2024 International Conference on Unmanned Aircraft Systems (ICUAS)
June 4-7, 2024 | Chania, Crete, Greece

A Behavior Tree Approach for Battery-Aware Inspection of Large Structures Using Drones

Bernardo Martinez Rocamora Jr., Paulo V. G. Simplicio, and Guilherme A. S. Pereira

***Abstract*— Electric multi-rotor drones have been used to inspect several structures, including large buildings and dams. In these inspections, energy consumption is a concern. To prevent the drone from running out of battery, commercial drones usually come back to their home position when the battery level reaches a minimum threshold. The pilots then need to replace the battery and use their own experience to restart the inspection mission approximately from where it ended before the drone returned home. Instead of relying on the human operator, in this paper, we automate this process using behavior trees, which is an effective way to perform autonomous mission control and supervision. By integrating battery management strategies into a behavior tree framework, this paper demonstrates the drone's adaptive and resilient decision-making when confronted with limited power constraints. We implemented our methodology using a commercial drone and tested the proposed ideas in a photogrammetry-based inspection task.**

I. Introduction

Autonomous inspection robots and drones can have their behavior controlled using many different frameworks, from hard-coding sequential tasks to a more elegant solution like a state machine. State machines are defined through a set of states and state transitions, which can be triggered from predefined conditions. While state machines are a consolidated component of Robotics, they fail to scale up since the number of transitions grows fast with the number of states. Also, every time a new state needs to be added, the entire state machine is affected.

Behavior trees (BTs) have been proposed as an alternative to state machines to help mitigate these issues [1]. BTs offer a more flexible and hierarchical control system, enabling complex decision-making through a series of behaviors and priorities. The advantages of behavior trees can be summarized into three words: hierarchical, modular, and reactive. Behavior trees are hierarchical because depending on the position of a behavior on the tree, it will be given priority. They are modular because modifications are locally contained, and new behaviors can be grouped in sub-trees that can be used across different projects. Finally, they are reactive because the whole tree is run in a loop, and when something changes, the final robot's behavior can be changed accordingly. By integrating a behavior tree, robots in inspection tasks can show an advanced level of autonomy, ensuring efficient and reliable navigation through complex environments.

This work was supported by the Department of Energy of the United States of America (DOE), Grant DE-FE0032206.

The authors are with the Department of Mechanical, Materials and Aerospace Engineering, Benjamin M. Statler College of Engineering and Mineral Resources, West Virginia University. Morgantown, WV, 26501, USA. bm00002@mix.wvu.edu, pvg00001@mix.wvu.edu, guilherme.pereira@mail.wvu.edu

Fig. 1: Example of an inspection mission in a large tailing dam (used to store waste byproducts of mining). The drone takes off at the green dot with a mission defined by a lawn-mower pattern path, which would lead to the blue dot when complete. However, since the mission given by the user was too long for the battery capacity, a low-battery event happened. The drone decides to return to its home position and land. After the battery was replaced, the drone resumed the mission from the point it was interrupted. The blue path was followed before the low battery event and the red path after the battery was replaced.

This paper proposes the use of behavior trees to manage the battery level of an autonomous UAV performing inspection of large structures, where the limited flight time of the UAV may require battery replacements/recharges during the mission. An example of such a scenario, which motivates this work is shown in Fig. 1. Since most of the inspection trajectories for large structures do not include any information about the UAV's battery, in some cases, the operator needs to conduct several manual activities (e.g., change the trajectory, restart the recording, fly the drone manually close to the trajectory) to resume an inspection that was interrupted because it was too long to be executed with a single battery. Through the behavior tree architecture, nodes can be dedicated to monitoring and managing battery levels as an integral part of decision-making. These nodes can encompass actions such as periodic checks on the battery status, assessing energy consumption rates during different tasks, and dynamically altering the robot's behavior based on the remaining charge. Different from other implementations, our tree is deployed offboard and also encodes behaviors that allow an operator to interact with the UAV, turning it off

and replacing the battery. The drone resumes the operation automatically after that. In this paper the objective was to develop a BT that allows for adaptive decision-making regarding the drone's actions concerning its battery status, enabling it to prioritize tasks or modify its route to ensure a safe return within the constraints of available power.

