complexity are the CoBots [45] and STRANDS [26] projects. The CoBots [45, 46] operated in a university office building to schedule tasks such as visitor escort, remote telepresence, and object transport on a team of mobile robots, with a task scheduler distributing these tasks between deployed robots. The robots travelled over 1000km in 3100 deployments, over the course of 1280 hours of deployment. As such, each individual deployment was relatively short, while our aim is to build a system for *uninterrupted* autonomy over long periods of time. One of the primary aims of the STRANDS project was uninterrupted long-term autonomy [26]. Operating in offices and a care home, the robots ran for a total of 104 days over 43 runs, the longest being a single uninterrupted run of 28 days. The project developed two performance measures for LTA: total system lifetime (TSL) measures how long the system is available for autonomous operation; and autonomy percentage (A%) the proportion of the TSL during which the system was actively performing tasks autonomously, an important metric of system utilisation. The reliability of platforms such as ANYmal and Spot, along with their autonomous docking capabilities, has lowered the barrier to entry into LTA in industrial environments. However, as mentioned above, published work in this area is limited, likely due to commercial sensitivity.

Some LTA projects use human interventions to augment autonomous recovery behaviours, and Meeussen *et al.* argue that so long as human intervention is not too frequent, this method can be very effective in maintaining uptime [47]. The STRANDS project's monitored navigation system [26] sent requests for help from humans in situations where the autonomy system was unable to recover. Reliability engineering is a significant hurdle to more systems with LTA capabilities as software maintenance and management

and complex interfacing tends to be brittle [48]. In our work, we track interventions as a measure of the robustness of autonomy, as part of the reliability engineering process, identifying and improving parts of the system which cause interruptions to autonomy. This is a key part of the iterative process of refinement for LTA systems.

D. High-Level Autonomy

High-level autonomy involves control and coordination of a robot's capabilities to fulfil tasks specified by operators, or generated by the autonomy system itself. A common feature of approaches to high-level autonomy is the use of hierarchies to abstract away some of the complexity. Frameworks like SMACH[6] and RAFCON[7] facilitate the construction of hierarchical finite state machines for this purpose, which can be coordinated to build complex behaviours. While powerful, a weakness of state machine approaches is that as complexity increases, it can be challenging to modify existing components or integrate new ones as the complex interlinking of components can cause unexpected results. Behaviour trees are an alternative approach which takes advantage of the hierarchial nature of trees [49]. They can be used to build behaviours which are more complex, but nonetheless easier to understand due to their modularity. Other architectures such as ROSPlan [50] make use of more formal planning techniques, using the Planning Domain Definition Language (PDDL) to hook into existing classical planning solvers. This requires explicit definition of the domain and the goal of the system, which may be challenging if the aim is to perform different types of tasks. AutoInspect is not intended to replace any of these approaches. As tasks are abstracted in our topological autonomy system, it is possible to use any of these types of frameworks to define tasks and missions. The primary goal of AutoInspect's topological autonomy system is to link a topological representation of the environment with task execution, rather than to define a specific model of high-level autonomy. It can be used as a simple high-level autonomy framework, or integrated into an existing system as another layer of abstraction.

Autonomy systems may have varying degrees of human supervision, and thus lie on a spectrum from supervised to unsupervised. Human supervision was a key part of the DARPA SubT challenge, in which robot teams were deployed for 60 minutes in unknown subterranean environments with the goal of building maps and discovering artifacts [51–54]. Participating teams were allowed only a single operator controlling the system. With most teams deploying more than 4 robots at once, the operator was under very high cognitive load. The robots thus had a high degree of autonomy, with strictly controlled access to the operator's attention. The challenge involved highly complex and challenging environmental conditions, time pressure, and active operator interaction. In contrast, AutoInspect is aimed at long-term operation in more controlled environments. Our goal is to minimise the need for operator interaction, by creating a robust system that can operate independently for long periods, using a schedule for mission execution provided by operators.

[6] github.com/ros/executive_smach

[7] rafcon.readthedocs.io

III. Mapping, Localisation, and Change Detection

When operating autonomously in an industrial facility, our system localises in a prior map of the environment. While the map can be generated from existing 3D LiDAR scans taken with a terrestrial LiDAR scanner; we typically create a map from scratch by running our SLAM system while teleoperating the robot. In this section we describe the real-time SLAM system used to build the prior map, how we localise within the map, as well as a more recent system component which carries out 3D object-level change between successive missions. Previous applications of this system include forest inventory [55] and aerial inspection of tall buildings [56].

A. Multi-sensor Odometry

The first module in the navigation system is VILENS [1], a multi-sensor odometry system which can fuse IMU, leg odometry, vision, and LiDAR data. VILENS has several configurations. In this project, we maintain a windowed factor graph which smooths the IMU measurements and relative odometry constraints from an ICP-based module based on `libpointmatcher` [57] (running at 3Hz). The motion of the robot can be dynamic and jerky. To achieve the most accurate maps it is important to correct the motion of the LiDAR during each scanning sweep.

When localizing in a prior map, we can instead leverage the leg odometry estimate provided by the Boston Dynamics API. We use this motion estimate as a prior for a simple ICP odometry system. This is preferable because the windowed factor-graph can be computationally expensive.

Because the robot operates in narrow and confined locations, odometry failure due to degeneracy is a possibility. Our current system leverages a LiDAR with a wide vertical field-of-view (104.2 degrees), as well as the point clouds produced by the Spot's depth cameras, to ensure that a well-defined point cloud can be observed.

B. Building a Prior Map with SLAM

The odometry system passes a steam of registered LiDAR scans to our real-time SLAM system, VILENS SLAM. At a spacing of about one metre, we sample the odometry stream and add a new odometry node to a pose graph. By spatially sampling these nodes, we ensure that the map has uniform density even if the robot stands still for periods during the mapping phase. It also aids the search for independent loop closures (described below). Nonetheless, mild odometry drift will occur which necessitates correcting the pose graph using pose graph optimization with the iSAM2 solver [58].

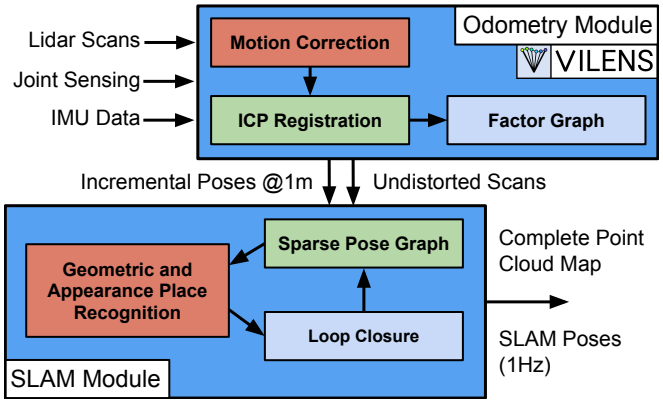

FIGURE 4. **Overview of the VILENS odometry and VILENS SLAM system. On Spot, AutoInspect uses leg odometry as an additional input to the odometry module.**

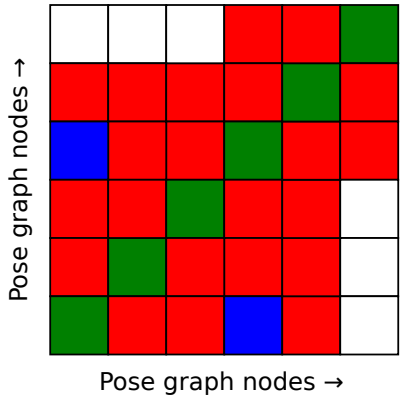

FIGURE 5. **Illustration of the pose graph density check. A connectivity matrix is built using the odometry edges (the green diagonal) and loop closure edges (blue). A dilation operation declares several more cells as being occupied (red). After this, only the unoccupied cells (white) are considered for potential loop closures - greatly reducing unproductive computation.**

In a separate thread, the SLAM system repeatedly searches for loop closures. In a small facility, using loop proposals drawn from the existing pose-graph is sufficient. For larger environments we rely on place recognition to propose potential loop closures. We have used visual place recognition systems as well as lidar systems such as ScanContext [59] and Logg3dNet [60]. Given a suitable loop proposal, we refine the proposal using ICP refinement. If the refinement is successful, we add a loop closure constraint to the pose graph. An overview of the system is shown in Figure 4.

Searching for loop closures across a graph with perhaps 1000 nodes requires judicious use of computation. After a loop proposal has been proposed, each ICP loop refinement takes several hundred milliseconds, meaning that brute force pair-wise search is not feasible. When given a potential loop closure pair, we first determine if the loop closure would be helpful using the density of the pose graph, and only then attempt ICP loop refinement.

To implement this test of loop closure value, we build an matrix corresponding to the pose graph connectivity. We consider the two nodes which form an edge in the pose graph as a pair. The set of pair-wise constraints are then treated as pixels in the connectivity matrix. Next we dilate this matrix by several pixels — corresponding to about five adjacent odometry or loop closure edges as illustrated in Figure 5.

To test if a loop closure between two nodes (P, Q) would help to improve the pose graph, we determine if the corresponding pixel lies close to an existing edge using the connectivity matrix. If it does, we abandon this loop closure proposal because adding it would not significantly affect the graph and might actually over-constrain it. In doing so we avoid carrying out computation in well-constrained parts of the graph. In addition to this check we also take care to ensure that potential loop closures are consistent with one another. For this we carry out pairwise consistency checks to determine if two consecutive potential loop closures agree well with one another. More details about these filters and checks can be found in [55].

The output of the mapping step is the pose graph with corresponding individual pointclouds, as well as a global map in which all individual clouds have been registered in a global reference frame. Figure 6 shows the global map from the JET deployment described in Section VII.

C. Subsequent Localisation in Prior Map

To re-localise in the global prior map we again use ICP which requires an accurate initial pose estimate to be provided by the operator or by initialising the robot in a known location. The pose estimate is then iteratively updated using the leg odometry motion prior and ICP to the prior map at 2Hz.

An alternative approach is to localise in a prior map made up of the individual pose-graph pointclouds using place recognition (for example ScanContext) to determine the pose guess at each iteration. This approach does not require an initial pose estimate and can be particularly effective in very large environments where performing ICP to a single facility-sized pointcloud map would be computationally prohibitive.

D. 3D Change Detection

AutoInspect incorporates LiSTA [61], a 3D change detection system. It uses volumetric differencing to detect object-level changes between pointclouds acquired across different missions. During each mission, a set of local pointclouds are acquired and converted into octrees. Octrees corresponding to the same spatial area, but from different missions, are then compared to generate a set of *difference octrees* which are projected back into the original pointcloud. The pipeline includes ground filtering via RANSAC, Moving Least Squares smoothing, and morphological opening to obtain a set of pointcloud clusters. The pointcloud clusters for each discovered object are then segmented out using

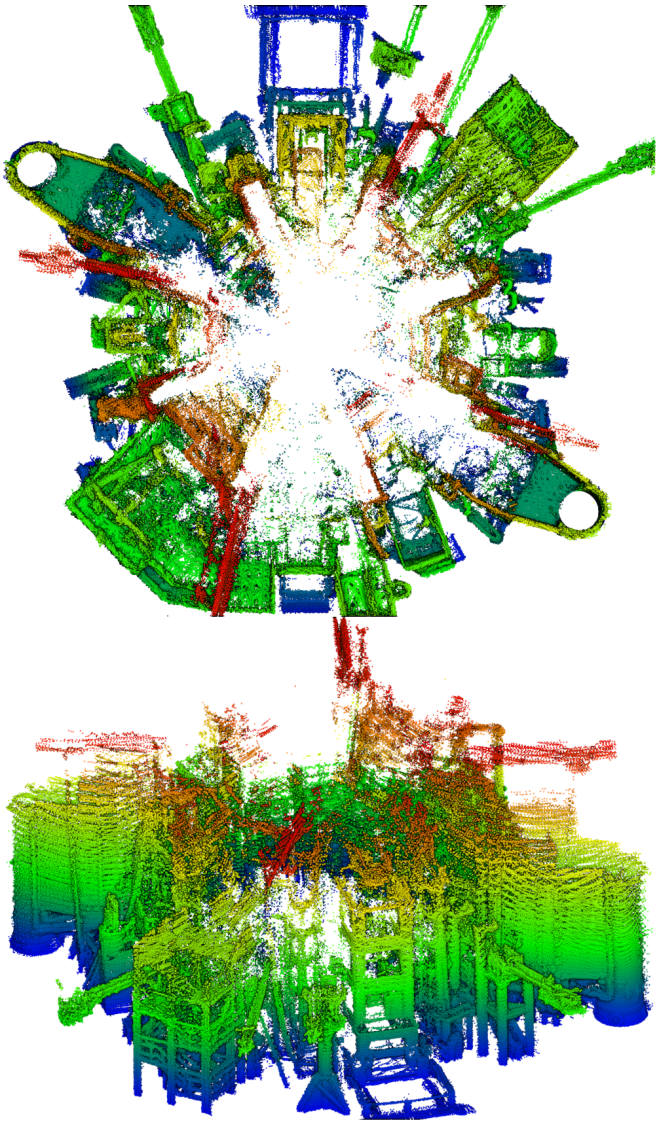

FIGURE 6. **Top-down and side view of the central JET structure cropped from the full SLAM pointcloud. The structure is 30m across on its widest axis, and a has a height of 15m. Due to the height of the robot and the field of view of the LiDAR, upper parts the structure receive sparse coverage.**

Euclidean clustering. Finally, inter-mission correspondences of objects are determined through K-means clustering of SE(3) invariant descriptors assigned to each object using a learning-based 3D pointcloud descriptor [62]. This provides actionable insights to operators in the form of pointclouds of object-level changes to the environment between missions. An application of the system is shown in Figure 15.

IV. Topological Autonomy

In this section we describe the second major component of AutoInspect: the Topological Autonomy system. Implemented in ROS, it provides autonomous navigation and task execution capabilities. At its core is a topological map representation. In combination with the mapping and localisation system, it allows the robot to act in its environment without the need for continuous monitoring or input from operators. The autonomy system is designed with flexibility in mind to facilitate integration with a variety of platforms. Figure 7 shows an overview of the system. The initial topological map is constructed based on the SLAM pose graph. The robot is localised in the topological map based on the closest node to the 3D pose received from the localisation system. The system uses the robot's navigation capabilities through the navigation interface. The robot's action capabilities are registered with the action register, which can call the action executor to execute them. Mission execution navigates and executes tasks based on mission specifications. The scheduler can execute missions on a schedule, subject to interrupts from system monitors. The operator typically controls the robot by scheduling or executing missions. The topological map, graph localisation, and the robot's navigation and action capabilities are integrated differently depending on the specific platform. We give an overview of integration on a variety of platforms in Sections VI and VIII.

A. Topological Map

The core of the autonomy system is the topological map, which is a graph of nodes and edges, examples of which are shown in Figures 8 and 14. A node is a location in the environment and is both a waypoint for navigation and a location at which tasks may be executed. Edges are used to connect one node to another, indicating direct traversability between them. We collectively refer to nodes and edges as map components.

The base topological map is an abstraction of the environment and does not contain spatial information. This was done to avoid committing to a particular spatial model, and has allow topological autonomy to be used on both systems with global metric maps, and non-visual teaching-and-repeat systems which use a non-metric local spatial model. To add spatial (and other) information to map components, we use *overlays*. An overlay maps node or edge identifiers to arbitrary values represented by ROS messages. While each

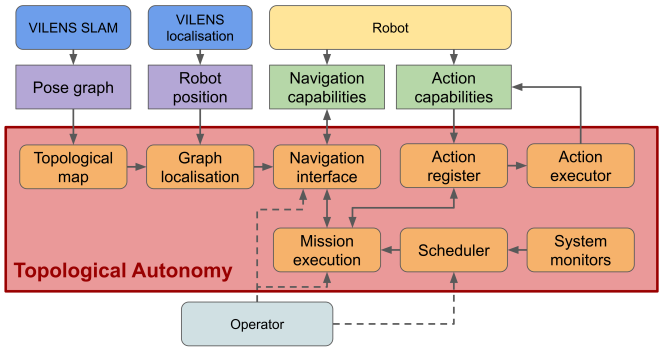

FIGURE 7. **Overview of topological autonomy on Spot. The topological map, graph localisation, navigation interface, and action register adapt the system to different platforms.**

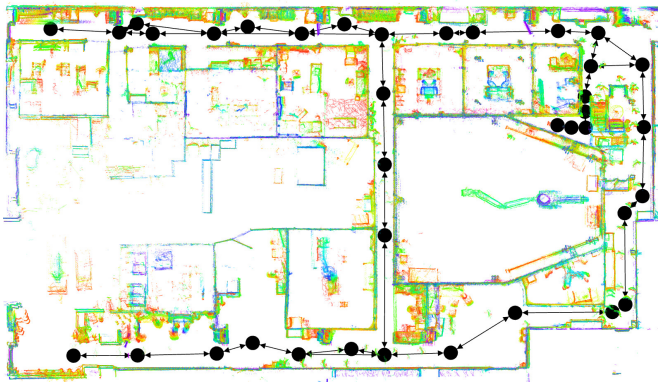

FIGURE 8. Top-down view of RACE B1 pointcloud with overlaid topological map. Points coloured by z coordinate height, red is lowest.

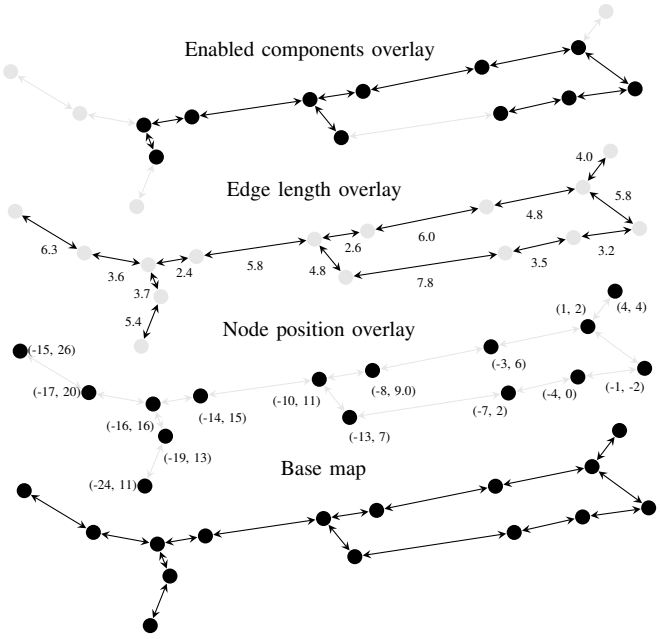

FIGURE 9. Example topological map and overlays. The node position overlay ties each node to a 2D coordinate. The edge length overlay reads the node position overlay to generate lengths for all edges. Enabled components is a binary overlay which can be used to specify which edges and nodes in the map are enabled. For presentation purposes, the nodes are depicted in a 2D plane, and node positions and edge lengths are approximated. Nodes or edges which do not have a value in an overlay are show in grey.

overlay depends on the topological map for information about map components, it exists as an independent part of the system and is published to its own ROS topic. Each node in the system can thus subscribe to only those overlays it needs for operation, and can publish its own overlays. This provides flexibility, as different parts of the system can add a variety of overlays as needed without affecting the operation of others, or needing to modify the base map. This design was chosen based on experiences in the STRANDS project [26], where topological map data types became over-fitted to particular application elements, and thus limited our ability to re-use the software across robots and projects.

Figure 9 shows an example of overlays defining a 2D structure for the topological map. The node position overlay assigns a 2D position to each node in the map. The edge length overlay then uses the node position overlay to define edge lengths for each edge in the map, which can be used for path planning. The enabled components overlay defines which parts of the topological map can be used for navigation. Any changes to the node position overlay can be propagated to other overlays which depend on it through an update procedure. For example, in the edge length overlay the procedure is to recompute edge lengths for any edges attached to a node whose position is changed. This update procedure also means that we can make modifications to the topological map online and have those changes propagate through the entire system automatically.

Topological maps are defined using a `yaml` structure, and can thus be constructed in a variety of ways. Our GUI tools allow us to construct maps entirely by hand, and we have also constructed maps online while exploring an environment [63]. In Section VII-B we describe how we use the SLAM pose graph as a basis for constructing the topological map. The topological map and overlays are a simple yet powerful representation, allowing us to annotate nodes and edges with domain-specific information for deployments. It is also an ideal structured abstraction of the environment for input to planning algorithms [33].

B. Topological Navigation

We use the topological map as the basis of our autonomous navigation framework, which is built to be adapted to a variety of robot platforms. Integrating a platform into the framework requires the implementation of two components. The first component is a *topological localiser*, which connects the robot's position in physical space with its position in the topological map. For most deployments, we define an overlay of 3D node poses indicating their location and orientation in a global reference frame, and an *influence zone* which defines their extent in space. The localiser must translate the robot's pose in the environment to a node in the topological map. We typically do this using the ROS transform tree. If the pose of the robot is within the influence zone of a node, it is considered to be at that node in the topological map.

The second component is a set of *edge traversal* action-servers, which use the local motion control system of the robot to move it between nodes which are connected in the topological map. Each edge in the map is associated with an action server, which has a standardised interface to receive edge information. Traversing an edge should move the robot from the edge's start node to its end node. This means that the topological map effectively acts as a soft constraint on where in the environment the robot can move. The exact form of this motion is defined by the implementation of the action server. This abstraction is a

key part of flexibility of our system, as the action server can be customised to interface with the local navigation system of a variety of robot platforms. Associating different types of traversal actions with edges allows us to define how the robot behaves when travelling between nodes, with no human input necessary during navigation. For example, custom actions may be required for moving through doors or up stairs. The edge traversal action server can also be used to combine the edge traversal with other inspection or monitoring actions. Each action server is required to report success or failure of the traversal, which is used within the topological navigation framework to retry or abandon traversal of the edge. We store the start and end time, duration, and success or failure of edge traversals in a database, which facilitates analysis of the robot's performance during deployments.