This paper is structured as follows. Section II provides an overview of existing behavior tree applications in autonomous systems. Section III provides a brief explanation of how behavior trees and their important components work. Section IV presents the design of a behavior tree for the proposed application, elucidating their hierarchical structure and decision-making capabilities for UAV navigation and inspection tasks. Section V presents experiments that show the proposed BT controlling the behavior of a commercial drone during an inspection task, and finally, Sect. VI summarizes key insights, highlighting the potential of behavior tree-based autonomous systems in facilitating the inspection of large structures.

II. Related Work

Initially conceived in the gaming industry to streamline decision-making processes for non-player characters (NPCs) in video games [2], behavior trees (BTs) provided a structured yet flexible framework to define and prioritize various behaviors. Their hierarchical nature allowed developers to create complex and adaptive character actions by arranging individual behaviors in a tree-like structure while increasing modularity, reactivity, robustness, and safety [3], [4]. Over time, behavior trees started being adopted outside of the gaming industry and found applications in diverse fields such as robotics, autonomous systems, and AI-driven simulations. Their ability to handle dynamic and reactive behaviors led to adoption across industries seeking intelligent decision-making systems. Additionally, they have been proven to generalize other architectures like teleo-reactive programs [5], sequential compositions, subsumption architecture, and state-machines [1].

Today, behavior trees stand as a promising component in the design and development of sophisticated autonomous systems. It has been adopted by many Robotics companies and they have been tested in many types of robotic systems: soccer robots [6], mobile robots [7], unmanned aerial vehicles [3], surgical robots [8], autonomous underwater vehicles [9], manufacturing robots [10], wheeled-legged robots [11], robotic assistive systems [12], [13] and, disaster-response robots [14]. They have been central to the solution that led to the "Most Sectors Explored" award of the DARPA Subterranean Challenge [15].

A review of the application of behavior trees in Robotics is provided by [16] and practical aspects of BTs in Robotics are analyzed in [17]. In that review, different libraries are compared and a suggestion of how BTs can be fit in a robotic software architecture is provided. A survey of the use of BTs in Robotics and AI was conducted by [18]. The authors note that recent publications are either related to the use of BTs in new domains in Robotics (manipulators, mobile robots, aerial robots, and other types of robots) and Gaming (dialogue, platforms, FPS, and RTS games), or to their design method (manual design, learning from demonstration, reinforcement learning). The performance of the most important behavior tree and state machine libraries are compared in [19]. The authors note the rapid increase in BT libraries usage, especially "py_trees" [20] and "BehaviorTree.CPP" [21], in Robotics.

Hand-crafting is a major component of designing autonomy frameworks, and that is not different for behavior trees. In most applications, the BT is designed by an experienced user, that knows how to make the robot fit for real-world use. However, a lot of the recent developments have been focused on creating BTs in different manners: automatic node reordering, temporary modifications, dynamic construction, learning from demonstration, learning from experts, etc. Utility BTs [22] have been proposed to reorder nodes in the tree based on information about their utility (what could be gained). Utility BTs have been extended as Stochastic BTs [23] with the idea that the utility could consider success probabilities.

Parallelization is another important aspect of robotic behaviors. While most implementations provide a parallel node, that can trigger more than one behavior at the same time, these behaviors are assumed to be independent. The consequence is that parallel nodes are rarely used, because concurrent actions may lead to unexpected problems. Concurrent BTs [24], [25] try to mitigate this issue by including the notions of progress and resource usage. In [26], parallel nodes were extended to achieve synchronization of parallel tasks.

This paper applies BTs to control the execution of a drone-based inspection task while managing the battery level. Our proposed solution uses a state-of-the-art BT implementation that considers parallel nodes, selectors, and sequencers. The next section introduces these concepts, which will be needed to completely understand our approach.