Robot tasks are generally driven by the need to be at a specific location, so navigation on a purely edge-by-edge basis would be cumbersome. We provide an additional node-based navigation layer on top of edge traversal to send the robot to a specific node, called the *traversal policy executor*. This component receives the robot's current and destination nodes, and computes the shortest path between them. The cost of each edge in the shortest path calculation is defined by an overlay. We can easily use a variety of cost measures such as distance, duration, or congestion, or even define an overlay which combines all of these into a single measure. The computed path is executed by calling the action server for each edge in turn. Paths through the map can involve multiple different types of edge traversals, all handled automatically. For efficiency, we only require the robot to orient itself to a node if it is the final node in a path. Once an intermediate node is reached, the policy executor immediately requests the next edge traversal. If an edge traversal fails, the underlying action server will report failure, which temporarily adds the edge to an overlay of disabled edges which cannot be used for navigation. The edges in this overlay are excluded when computing the shortest path, so whenever a failure occurs the system can automatically reroute and attempt an alternative path to the destination. As disabled edges are stored in an overlay, it is not necessary to modify the underlying topological map to reroute the robot.

C. Mission Planning and Scheduling

Mission execution and scheduling links navigation with task execution. We define a task as an action performed at a topological node. Each action available to the system is defined in an *action register*. Each action has a name, and is associated with a specific action server and a set of parameters for that action server. The action register acts as an abstraction layer between the autonomy system and the implementation of the actions. By providing parameters for actions, we ensure that at execution time no additional human input is necessary. When integrating with a platform, we can provide a set of default actions which interact with its basic capabilities, leaving users to integrate actions for custom payloads.

We define a mission as a list of tasks. Missions are usually specified by human operators, but can also be constructed programmatically. To execute a mission, each task is processed in order. First, the mission planner sends a request to the navigation system to move to the task node, and waits for the result. The navigation system will automatically retry if its initial path is blocked. If navigation is successful, the task's action is executed by calling the action register and then waiting for the task to complete. These steps repeat until the last task is processed. Tasks in the mission may fail either because the task location cannot be reached, or the action itself fails. When this happens, we can configure the system to either attempt to complete the remaining tasks, or abort. For missions where the order of tasks is not important, it can be challenging for humans to define the optimal ordering in terms of mission distance. The mission planner can automatically minimise the path cost by reordering tasks according to an approximate solution to a standard travelling salesperson problem (TSP) [64].

To provide control over when missions are executed, we use a scheduler, the key component for long-term autonomous operation. The scheduler builds on the mission execution system and can schedule a mission for execution once at a specific time, or repeat missions on a fixed schedule (e.g. every hour, every day at 09:00, every Tuesday at 16:00). The scheduler includes *monitors* which, when specified conditions are fulfilled, can request execution of a mission, or disable execution of missions. Monitors can be used to handle a variety of contingencies. For example, when the robot's battery is low, the battery monitor prevents any scheduled or user-requested mission from being executed, and immediately executes a mission which sends the robot to the nearest charging station.

D. User Interfaces

We provide interfaces implemented in the ROS RViz tool to interact with the system quickly and conveniently. Users can modify topological maps in the RViz window (Figure 10), which allows repositioning, addition, or deletion of nodes and edges. RViz can also display the point cloud of the environment, so users can take its structure into account while modifying the topological map. The map modification interface allows the map to be modified while the system is running and provides immediate feedback on the updated structure. During deployments, this interface allows us to assess potential problem areas in the map, and modify it accurately. The mission and scheduling panels (Figure 11) allow operators to define, save, load, execute, or interrupt missions and schedules, and to monitor their status.

These interfaces make interaction with the system easier, faster, and less error-prone for both expert and non-expert users of the system. After moving from command line interfaces to graphical interfaces, we observed a marked

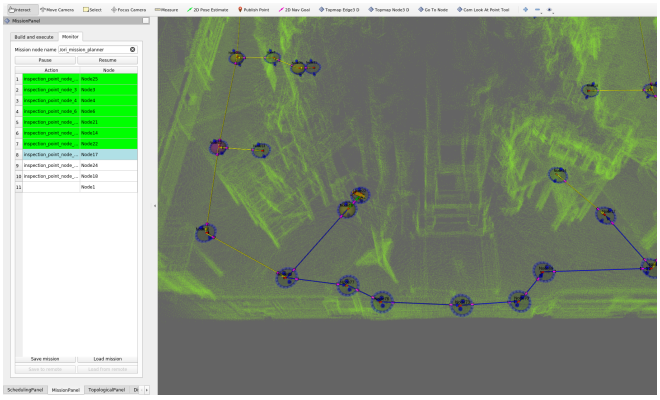

FIGURE 10. RViz interface showing the global map point cloud and topological map during AutoInspect deployment on Spot in the Torus Hall of the JET fusion reactor. Yellow arrows on edges indicate the global navigation policy, and blue arrows the expected path to the robot's next task location.

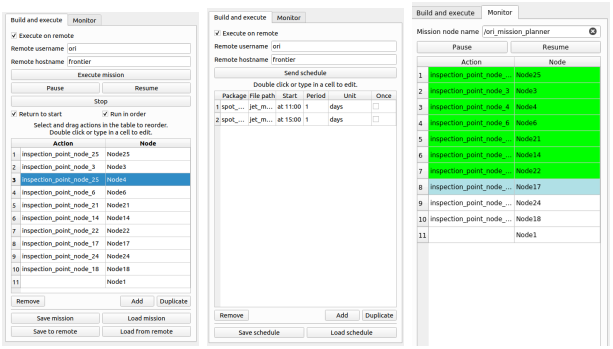

FIGURE 11. Topological autonomy user interfaces. *Left*: mission construction, with action and node specification, remote and local saving, loading, and execution. Ordering of missions can be changed by dragging and dropping tasks. *Centre*: schedule construction, with specification of predefined mission files and their repeat frequency. *Right*: mission monitoring, where row colour indicates status of a task. Green tasks have been successfully completed, and light blue indicates the robot is navigating to the task location.

decrease in the time from arrival on site to autonomous operation.

E. Mission Execution Example

We now briefly illustrate a typical execution with the topological autonomy system. The mission is executed in a Gazebo simulation using a Clearpath Jackal running the full AutoInspect system. Figure 12 shows the execution trace of the mission. Figure 13 shows the mission structure and the simulation environment, as well as images corresponding to parts of the execution trace. The *location* row shows the current position of the robot in the topological map. Where this row is blank, the robot is not currently at a node. The robot stays longer at goal nodes as it must orient to the node, and `move_base` can take some time to report success. A mission starts by requesting navigation to the first task location, indicated by the *go to node* row. This action starts with navigation policy generation, shown in the *generate policy* row. The traversal policy executor *executes the policy*, which sends commands to the *edge traversal*. Edge traversal receives a specific edge to traverse, then translates this into a pose to send to the *move_base* actionserver, which actually executes the motion. Once the robot reaches the task location, the *action* row shows when the action executor executes the action requested by the mission planner. In this example, we introduce an obstacle into the environment which blocks the edge between Node10 and Node11. When edge traversal executes the edge, it monitors the execution status of the `move_base` action. While initially there appears to be a valid path to go from Node10 to Node11, there is not enough space for the robot to pass through. `move_base` generates a much longer global plan to execute the traversal. Edge traversal compares the length of planned paths to the length of the edge, aborting the traversal if it is much longer than expected. This allows for flexibility for obstacle avoidance, but will abort edges which are entirely blocked. The *Edge blocked* row shows when the overlay tracking blocked edges is updated. The failure of the edge propagates up to the traversal policy executor, which retries the navigation task, generating a new policy which factors in the blocked edge by using the overlay. The new policy successfully executes, taking the robot back to the node at which the mission started.

V. Deployment Overview

In this section we present the process we follow to deploy an AutoInspect system in a new environment.

A. Localisation Pointcloud Generation

Once on site, the first step is to build the metric map of the environment (unless an existing point cloud is already available). To do this, we teleoperate the robot in the environment while running the SLAM system. The time needed for this process depends on the size of the environment, but generally only takes as long as is needed to move the robot around its areas of operation. To ensure a complete and accurate map is generated, operators must ensure that loop closures happen during the mapping process, since these necessary for correcting poses based on odometry drift over long distances. Once mapping is complete, the resulting point cloud does not require any post-processing and can be used directly to support autonomous navigation.

B. Topological Map Construction

The next step is to build the topological map. This can be done manually or automatically. In AutoInspect deployments we use the pose graph generated by the SLAM system as the basis of an initial topological map. Figure 14 shows an example of this process. Following the order in which the poses in the pose graph were generated, each one is converted to a topological map node with the same pose. Before creating a node, we check if the pose is close to an existing node, in which case it is merged into the existing

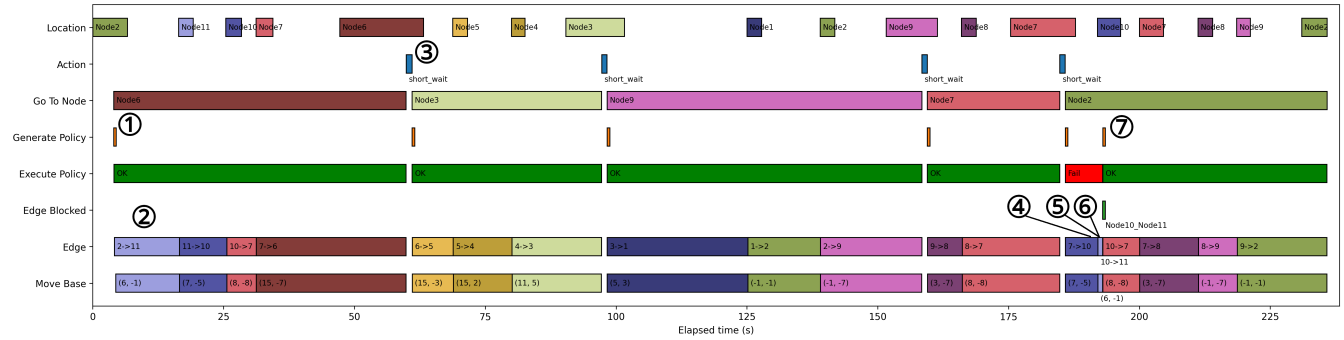

FIGURE 12. **Execution trace of a mission with four tasks. Bars correspond to the state of the system or the execution of a goal which has been sent to an actionserver. Bars are colour-coded according to the node associated with the action. Circled numbers refer to images in Figure 13.**

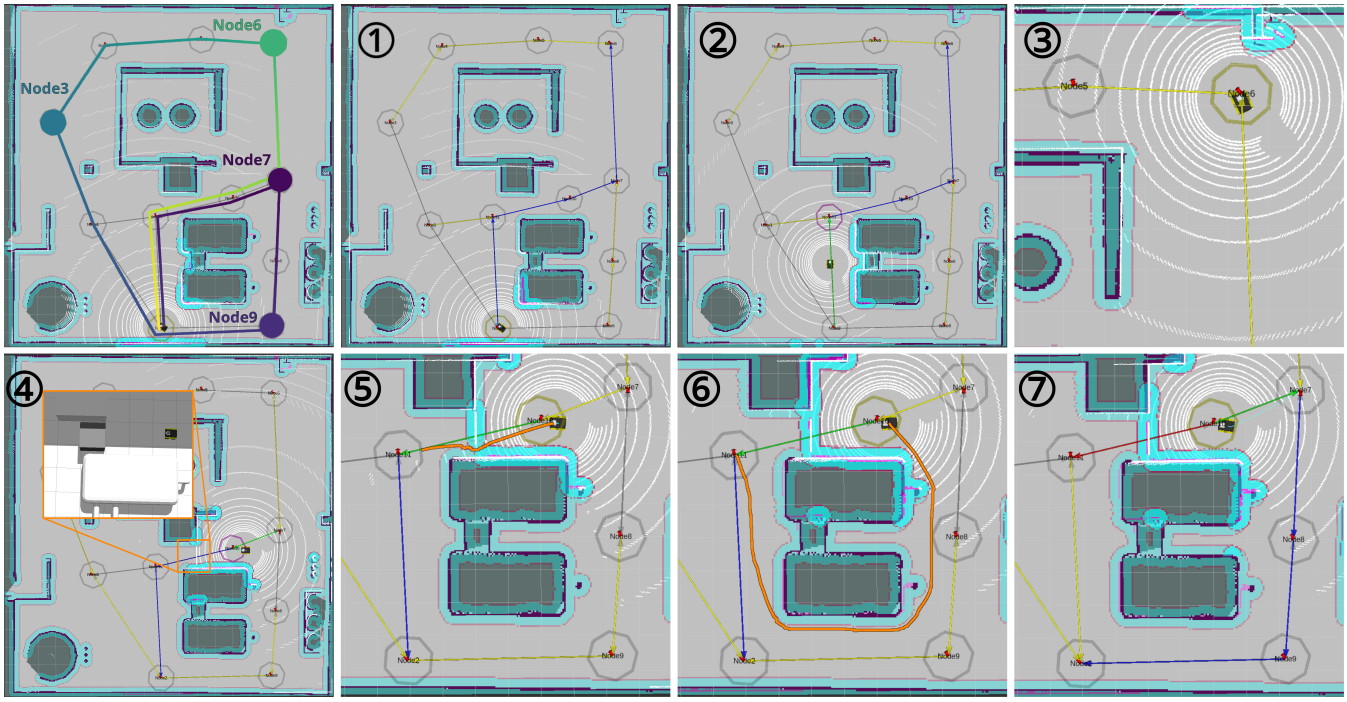

FIGURE 13. **Mission execution in simulation, circled numbers refer to parts of the execution trace in Figure 12. *Top left:* Expected mission route. Yellow to purple gradient indicates progression from start to end of route. Actions are executed at the 4 highlighted nodes. ① Generating the traversal policy (yellow arrows) and expected route (blue arrows) to Node6. ② Traversing to Node6, current edge in green. ③ Rotation to node orientation and execution of short wait action. ④ Obstacle is placed, blocking edge from Node10 to Node11. ⑤ `move_base` attempts to plan around obstacle (orange path). ⑥ `move_base` replans much longer route due to infeasible route. ⑦ Traversal actionserver aborts due to longer than expected route, causing edge from Node10 to Node11 to be blocked (red arrow). Policy is regenerated taking into account blockage, and executed.**

node. Edges in the pose graph either connect successive poses or define loop closures. We create bidirectional edges in the topological map using the edges in the pose graph, ensuring that edges which were connected to any merged poses are correctly connected to the corresponding node in the topological map.

This step provides the basic structure of the topological map. In the mapping stage every edge except those defined by loop closures was traversed, so we know which edges should be traversable. However, the resulting map has many redundant nodes due to pose graph nodes being placed at 1m intervals. We remove redundant nodes with a user-configurable sparsification process. We then hand-adjust the map to conform to operational requirements, adding nodes where actions are to be performed, and adjusting node positions to ensure safe and reliable navigation. Edges in the map are annotated according to the traversal method which must be used, for example door or stair traversal. For long-term deployments, the robot's charging stations are also added to the map as nodes connected by a suitable docking edge.

C. Autonomous Operation

After the topological map is created the system is ready to support autonomy. In our experience, the time from arriving on site to autonomous navigation is usually less than an hour. With the final topological map in hand, missions and schedules for long-term operation can be defined using the actions available to the robot. Once a schedule is set for the robot, it can operate autonomously, with operator action only needed to modify the schedule.

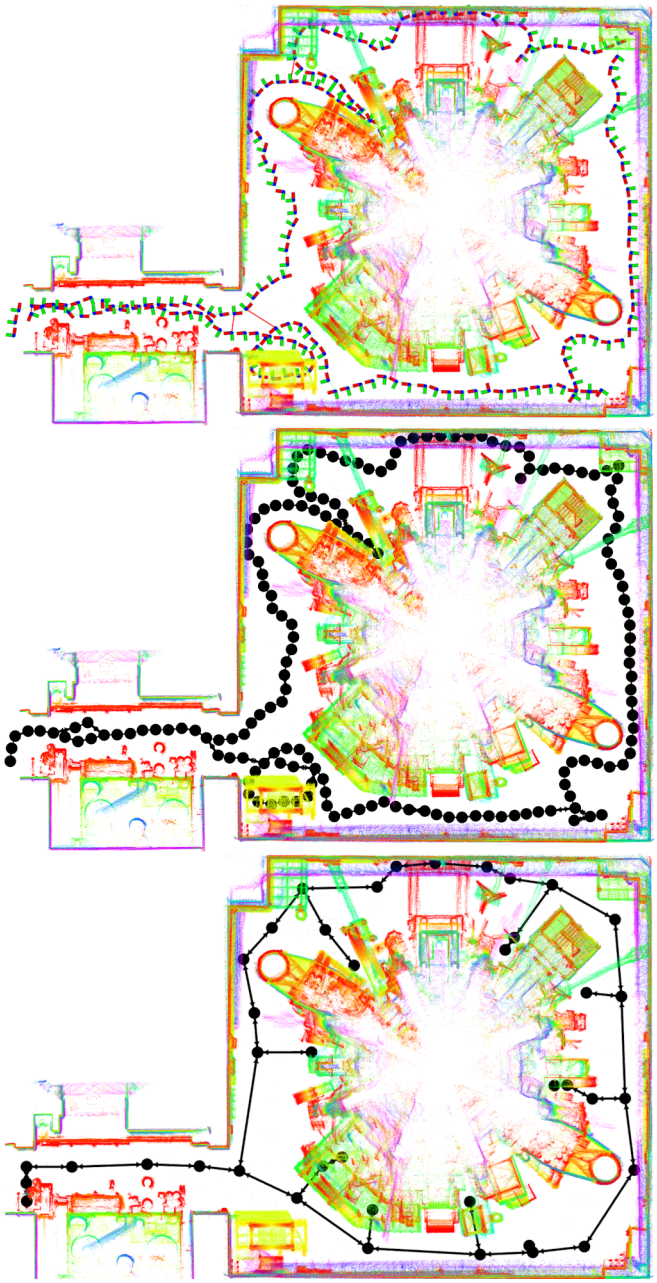

FIGURE 14. Building a topological map from the SLAM pose graph at JET. *Top*: SLAM pose graph, red edges indicate loop closures. *Middle*: Automatically constructed topological map. *Bottom*: Final topological map after manual pruning and adjustment for mission objectives.

VI. Spot AutoInspect Integration

We selected the Boston Dynamics Spot robot as our development platform for AutoInspect due to its reliable collision avoidance and stair climbing capabilities, which fit well with our intended application areas, as well as its autonomous docking capability, which facilitates long-term deployment. This section provides more detail on the integration of AutoInspect with the Spot robot.

A. Frontier Payload

The key hardware component of AutoInspect is our *Frontier* autonomy payload (Figure 3). It combines: a NUC mini-PC with i7-1165G72.8 GHz and 32 GB of RAM; a Hesai XT-32 LiDAR; three time-synchronised fish-eye Sevensense Alphasense cameras; and a Bosch BMI085 IMU. These are mounted in a compact 3D-printed case, with a combined weight of 1.5 kg and a power consumption of less than 60 W. Camera intrinsics and extrinsics are calibrated with Kalibr [65] while the camera-to-LiDAR extrinsics are calibrated according to [66]. As all sensors are mounted rigidly, we can move the Frontier between robots without recalibration.

B. Robot Hardware Integration

To integrate AutoInspect on to Spot, we attached the Frontier via a 3D-printed mounting plate connecting it to the robot's network and power supply through a payload port. To maintain network connectivity between the operator computer and the robot, we use an industrial-grade mesh network. One Rajant ES-1 network node is mounted on the robot, and another is connected to the operator computer. Other Rajant mesh nodes are set up in the environment to cover the operational area of the robot. Network connectivity across an entire site is only necessary if constant monitoring is required. AutoInspect is self-contained on the Frontier and does not require a network connection except to receive operator commands.

Depending on the particular application, we extend the system with different inspection payloads. For the deployments described in Section VII we mounted the Spot CAM+IR which has a pan-tilt-zoom (PTZ) unit with thermal and visual cameras. We also added a Kromek Sigma 50 Gamma ray detector, along with a TEMPerHUM temperature and humidity sensor, both connected to the Frontier device.

C. Topological Autonomy Integration

Topological navigation interfaces with Spot using the `spot_ros` driver[8], which provides access to point-to-point navigation and velocity controls. We use the driver to implement a variety of edge traversal actions, including stair climbing and autonomous docking. Topological localisation is achieved by using the robot's current pose in the TF tree provided by ICP odometry, and comparing it to locations

[8] github.com/heuristicus/spot_ros

of nodes in the map using a k-d tree. For task execution we implemented two action servers to integrate the sensing payloads. The Spot CAM+IR actions allows us to point the PTZ camera at coordinates in the global reference frame, set the zoom level, and capture images in either the optical or IR spectrum. The gamma ray detector action triggers the gamma ray detector to capture radiation spectrum data for 1 minute. The TEMPerHUM sensor did not require an action as it was set to collect data continuously.

VII. Spot AutoInspect Deployments

This section describe two long-term deployments of AutoInspect controlling Spot in the configuration described in the previous section. Both deployments took place at the Culham Centre for Fusion Energy in Oxfordshire, UK, in collaboration with UKAEA's centre for Remote Applications in Challenging Environments (RACE). The first deployment was at the B1 robotics test facility, and the second in the torus hall of the JET fusion reactor, shown in Figure I.

Throughout the deployments we tracked *interventions*, which we define as any modification of the system's state by a human. *Minor interventions* are normal user interactions with the system, such as modifying the topological map, running additional missions, or software failures which do not affect the system's functionality, such as logging. As such, we do not count minor interventions as interrupting the system. *Serious interventions* are when components or subsystems are modified or restarted during operation. *Fatal interventions* are when the entire software stack or the hardware is restarted.