III. Background

Behavior trees operate as hierarchical structures governing decision-making in autonomous systems. Comprising nodes interconnected in a tree-like fashion, they organize behaviors into a clear and prioritized flow. An example of BT is shown in Fig. 2. A BT is activated at a fixed frequency. When this happens, a signal (called "tick") traverses from the root node down the branches, evaluating nodes based on their type and conditions.

When a node is ticked, it can return one of three responses: running (when the node process is ongoing), success (when the node process has been successful in reaching its desired behavior), or failure (when the node has failed in reaching its desired behavior). Each traversal cycle updates the state of the tree, allowing for dynamic responses to changing environments or inputs.

At the top of the BT, the root node oversees the decision-making process. Below the root node, the BT branches into two major categories of nodes: internal nodes, also known as control nodes, and leaf nodes, also known as execution

nodes. The most popular behavior tree libraries provide a blackboard, which is usually implemented as a hash table, that can store data that is shared globally with the nodes.

Internal nodes are typically: *Sequence*, *Selector*, *Parallel*, or *Decorator* nodes. A *Sequence* node requires that all its children, executed in sequence, return success so that it can also return success. It returns failure if any of its children fails. A *Selector* node requires that at least one of its children, tested in sequence, return success to be successful. It returns failure only if all its children also return failure. *Parallel* nodes tick all of their children every time they are ticked. The children are sequentially ticked. While some of the children can trigger some parallel processes in the background, from the tree's perspective, the processes happen in a single-threaded operation.

One recent addition to internal nodes of behavior trees is a feature called "memory", which is useful for nodes with long sequences of tasks that do not need re-execution of all their children. When a node is configured with memory, if one of its children returns running in the previous tick, in the next tick the node will skip the children executed before and return directly to the one that was previously running.

Decorator nodes can only have a single child and can be custom-made. They modify or add functionality to their child nodes, altering their behavior dynamically. For example, an "Inverter" decorator node will return failure when its child returns success and vice-versa, and a "One Shot" node will only execute its children once. Execution or leaf nodes are typically *Conditions* or *Actions*. *Conditions* check a specific variable, typically stored in the blackboard, and return success or failure depending on the value of that variable. *Action* nodes represent the lowest level of behavior, in which the robot executes specific tasks or actions.

Although several software libraries implement BTs [19], in this paper, we used "py_trees" [20]. The development of "py_trees" was primarily motivated by robotics, in particular, the higher level of decision-making for a single robot, i.e. the scenario/application layer. The project itself was developed in Python due to its short learning curve and faster development cycle. It was developed for medium-scale use, considering a single robot and hundreds of different behaviors. Finally, it was developed to be reactive, but not real-time reactive. The assumption is that a real-time decision-making process should be handled by a control layer and the higher-level decision-making can operate with a latency of around 200 to $1000\,\mathrm{ms}$, which would not produce a noticeable delay in the perspective of the humans interacting with the robot. Additionally, the "py_trees" project includes "py_trees_ros", a tree manager and predefined behaviors designed for use specifically with the Robot Operating System 2 (ROS 2), and "py_trees_ros_viewer" a Qt/ROS2 implementation of a runtime visualization tool.

IV. Methodology

The basic mission for an autonomous inspection drone can be described as follows. The mission starts with a drone saving the home position and loading a list of waypoints for the inspection. The definition of the waypoints considers the requirements of the task and the UAV payload [27], [28], and is not discussed in this paper. After loading the waypoints, the drone takes off and starts going to each of these waypoints in sequence. The vehicle may take pictures, videos, and point-cloud data while moving from waypoint to waypoint. It may hover or return to home at the end of the trajectory.

We propose a behavior tree (BT) that executes/coordinates this mission but also checks for the battery state and GPS coordinates while flying. Our proposed behavior tree, shown in Fig. 2, works as follows. At the root of the BT, there is a parallel node. This means that the two sides of the tree will run every time the root is ticked. The root is ticked at $1\,\mathrm{Hz}$. Assuming a ROS 2 implementation, on the left, we have subscribers to ROS topics "drone/state", "gps/location" and "battery/percentage", which correspond to the drone's internal state (LANDED, HOVERING, etc), GPS coordinates, and battery level. These nodes (StateToBlackboard, BatteryToBlackboard, and GPSToBlackboard) listen to their corresponding topics and save the most updated data into the blackboard. On the right, there is a sequence of tasks that encode the mission of the drone.