1) RACE B1 Test Facility

The robot was deployed at the B1 test facility for total of 49 consecutive days from 18th July to 5th September 2023. The intent of this deployment was to build confidence with UKAEA staff and understand the requirements for future deployments, as well as to stress-test the system. Prior to this deployments of AutoInspect had focused on integration testing and short-term performance. In those deployments the system was not expected to run for more than several hours at a time. In contrast, the explicit aim of this deployment was continuous operation with minimal operator interruption.

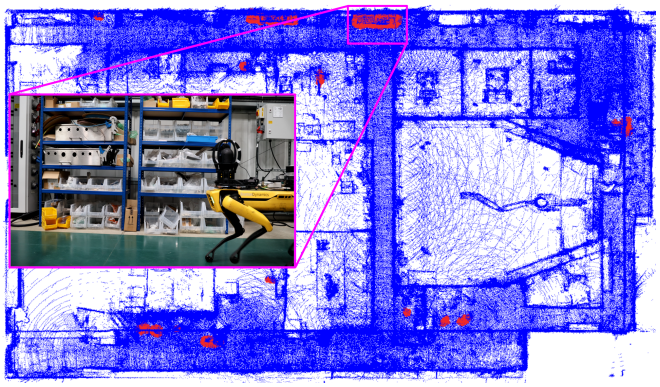

FIGURE 15. Change detection results from RACE deployment. Detections in red, static cloud in blue. Changes were organically induced by workers in the environment.

We logged 25 interventions over the course of the deployment. The longest period without any serious or fatal interventions was 14 days. Twelve interventions were minor. The nine serious interventions included manually re-localising the robot (3 times), and restarting navigation (3), the camera driver (2), or localization (1). Four fatal interventions were restarting the entire software stack (3) or rebooting the Frontier payload (1). Not counting minor interventions, the mean time between interventions (MTBI) was 78 hours. In practice, interventions tended to be clustered together, as configuration or hardware issues caused repeated failures. All of the interventions provided us with valuable information about how the system was used in practice, and how to improve its robustness. In this deployment, several interventions were caused by a bug in the localisation system which only manifested after long periods of time and was caused by thermal throttling of the CPU, which we may not have discovered without a long-term deployment.

The topological map, shown in Figure 8, consisted of 37 nodes, of which 9 were inspection points. We captured a visual spectrum image at all 9 locations, and IR images at 2 locations. At each inspection point, the we also recorded temperature and humidity readings. The SLAM system ran continuously in order to capture new pointcloud maps for input to the offline change detection system. The robot performed a scheduled mission at 11:00 and 15:00 each weekday, with an average duration of 11 minutes 30 seconds. Including unscheduled missions for visitors, it executed 84 missions, during which 673 of 730 inspection actions were successfully performed. There were 7 partially successful missions during the deployment, in which at least one action or traversal failed, accounting for 9 failed actions. Five failed missions account for the other 48 failures, where the entire mission failed to execute. These missions were associated with major interventions, failing due to errors with SPOT CAM+IR functionality, and a disk capacity issue caused by excessive logging. The robot walked approximately 13.6 km over the 13 hours in which it was actively performing missions. A total of 4565 edge traversals were attempted, 4191 of which were successful. The 374 failed attempts were associated with 249 requests for an edge traversal. 218 of these traversals were successfully completed after automatic retries generated by the autonomy system, with the remaining 31 unsuccessful. As with action failures during missions, the majority of edge failures occurred on days where we also recorded major interventions. Figure 15 gives an example of changes in the environment detected during the deployment, which were organically induced by workers, showcasing this important capability which provides operators with actionable insights.

2) Joint European Torus

The deployment in the torus hall of the JET fusion reactor ran for 35 consecutive days from 21st February to 27th March 2024. The robot was fitted with a gamma ray detector to perform radiation measurements near the reactor. We implemented a variety of improvements to the software based on experience from the previous deployment, including a fix for the localisation system's thermal throttling issue.

In this deployment there were 16 interventions. Excluding minor interventions, this is an MTBI of 140 hours, almost twice as long between interventions as the B1 deployment. The longest period without serious or fatal interventions was 15 days. 10 interventions were minor, 1 serious, and 5 fatal. The fatal interventions were all caused by a user-facing process manager used to monitor the system that would intermittently cause all ROS nodes to shut down when operators reconnected to the system. We noticed this bug during this deployment as due to construction and maintenance activities in the area operators would regularly connect to the running system and apply manual adjustments to the schedule. We suspect this was the cause of 3 fatal interventions at B1, which we were unable to identify at the time. None of the interventions were caused by the core systems, indicating an increase in reliability compared to the B1 deployment.

The topological map, shown in Figure 14, had 41 nodes including 7 inspection points. Two of the inspection points were for visual spectrum images, 3 for IR images, and at the remaining 2 we recorded the gamma radiation spectrum for 1 minute. Missions ran at 11:00 and 15:00, this time every day instead of just weekdays, with an average duration of 20 minutes. The robot executed 81 missions, during which 571 of 653 inspection actions succeeded. Three missions were partially successful, accounting for 10 action failures. The remainder occurred in 8 failed missions. Four of these failed missions were due to planned downtime over a weekend, during which a flag was set to disable the robot's motion, but all components continued running as normal. As in the B1 deployment, the remaining failed missions were associated with major interventions, this time caused by the process manager issue. The robot walked approximately 15km over the 19 hours 30 minutes in which it was performing missions. A total of 4464 edge traversals were attempted, of which 3833 were successful. 471 of the failures were on the edge leaving the docking station. Approximately 300 of these were caused by the weekend downtime, and the others by the process manager causing autonomy to fail. Of the remaining 160 failures, only a single edge did not succeed after automated retries. These numbers are further indication of improvement in reliability over the B1 deployment.

To capture images during the deployment, the PTZ camera was commanded to look at a point in the global reference frame. The accuracy of this command was entirely based on the localisation quality, with no visual servoing used. Figure 16 shows a sample of images taken from the same location across the deployment. We use descriptor matching to find a quantitative measure of the image variation, as we do not have access to ground truth. For each possible pairing of 60 images of the target, we compute ORB [67] keypoints and descriptors, then match the descriptors across the two images. Each match gives an estimate for the offset between the two images, though this is an approximation since the image is not strictly planar. Using the top 10 matches, we compute the mean offset in the x and y axes, which is then used as the estimated offset for that image. For all possible pairs of images of 1920 by 1000 pixels, the mean of the estimated offsets is 54 ± 71 pixels along x, and 51 ± 42 pixels along y, an average variation of less than 5%. The similarity of these images is an indicator of both localisation robustness and repeatability, which are important for long-term deployments. Figure 17 shows the cooling process of a reactor component which was captured during the deployment. These examples show how autonomous inspection can be used to provide useful information to plant operators independent of fixed instrumentation.

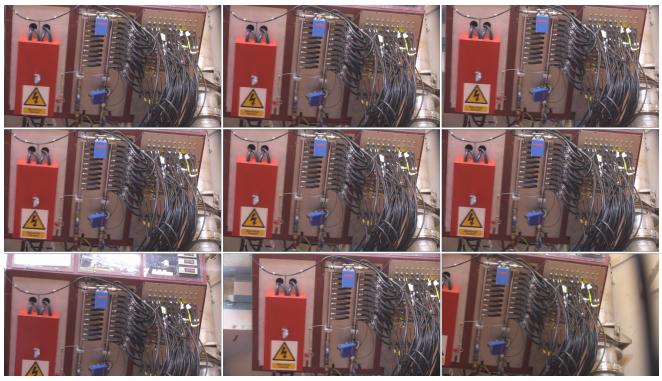

FIGURE 16. Inspection images of an electronics patch board taken with the Spot CAM+IR during the JET trial of object about 1.5m wide from a distance of approximately 5m. Top 6 images randomly sampled from 60 taken at the location. Bottom row are images with largest estimated offset of all images in the set.

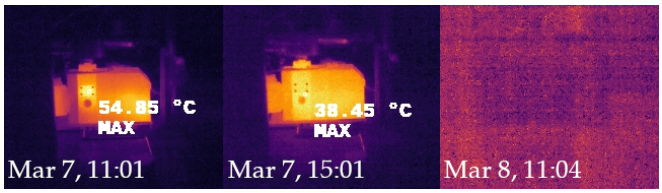

FIGURE 17. Images from Spot CAM+IR across three different missions at JET, showing cooling of a reactor component over the course of 24 hours.

VIII. Deployments on other Platforms

AutoInspect was developed to be robot platform-agnostic, and to provide a flexible substrate for application-driven autonomy. To demonstrate this, we now describe the use of AutoInspect for other hardware platforms and applications.

FIGURE 18. ANYmal Monitoring in Wytham Woods. ***Top:*** **ANYmal navigating the forest.** ***Bottom:*** **Topological map and pointcloud used by the robot to navigate in a** $50\,\mathrm{m}\times35\,\mathrm{m}$ **forest patch, with robot's front camera view overlaid.**

A. ANYmal Monitoring in Wytham Woods

We used AutoInspect as part of a system for continuous monitoring in forests with the ANYbotics ANYmal D platform (see Fig. 18). ANYmal is an IP67-certified and ROS-native quadruped robot, which has been extensively tested in unstructured [52], as well as natural [68], environments. In contrast to Spot, direct access to many of the robot's low level control systems is available as standard, and parts of the software system can be customised by users. The payload set-up on the ANYmal platform is similar to that of Spot, with the Frontier device mounted on the front of the robot together with a Rajant DX-2 mesh network node. As the accessible power connections on the robot are current-limited, Frontier was powered by an external battery. We protected the Frontier with a roll cage, as natural environments often have uneven terrain and a variety of other hazards such as low branches.

Integration of AutoInspect on ANYmal is similar to Spot. The Frontier provides state estimation through LiDAR-inertial odometry, LiDAR mapping, and ICP- or place recognition-based localisation. Both localisation methods provide the 3D pose of the robot in a global frame. In contrast to Spot, we did not use the leg odometry estimate from the robot, as we observed that LiDAR-inertial was sufficient to localize in a forest. Leg odometry is helpful in confined, narrow, or repetitive environments, which are not generally characteristics of natural spaces. To integrate topological navigation, we used a reactive local planner [69] that provides an interface similar to `move_base` [70], which we tuned and extensively tested in natural environments [68]. The local planner relies on a $6\,\mathrm{m}\times6\,\mathrm{m}$ local terrain map built by the default robot stack using its front and side-mounted depth cameras.

We carried out field trials in Wytham Woods, Oxford, UK, on the 26th and 27th June 2024, with the objective of testing ANYmal integration and evaluating potential uses cases in forestry applications. For our experiments, we selected a $50\,\mathrm{m}\times35\,\mathrm{m}$ forest patch, and manually teleoperated the robot to build the localisation map. The resulting topological map contained 234 topological nodes, which was then downsampled to 46 nodes for smoother navigation (see Fig. 18). The goal of the deployment was to assess the system's ability to traverse long edges in the topological map. We did this by testing autonomous navigation between pairs of nodes on the topological map which were an average of 25 m apart. In addition, we automatically added new edges to the topological map had not been traversed during the mapping phase. This was achieved by implementing a geometric obstacle check performed using VP-STO [71] on the pointcloud. If a path could be found by VP-STO in the pointcloud then the edge was added to the map, otherwise it was discarded. Despite this initial filtering step, some of these new edges could not be autonomously navigated due to terrain constraints. Such edges were marked as failed in the topological autonomy system after a specified timeout, and subsequent routes would then avoid using the failed edge.

This demonstrated that AutoInspect can be adapted for other robotic platforms, such as the ANYmal. This adaptation is particularly straightforward when the basic navigation interfaces are similar. Once the AutoInspect integration was completed, seamless execution of autonomous missions was possible.

B. Husky Long-term Biodiversity Monitoring

We have also applied the topological autonomy component of AutoInspect to an autonomous system for biodiversity monitoring, using the Husky UGV [72]. The primary difference between this deployment and those above is that we used a visual experience-based map for navigation and localisation [73], instead of a metric map. This highlights the flexibility of design of the topological autonomy model, which does not require a metric map, or any global reference frame, since autonomy is modelled as edge traversals in a graph. As with ANYmal, this system was deployed at Wytham Woods. It was deployed as part of a project to study plant biodiversity changes in experimental drought conditions [74]. The deployment area contains over 60 permanent experimental grassland plots, each of which has controlled

watering and irrigation, simulating the effects of climate change and human disturbance on grassland ecosystems. Data on these plots was previously collected entirely based on manual observations.

The Clearpath Husky A200 is an all-terrain UGV with a high payload capacity. We equipped the robot with a forward-facing Hokuyo planar LiDAR which we use to set speed limits near obstacles. A forward-facing Bumblebee2 stereo camera is used for mapping and localisation using visual teach-and-repeat. Autonomous docking is implemented using laser-based line-segment matching and visual servoing to guide the robot to a Wibotic wireless charger. The robot is equipped with side-facing monocular, thermal, and multispectral cameras for data collection, which feed data into a biodiversity estimator [75]. Experience-based mapping and localisation with vision [73] has been proven to be robust in extremely challenging domains, e.g. Martian-analogue [76]. Our system uses FAST-BRIEF feature point detection and description [77, 78], visual odometry [79], and place recognition with FAB-MAP [80]. To build the experience map, we teleoperated the robot along various routes shown in Figure 19①. The resulting experiences are dense strings of experience nodes, each of which is associated with images captured when it was first traversed. Once an experience has been taught, the robot is able to repeat it autonomously.

Topological autonomy was integrated on top of the experience-based localisation and navigation system. We constructed the topological graph by first selecting a subset of the experience nodes to map to topological nodes. We did this for the nodes at the data-capture points (since the robot should stop here to record data) and at the intersections of experiences in order to allow the topological autonomy system to recombine experiences into new routes not traversed during the teaching phase. Topological localisation requires determining the closest topological node to the robot. Since the visual teach-and-repeat system does not have a global metric frame, we achieve this via a breadth-first search up to a fixed radius (typically 50m) on the experience graph, starting at the experience node that the robot is localised to best. Since topological nodes are anchored to experience nodes, the first experience node/topological node pair which is found is considered the closest node. Since we use visual teach-and-repeat, the traversability of edges is not in question. If required, traversability information could be integrated into the system as an overlay which makes use of prior work in safe exploration [63] or traversability estimation [81].

The robot was deployed from 22nd June 2024 to 1st August 2024. We executed many autonomous missions across the period, with the longest being 1 hour and 45 minutes, which is close to the limit of the platform's battery life. Figure 19 (*Bottom*) shows the timing diagram for the longest mission. Over the course of the deployment the robot was autonomous for 33 h, and drove more than 14 km. More detail is available in [72]. This deployment shows that the design of

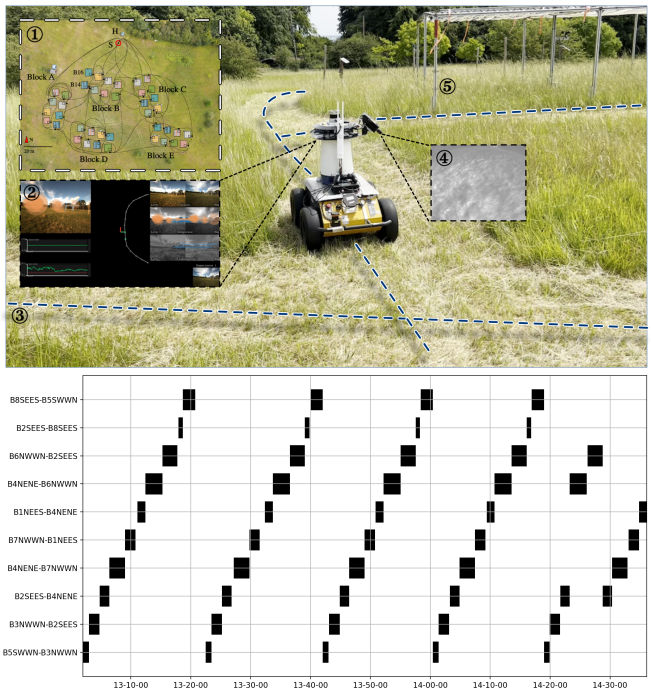

FIGURE 19. ***Top*: Application of the topological autonomy component of *AutoInspect* to autonomous biodiversity monitoring. In this a UGV was deployed over 6 weeks to automatically collect thermal images (④) at enclosures (⑤) at which endemic grasses' response to climate change is studied. A visual teach-and-repeat navigation system (②) which has the UGV follow taught paths (③) was integrated with *AutoInspect* for this, with the taught paths forming a network (①) over the entire site, including autonomously docking and charging (①H). *Bottom*: Timing diagrams of an example autonomous mission lasting 1.5 hours. The vertical axis shows pairs of node IDs for enclosures where thermal images are gathered. One entry in this plot corresponds to the duration of a journey from one collection site to another (e.g. from `B5SWWN` to `B3NWWN`).**

the topological autonomy component of AutoInspect allows it to be adopted to localisation methods other than those based on metric maps. As with the ANYmal integration in the previous section, once the navigation integration was achieved, the autonomy capabilities (e.g. task execution, mission planning, and scheduling) from AutoInspect became available without further effort.

C. Jackal Simulation and Transition to ROS2

We are in the process of integrating AutoInspect onto the Clearpath Jackal UGV, seen in Figure 21. As with Spot and ANYmal, Frontier localises the robot through place recognition or ICP localisation, using wheel odometry to extrapolate the pose between two iterations of ICP. Topological navigation uses the ROS navigation stack [70], with `move_base` used to traverse between topological nodes. Dynamic obstacles are inserted into the 2D local cost map by taking a slice of the current 3D LiDAR scan and projecting the points onto the ground plane. We create a global costmap by slicing the SLAM map at a fixed height.

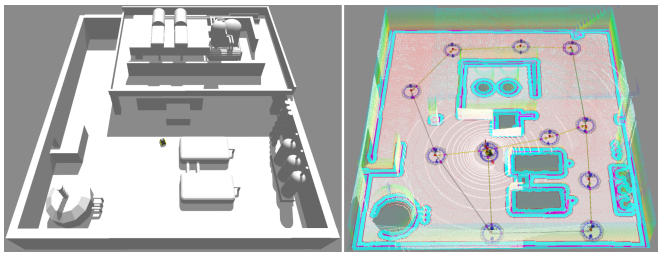

FIGURE 20. *Left*: ROS1 Gazebo simulation of industrial environment and Jackal. *Right*: RViz display of simulated components and AutoInspect in autonomy mode. White points are simulated 3D LiDAR. Coloured points are the 3D map generated by VILENS SLAM in the initial mapping step, here used for ICP localisation. 2D overlay is the global costmap used by `move_base` for navigation. Topological map shown as connected nodes and edges. Mission execution is shown in Figures 12 and 13.

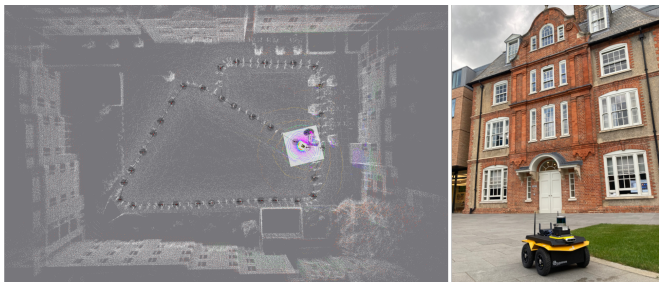

FIGURE 21. Initial deployment of the ROS2 prototype on the Clearpath Jackal at H.B. Allen Centre. *Left*: Localising in the SLAM map, with topological map generated from pruned SLAM poses. *Right*: Jackal hardware configuration with Frontier and Realsense D435i.

Before working on the physical system, we integrated the full AutoInspect system onto a simulated Jackal where SLAM, localisation, and topological autonomy could run. Figure 20 shows a small test environment in Gazebo with a simulated Jackal. The simulated 3D LiDAR is used during the mapping step to generate a 3D pointcloud that is used to localise during autonomous operation, as in the physical system. The topological map is constructed using the SLAM pose graph as a base, then modified by hand as needed.

Having a simulation means we can also perform some comparisons to ground truth data for localisation, which was not possible in the environments in which AutoInspect was physically deployed. We executed the mission described in Section IV-E 120 times. The missions were executed in real-time in the simulation, over the course of approximately 7 hours. All 120 missions were successfully completed. The average duration of each mission was 3 minutes 30 seconds, with a standard deviation of 9 seconds. In each mission the robot visited 4 locations, then returned to the start location. A total of 2124 edge traversals were attempted, all of which succeeded. During execution we recorded our localisation of the robot using ICP, and its ground truth position as defined by Gazebo, both at 2Hz. With approximately 52,000 pairs of samples, the mean position difference was $0.020m \pm 0.014m$, and the mean angle difference was $0.325° \pm 0.680°$. The Jackal simulation has some idiosyncrasies which contribute to lower angular accuracy, in particular very rapid acceleration when turning on the spot. The physics simulation also causes the robot's model to translate during rotations, which does not occur on the physical robot.

With the end of support for ROS1 approaching, we took the opportunity provided by the simulation to test a ROS2 implementation of AutoInspect. We updated our simulation to use ROS2 implementations of AutoInspect components, and verified system integration by building a map of a simulated environment, then executing autonomous navigation tasks. We performed the first physical deployment of the AutoInspect system in ROS2 at Keble College's H.B. Allen Centre, using the maps in Figure 21 for localisation and navigation. In our initial test we executed 19 missions, of which 5 failed due to issues with navigation controllers. We were able to quickly resolve these issues through further testing in simulation. We then ran 20 missions with the new configuration, all of which were successful. Over both sets of missions, the robot performed 472 edge traversals with only 5 failures caused by the initial controller issues, travelling 1.5km in 1 hour and 40 minutes.