With higher priority (and executed first) there is a sequence sub-tree that does pre-flight tasks. First, and only once (due to the OneShot decorator), it will set a home position (SetHome) and request a list of waypoints (Plan), which defines the mission, as mentioned above. Once this is achieved, it will make sure that the drone state is "LANDED" and take off. The take-off sequence is defined by sending the take-off command (Takeoff), setting an initial goal (SetGoal), and sending a command to move the drone to this goal (simply defined by the home position here). Since these actions are guarded by a check on the drone's internal state, this sub-tree does not result in any other actions until the drone returns, lands, and needs to take off again. Notice in Fig. 2 that some of the blocks show an encircled M, which means that they use memory to remember from with the child to resume if any of them return running.

The next part of the mission sequence is a selector node. The selector node has a battery emergency sub-tree, a set waypoint sub-tree, and an idle node as children. The battery emergency sub-tree is only activated when the battery level is below a predefined threshold. In this case, the tree will execute a sequence: push the current coordinate and the last popped goal to the list of waypoints (Replan), move to the home location, land, and stay idle (while the battery is changed), and, if was landed, take off after a command from the human user (pre-flight sub-tree). Before landing, the drone checks if it is "HOVERING", which denotes that it has arrived at the home location. It is important to mention that the execution of the behavior tree continues while the battery is replaced. In our implementation, discussed in the next section, this is possible because the drone is controlled by a remote computer, which is not turned off when the battery is removed.