Simulation of the system is helpful for integration, testing, and debugging, but there are some important limitations. In earlier sections we described how physical instantiation of the system on Spot experienced thermal throttling issues caused by interaction between hardware and software, which we would be unlikely to observe in simulation. Motion of the robot is another key concern. Even a relatively simple robot like Jackal has significant differences in how it moves in simulation and in the physical world. Accurately simulating quadrupeds like Spot and ANYmal would require significant computational resources. Complex built-in components like obstacle avoidance and stair climbing would have to be reproduced, making it impractical. Simulations also cannot reproduce the unknown dynamics of the real world. As we experienced during the JET deployment, even in the most controlled environments the robot's operational area is changed by the people who work in those spaces. Human presence in these spaces is another barrier to accurately simulating them, as it is very difficult to predict the number and type of interactions a system is likely to experience. The constantly changing demands of the real world mean that simulation can never truly be a substitute for physical deployments.

IX. Discussion

A. AutoInspect as a Substrate for Autonomy

AutoInspect has become our standard substrate for autonomy. Whilst it is often tempting to develop new software platforms, and unique hardware integrations, when faced with a new application or robot platform, we argue that adopting a standard approach to an autonomy stack has a number of benefits. The repeated use of a single soft-

ware/hardware tool like AutoInspect across a range of settings increases its robustness, since each new adoption drives the discovery and fixing of bugs, and the development of new capabilities. The benefits of standardisation and reuse are widely known across engineering fields, including robotics, but bear repeating when working in a space such as robot autonomy, where the pace of change, and often different appearances of applications/platforms, often drives people to reinvent the wheel. A powerful additional benefit we have found through adopting AutoInspect is that the elements of topological autonomy, have become a clear *design language* that we can use to communicate with stakeholders, from integration engineers to end-users. For example, by framing the problem of making a platform autonomous as the problems of supporting topological edge traversal and topological localisation, what could be an unconstrained challenge, becomes a clear set of integration tasks. Similarly, when planning how to deliver a new application, rather than consider all the possible ways this could be done with the chosen robot platform, the question can be reframed as how to map the autonomy requirements on to the AutoInspect framework (in terms of payload, prior map, task nodes etc.).

The value of AutoInspect as both a tool for both facilitating communication about, and enabling the practical deployment of, autonomy, as allowed us to use it to support both novel autonomy research, and quick translation of that research to applied projects. For example, AutoInspect has been used to underpin research in exploration [63], time-bounded mission planning [82], and change detection [61], as well as to deploy autonomous inspection systems into high-value and high-risk industrial settings, including active nuclear sites, large-scale perimeter monitoring settings, and human-in-the-loop mapping of decommissioning environments that are inaccessible to humans.

The design of AutoInspect was inspired by years of experience in developing and deploying autonomous systems, from lab demonstrators [83], to large scale competitions [52, 84] and in-the-wild long-term deployments [26]. The topological map is widely used in mobile autonomy [21, 33, 34]. However, it is unlikely to be the best spatial representation, or abstraction of autonomy, for all platforms and applications. For example, if the precise behaviour of the platform in continuous space and time is crucial for your use-case, then topological autonomy's inherently discrete abstract model of the robot's environment and capabilities may obscure too much of the continuous world to provide added value.

B. Engineering for Long-Term Autonomy

More practically, AutoInspect has allowed us to build and deploy mobile robot systems capable of long-term autonomy in real industrial environments. In doing this, we have learned lessons about both systems engineering and interacting with users and challenge owners. As above, standardisation has been key to fast deployments. This includes adopting standard sensors and compute (the Frontier), networking (mostly Rajant systems), and even platforms (defaulting to Spot where possible).

We have also found that developing a standard user interface (UI) that allows the operation of the autonomy system with only a small amount of training to be hugely valuable. In our experience a good UI makes system operation faster and less error-prone than using the command line, and allows us to quickly place the system in the hands of independent users, placing our team into a remote (rather than hands-on) support role.

For successful long-term deployments, a pre-deployment testing period in a new environment is necessary to ensure that there are no unexpected things which cause unnecessary interruptions to the deployment. This is true even when deploying a well-tested autonomy system like AutoInspect, since particular features of an application environment, from user behaviours to the presence of other wireless signals, may impact on the system as complex as an autonomous robot in new ways that have not previously been encountered.

To support monitoring and debugging, during both pre-deployment and live operation, we recommend logging system configuration and state extensively, ideally across the levels of the autonomy stack. In AutoInspect, in addition to logging low-level robot state, we record logs from the Frontier's localisation system (e.g. localisation uncertainty), and events across the topological autonomy system, including edge traversal and task execution statistics, mission plan evolution, and schedule status. This allows us to quickly assess the state of an AutoInspect system remotely (e.g. via ROS topic inspection) as well as replay or reconstruct the events leading up to an intervention when debugging.

One of the advantages of combining pointcloud localisation and a topological map for navigation within AutoInspect is that these approaches make it easy for a user to extend or modify the robot's model of its environment. The full extent of the robot's operational area is defined by the localisation pointcloud, so to extend AutoInspect to a new area, only an updated pointcloud is required for the previously unmapped region. Once nodes and edges are added to include the extended area, those areas are immediately accessible to autonomy without the robot needing to visit them. The modification of navigation routes within the operational area can be performed simply by changing the topological map, which can be done while the system is running.

X. Conclusion

We have presented an overview of AutoInspect, our mission-level mapping and autonomy system. We described two long-term deployments of the system with Spot at the Culham Centre for Fusion Energy, during which the robot operated autonomously for 2 weeks without interruption, demonstrating its robustness. The deployment at JET was the first ever deployment of a fully autonomous mobile robot in a fusion facility. We showcased the flexibility of our system through

descriptions of a further three robot platforms on which it is integrated.

The primary aim of future work is continued improvement in robustness of the system by stress-testing with increased up-time. In the near term, we plan to extend the system to multi-robot applications, and continue to explore applications for autonomous inspection robots at other large-scale industrial sites. Another key goal is to use planning and scheduling algorithms to generate policies for completion of missions which take into account uncertainty in the environment, such as the probability of successfully traversing edges and completing actions, as well as the duration of actions and edge traversals, which we can learn from data gathered during deployment. This will make it possible to provide feedback to operators about the probability of mission success. We will use the data from initial deployments to design autonomous inspection routines and sensor suites to gather actionable scientific and operational data, and to demonstrate that autonomous robots can provide tangible benefits to operators.

References

[1] David Wisth, Marco Camurri, and Maurice Fallon. "VILENS: Visual, Inertial, Lidar, and Leg Odometry for All-Terrain Legged Robots". In: *IEEE Trans. Robotics* 39.1 (2023), pp. 309–326.

[2] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. Vol. 3. Jan. 2009.

[3] Kristopher Toussaint, Nicolas Pouliot, and Serge Montambault. "Transmission Line Maintenance Robots Capable of Crossing Obstacles: State-of-the-art Review and Challenges Ahead". In: *Journal of Field Robotics* 26.5 (May 2009), pp. 477–499.

[4] Josep M. Mirats Tur and William Garthwaite. "Robotic Devices for Water Main In-pipe Inspection: A Survey". In: *Journal of Field Robotics* 27.4 (July 2010), pp. 491–508.

[5] David Lattanzi and Gregory Miller. "Review of Robotic Infrastructure Inspection Systems". In: *J. of Infrastructure Systems* 23.3 (Sept. 2017).

[6] Srijeet Halder and Kereshmeh Afsari. "Robots in Inspection and Monitoring of Buildings and Infrastructure: A Systematic Review". In: *Applied Sciences* 13.4 (Feb. 2023), p. 2304.

[7] C. Dario Bellicoso et al. "Advances in real-world applications for legged robots". In: *J. Field Robot.* 35.8 (2018), pp. 1311–1326.

[8] C. Gehring et al. "ANYmal in the Field: Solving Industrial Inspection of an Offshore HVDC Platform with a Quadrupedal Robot". In: *Field and Service Robotics*. 2021, pp. 247–260.

[9] Konstantinos Loupos et al. "Autonomous Robotic System for Tunnel Structural Inspection and Assessment". In: *Intl. J. of Intelligent Robotics and Applications* 2.1 (Mar. 2018), pp. 43–66.

[10] Filipe Rocha et al. "ROSI: A Robotic System for Harsh Outdoor Industrial Inspection - System Design and Applications". In: *J. of Intelligent & Robotic Systems* 103.2 (Oct. 2021), p. 30.

[11] M. Bloesch et al. "The Two-State Implicit Filter Recursive Estimation for Mobile Robots". In: *IEEE Robotics and Automation Letters* 3.1 (2018), pp. 573–580.

[12] Marco Camurri et al. "Pronto: A multi-sensor state estimator for legged robots in real-world scenarios". In: *Frontiers in Robotics and AI* 7 (2020), pp. 1–18.

[13] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time". In: *Proceedings of Robotics: Science and Systems*. 2014.

[14] Wei Xu et al. "Fast-lio2: Fast direct lidar-inertial odometry". In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2053–2073.

[15] Milad Ramezani et al. *Wildcat: Online Continuous-Time 3D Lidar-Inertial SLAM*. 2022. arXiv: 2205.12595 [cs.RO].

[16] Michael Kaess et al. "ISAM2: Incremental smoothing and mapping using the Bayes tree". In: *The International Journal of Robotics Research* 31.2 (2012), pp. 216–235.

[17] Kamak Ebadi et al. "Present and Future of SLAM in Extreme Underground Environments". In: *IEEE Transactions on Robotics (T-RO)* (Aug. 2022).

[18] R. Arandjelović et al. "NetVLAD: CNN architecture for weakly supervised place recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[19] Giseop Kim, Sunwook Choi, and Ayoung Kim. "Scan Context++: Structural Place Recognition Robust to Rotation and Lateral Variations in Urban Environments". In: *IEEE Transactions on Robotics* (2021). Accepted. To appear.

[20] Frank Dellaert et al. "Monte Carlo Localization for Mobile Robots". In: *Proceedings of (ICRA) International Conference on Robotics and Automation*. Vol. 2. May 1999, pp. 1322–1328.

[21] Benjamin Kuipers. "Modeling Spatial Knowledge". In: *Cognitive Science* 2.2 (Apr. 1978), pp. 129–153.

[22] R. Brooks. "Visual Map Making for a Mobile Robot". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. Vol. 2. 1985, pp. 824–829.

[23] Reid Simmons and Sven Koenig. "Probabilistic Robot Navigation in Partially Observable Environments". In: *IJCAI*. 1995.

[24] Sebastian Thrun. "Learning Metric-Topological Maps for Indoor Mobile Robot Navigation". In: *Artificial Intelligence* 99.1 (Feb. 1998), pp. 21–71.

[25] Kurt Konolige, Eitan Marder-Eppstein, and Bhaskara Marthi. "Navigation in Hybrid Metric-Topological

Maps". In: *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, May 2011, pp. 3041–3047.

[26] Nick Hawes et al. "The STRANDS Project: Long-Term Autonomy in Everyday Environments". In: *IEEE Robotics & Automation Magazine* 24.3 (Sept. 2017), pp. 146–156.

[27] Friedrich Fraundorfer, Christopher Engels, and David Nister. "Topological Mapping, Localization and Navigation Using Image Collections". In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. San Diego, CA, USA: IEEE, Oct. 2007, pp. 3872–3877.

[28] Rohan Paul and Paul Newman. "FAB-MAP 3D: Topological Mapping with Spatial and Visual Appearance". In: *2010 IEEE International Conference on Robotics and Automation*. Anchorage, AK: IEEE, May 2010, pp. 2649–2656.

[29] Ioannis Kostavelis and Antonios Gasteratos. "Semantic Mapping for Mobile Robotics Tasks: A Survey". In: *Robotics and Autonomous Systems* 66 (Apr. 2015), pp. 86–103.

[30] Emilio Garcia-Fidalgo and Alberto Ortiz. "Vision-Based Topological Mapping and Localization Methods: A Survey". In: *Robotics and Autonomous Systems* 64 (Feb. 2015), pp. 1–20.

[31] Lars Kunze et al. "Searching Objects in Large-Scale Indoor Environments: A Decision-Theoretic Approach". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. St Paul, MN, USA, May 2012, pp. 4385–4390.

[32] Lenka Mudrova, Bruno Lacerda, and Nick Hawes. "An Integrated Control Framework for Long-Term Autonomy in Mobile Service Robots". In: *European Conference on Mobile Robotics (ECMR)*. Sept. 2015, pp. 1–6.

[33] Bruno Lacerda et al. "Probabilistic Planning with Formal Performance Guarantees for Mobile Service Robots". In: *Intl. J. of Robot. Res.* 38.9 (Aug. 2019), pp. 1098–1123.

[34] Gautham Das et al. "A Unified Topological Representation for Robotic Fleets in Agricultural Applications". In: (Sept. 2023).

[35] Zuoyue Li, Jan Dirk Wegner, and Aurelien Lucchi. "Topological Map Extraction From Overhead Images". In: *Intl. Conf. on Computer Vision (ICCV)*. IEEE, Oct. 2019, pp. 1715–1724.

[36] Sebastian Thrun and Arno Bücken. "Integrating Grid-Based and Topological Maps for Mobile Robot Navigation". In: *National Conf. on Artificial Intelligence*. 1996.

[37] Z. Zivkovic, B. Bakker, and B. Krose. "Hierarchical Map Building and Planning Based on Graph Partitioning". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2006, pp. 803–809.

[38] Fabian Blochliger et al. "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. May 2018, pp. 3818–3825.

[39] Lars Kunze et al. "Artificial Intelligence for Long-Term Robot Autonomy: A Survey". In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 4023–4030.

[40] Wolfram Burgard et al. "Experiences with an Interactive Museum Tour-Guide Robot". In: *Artificial Intelligence* (1999).

[41] I.R. Nourbakhsh, C. Kunz, and T. Willeke. "The Mobot Museum Robot Installations: A Five Year Experiment". In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. Vol. 3. Las Vegas, NV, USA: IEEE, 2003, pp. 3636–3641.

[42] Shengye Wang and Henrik I. Christensen. "TritonBot: First Lessons Learned from Deployment of a Long-Term Autonomy Tour Guide Robot". In: *IEEE Intl. Sym. on Robot and Human Interactive Communication (RO-MAN)*. Nanjing: IEEE, Aug. 2018, pp. 158–165.

[43] Francesco Del Duchetto, Paul Baxter, and Marc Hanheide. "Lindsey the Tour Guide Robot - Usage Patterns in a Museum Long-Term Deployment". In: *IEEE Intl. Sym. on Robot and Human Interactive Communication (RO-MAN)*. New Delhi, India: IEEE, Oct. 2019, pp. 1–8.

[44] Marvin Stuede et al. "Sobi: An Interactive Social Service Robot for Long-Term Autonomy in Open Environments". In: *2021 European Conference on Mobile Robots (ECMR)*. Bonn, Germany: IEEE, Aug. 2021, pp. 1–8.

[45] Joydeep Biswas and Manuela Veloso. "The 1,000-Km Challenge: Insights and Quantitative and Qualitative Results". In: *IEEE Intelligent Systems* 31.3 (May 2016), pp. 86–96.

[46] Manuela Veloso et al. "CoBots: Robust Symbiotic Autonomous Mobile Service Robots". In: *International Joint Conference on Artificial Intelligence*. 2015.

[47] Wim Meeussen et al. "Long Term Autonomy in Office Environments". In: *ALONE Workshop, In Proceedings of Robotics: Science and Systems (RSS'11)*. 2011.

[48] Shengye Wang et al. "Robotic Reliability Engineering: Experience from Long-Term TritonBot Development". In: *Field and Service Robotics*. Ed. by Genya Ishigami and Kazuya Yoshida. Vol. 16. Singapore: Springer Singapore, 2021, pp. 45–58.

[49] Matteo Iovino et al. "A Survey of Behavior Trees in Robotics and AI". In: *Robotics and Autonomous Systems* 154 (Aug. 2022), p. 104096.

[50] Michael Cashmore et al. "ROSPlan: Planning in the Robot Operating System". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 25 (Apr. 2015), pp. 333–341.

[51] Ali Agha et al. "NeBula: TEAM CoSTAR's Robotic Autonomy Solution That Won Phase II of DARPA Subterranean Challenge". In: *Field Robotics* 2 (July 2022), pp. 1432–1506.

[52] Marco Tranzatto et al. "CERBERUS: Autonomous Legged and Aerial Robotic Exploration in the Tunnel and Urban Circuits of the DARPA Subterranean Challenge". In: *Field Robotics* 2 (Mar. 2022), pp. 274–324.

[53] Sebastian Scherer et al. "Resilient and Modular Subterranean Exploration with a Team of Roving and Flying Robots". In: *Field Robotics* 2 (May 2022), pp. 678–734.

[54] Harel Biggie et al. "Flexible Supervised Autonomy for Exploration in Subterranean Environments". In: *Field Robotics* 3 (Jan. 2023), pp. 125–189.

[55] Haedam Oh et al. "Evaluation and Deployment of LiDAR-based Place Recognition in Dense Forests". In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. Abu Dhabi, UAE, 2024.

[56] Rowan Border et al. "Osprey: Multi-Session Autonomous Aerial Mapping with LiDAR-based SLAM and Next Best View Planning". In: *IEEE Trans. Field Robotics* 1 (July 2024), pp. 113–130.

[57] François Pomerleau et al. "Comparing ICP Variants on Real-World Data Sets". In: *Autonomous Robots* 34.3 (2013), pp. 133–148.

[58] F Dellaert and M Kaess. "Factor Graphs for Robot Perception". In: *Foundations and Trends in Robotics* 6 (2017), pp. 1–139.

[59] Giseop Kim and Ayoung Kim. "Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map". In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. 2018, pp. 4802–4809.

[60] Kavisha Vidanapathirana et al. "LoGG3D-Net: Locally guided global descriptor learning for 3D place recognition". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2022.

[61] Joseph Rowell, Lintong Zhang, and Maurice Fallon. "LiSTA: Geometric Object-Based Change Detection in Cluttered Environments". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2024.

[62] Lintong Zhang et al. "InstaLoc: One-shot Global Lidar Localisation in Indoor Environments through Instance Learning". In: *Robotics: Science and Systems (RSS)*. 2023.

[63] Alex Stephens et al. "Planning under Uncertainty for Safe Robot Exploration Using Gaussian Process Prediction". In: *Autonomous Robots* 48.7 (Oct. 2024), p. 18.

[64] César Rego et al. "Traveling Salesman Problem Heuristics: Leading Methods, Implementations and Latest Advances". In: *European Journal of Operational Research* 211.3 (June 2011), pp. 427–441.

[65] Joern Rehder et al. "Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2016, pp. 4304–4311.

[66] Lanke Frank Tarimo Fu, Nived Chebrolu, and Maurice Fallon. "Extrinsic Calibration of Camera to LIDAR Using a Differentiable Checkerboard Model". In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. 2023, pp. 1825–1831.

[67] Ethan Rublee et al. "ORB: An Efficient Alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2564–2571.

[68] Matías Mattamala et al. "Deploying Autonomous Legged Robots in Forests — System, Lessons, and Challenges Ahead". In: *IEEE Trans. Field Robotics* (Oct. 2024). Submitted.

[69] Matías Mattamala, Nived Chebrolu, and Maurice F. Fallon. "An Efficient Locally Reactive Controller for Safe Navigation in Visual Teach and Repeat Missions". In: *IEEE Robot. Autom. Lett. (RA-L)* 7.2 (2022), pp. 2353–2360.

[70] Eitan Marder-Eppstein et al. "The Office Marathon: Robust Navigation in an Indoor Office Environment". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2010, pp. 300–307.

[71] Julius Jankowski et al. "VP-STO: Via-Point-based Stochastic Trajectory Optimization for Reactive Robot Behavior". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10125–10131.

[72] Matthew Gadd et al. *Watching Grass Grow: Long-term Visual Navigation and Mission Planning for Autonomous Biodiversity Monitoring*. 2024. arXiv: 2404.10446.

[73] Chris Linegar, Winston Churchill, and Paul Newman. "Work smart, not hard: Recalling relevant experiences for vast-scale but time-constrained localisation". In: *2015 IEEE International conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 90–97.

[74] J. Jackson et al. "Experimental drought reduces the productivity and stability of a calcareous grassland". In: *Journal of Ecology* (Feb. 2024).

[75] John Jackson et al. "Short-range multispectral imaging is an inexpensive, fast, and accurate approach to estimate biodiversity in a temperate calcareous grassland". In: *Ecology and Evolution* 12.12 (Dec. 2022).

[76] MR Balme et al. "The 2016 UK Space Agency Mars Utah Rover Field Investigation (MURFI)". In: *Planetary and Space Science* 165 (2019), pp. 31–56.

[77] Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection". In: *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer. 2006, pp. 430–443.

[78] Michael Calonder et al. “BRIEF: Computing a local binary descriptor very fast”. In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2011), pp. 1281–1298.

[79] David Nistér, Oleg Naroditsky, and James Bergen. “Visual odometry”. In: *Proceedings of the Computer Society Conference on Computer Vision and Pattern Recognition*. 2004.

[80] Mark Cummins and Paul Newman. “FAB-MAP: Probabilistic localization and mapping in the space of appearance”. In: *The International Journal of Robotics Research* 27.6 (2008), pp. 647–665.

[81] Jonas Frey et al. “Fast Traversability Estimation for Wild Visual Navigation”. In: *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation, July 2023.

[82] Michal Staniaszek et al. “Difficulty-Aware Time-Bounded Planning Under Uncertainty for Large-Scale Robot Missions”. In: *2023 European Conference on Mobile Robots (ECMR)*. 2023, pp. 1–7.

[83] Nick Hawes, Michael Zillich, and Patric Jensfelt. “Lessons Learnt from Scenario-Based Integration”. In: *Cognitive Systems*. Ed. by Henrik I. Christensen, Geert-Jan M. Kruijff, and Jeremy L. Wyatt. Vol. 8. Cognitive Systems Monographs. Springer Berlin Heidelberg, Apr. 2010, pp. 423–438.

[84] Pat Marion et al. “Director: A User Interface Designed for Robot Operation with Shared Autonomy”. In: *Journal of Field Robotics* 34.2 (2017), pp. 262–280. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21681.