If no emergency is present, then the drone either sets a

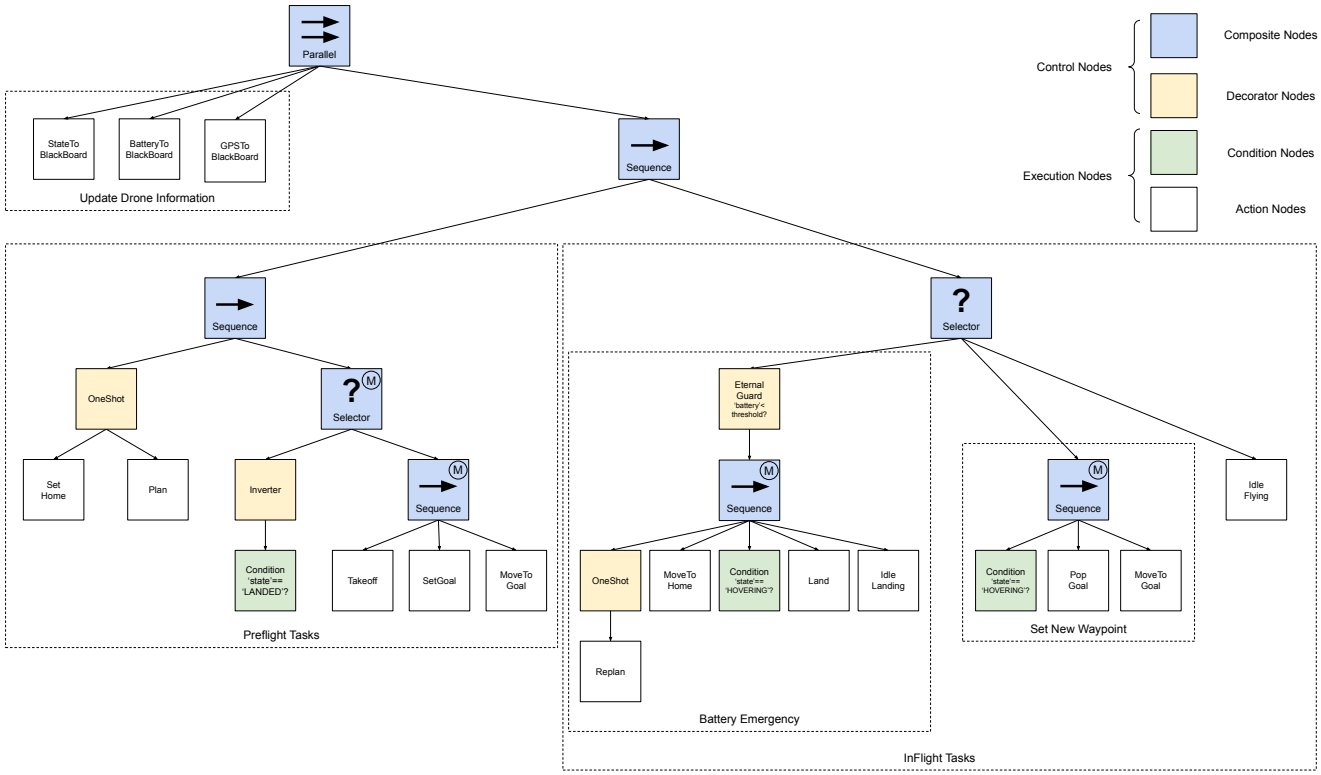

Fig. 2: Proposed Behavior Tree. The BT encodes a structured approach for a UAV conducting an inspection task. It saves the drone state internally and executes the mission. The mission begins with pre-flight checks and then navigates through the waypoints while the battery permits. The UAV returns to home when necessary, ensuring a systematic and efficient inspection process while prioritizing safety over data collection.

new waypoint or remains idle. Being idle means that it is either still flying to a waypoint or finished the mission. A new waypoint is set whenever the drone still has waypoints left and is "HOVERING", which means that the drone has arrived at the previously defined waypoint. Setting a waypoint is done using two behaviors: one that pops a waypoint of the list defined by the planning behavior and another that sets a goal for the drone waypoint follower. Notice that, after an emergency landing, the drone takes off and flies to where it stopped the mission since that coordinate was saved at the top of the list of waypoints. The next section presents the experiments performed to test the proposed tree.

V. Experiments

This section discusses the hardware and software used to test our methodology and present our experimental results.

A. UAV

We tested our approach with the Parrot ANAFI USA drone. This is a high-end drone designed for the U.S. Army and other government agencies. It is P53 certified (dust and rain resistant), National Defense Authorization Act (NDAA) and Trade Agreements Act (TAA) compliant, and Blue sUAS program approved [29]. It has a maximum flight time of $32\,\mathrm{min}$, a maximum horizontal speed of $14.7\,\mathrm{m\,s^{-1}}$, and a maximum ascent speed of $4\,\mathrm{m\,s^{-1}}$. The drone's fast-charging smart battery has a capacity of $3400\,\mathrm{mA\,h}$ and can be charged in $2\,\mathrm{h}$ with a USB-PD charger. The drone has two image sensors, a digital zoom of $32\times$, and a video resolution of 4K/FHD/HD. The video format is MP4 (H.264) and the photo resolutions are Wide: 21 megapixels ($84°$ FOV) and Rectilinear: up to 16 megapixels (up to $75.5°$ FOV).

B. Software

In our implementation, a behavior tree manager created using "py_trees_ros" is integrated into a ROS 2 node. The node connects to a bridge that communicates with the drone. The bridge can provide information about the UAV's sensor data, state, etc., and also send commands to control the UAV. Commands include arming, disarming, taking off, sending waypoints, controlling the camera, and landing, among others. The BT manager also communicates with a path planner ROS 2 node that provides a list of waypoints (defined by the user or automatically [27], [28]) that need to be followed by the UAV. While inspection tasks may include capturing images, collecting data, and transmitting it in real-time for analysis, this was not considered explicitly by our BT. However, the inclusion of these behaviors is straightforward as long as they are offered as ROS 2 services or actions. In our experiments, this was implemented by turning on the video capture mode of the drone upon taking off. Also, because we used the Parrot Anafi USA drone for our tests, the bridge used was the driver proposed in [30]. Figure 3 shows the UAV's control architecture. We made the developed software

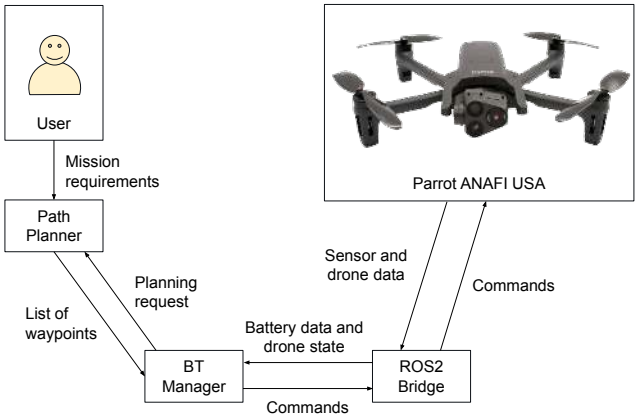

Fig. 3: Robot Control Architecture using a Behavior Tree Manager. The BT Manager will request waypoints to a Path Planner and save them on the blackboard. The mission tries to execute the planned path by interacting with the UAV through a ROS 2 Bridge [30]. It monitors sensor data and reacts if some fault condition is observed .

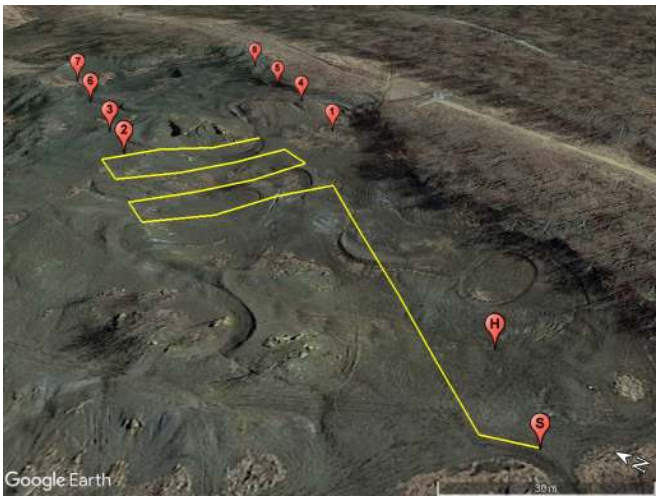

Fig. 4: The mission planning. Before the experiment, the operator selected 8 waypoints (marked 1 to 8) from Google Earth and saved them in a file. Once the operator gets to the experiment location (ash pit), using the drone itself, a position is selected for the base station (S) from where the drone is initially launched, and for the home position (H), which is close to the base and where the drone can land safely and the battery can be replaced.

available at `https://bitbucket.org/wvufarolab/faro_behavior_trees/`. Our experiments are detailed next.

C. Results

We flew our drone in an abandoned coal ash pit. The goal was to create a tridimensional (3D) map of the region using photogrammetry. For this task, a lawn-mower path was created to cover the ash pit. The waypoints are loaded from a text file and provided to the behavior tree. They are obtained beforehand with the aid of Google Earth as explained in the caption of Fig. 4. To make sure that the drone comes back home at the end of the mission, we included the home position at the end of the file. Although this procedure is not explicitly shown on the BT of Fig. 2, it is done in the one-shot plan block before the flight. For this mission, we did not use the BT to control the drone's camera. Instead, our code points the camera down and starts recording a video before the BT starts running.

Since the drone's battery allows a flight of up to 32 minutes and the ash pit to be surveyed has a small area, we simulated an event of battery emergency by setting the critical threshold for the battery at a high level (45%), instead of using the threshold set internally by the drone (around 15%). Besides that, for this specific experiment the drone started the mission with a battery level of 61%. Figure 5 shows the resulting trajectories using the behavior tree proposed in Figure 2. As shown by the red line, when the battery reaches the threshold, the drone flies back to the home location and lands. It takes off after the battery is replaced and resumes the mission. As can be seen in the figure, the drone then finishes the trajectory and flies back to the home position.

Fig 6 presents the 3D map of the area of interest obtained from the video collected during the mission in Fig. 5. This map was constructed with the use of the software COLMAP [31]. Using this software, the map is created in two steps. In the first step, the camera pose and a sparce point cloud are computed (left-hand side of Fig 6). In the second space, the information in the first step is then used to create a dense point cloud (right-hand side of Fig 6). Notice that the final map is complete, indicating that the drone followed the entire path despite the need for battery replacement. Notice also that, because the camera was turned on before take off, images obtained in between the home position and the actual inspection path were also considered for map building. This could be avoided if camera control nodes are added to the BT.

VI. Conclusions and Future Work

The exploration of autonomous inspection through behavior tree-driven UAVs presents a transformative approach to enhancing safety, efficiency, and precision. In this paper, we proposed and validated in practice a behavior tree that provides a battery management capability, which makes the drone return to home automatically when a certain condition is met and resumes the mission after the battery is replaced.

In this work, we added a simple battery percentage threshold to trigger a return-to-home behavior, but in our next steps, we desire to include estimates of the required battery to finish the remaining trajectory and return home. The BT can also be extended to take photos, record videos, and many other behaviors while executing the trajectory, thus proving itself useful for inspection applications.

The inclusion of such features will support our goal of controlling our drone in the inspection of large coal-mine

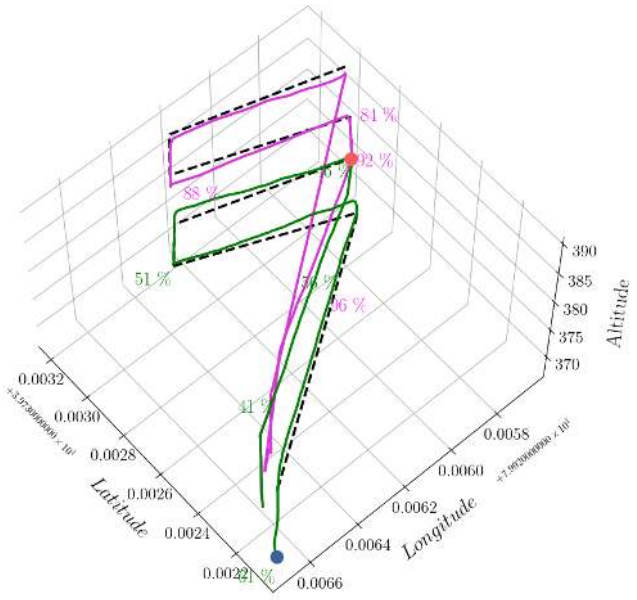

Fig. 5: Mission performed by the drone in the ash pit. The desired trajectory is represented by the dashed black line, while the trajectory performed by the drone is depicted by the green and magenta lines. The green line represents the drone path before the low battery event. The drone takes off from the blue dot and follows the desired trajectory until the battery status changes to low (red dot). At this point, the drone returns home, the battery is replaced, and the drone continues the mission, now represented by the magenta line. The battery percentage is also shown in different spots of the trajectory.

tailing dams, like the one shown in Fig. 1. We expect to create a system that performs a fully autonomous inspection of several structures of the dam, including the embankment and all the spillways, in a couple of hours. The current procedure is executed by a human inspector in a couple of days.

Future work includes the comparison between BTs and other competitive approaches, such as state machines. Another promising avenue for future work involves the integration of learning techniques to design the behavior tree architectures automatically. By incorporating reinforcement learning or other machine learning models, UAVs could learn and refine their behaviors based on experience and feedback. This advancement would enable these systems to autonomously evolve their decision-making processes, potentially optimizing inspection strategies over time.

REFERENCES

[1] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[2] M. Mateas and A. Stern, "A behavior language for story-based believable agents," *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 39–47, 2002.

[3] P. Ogren, "Increasing modularity of uav control systems using computer game behavior trees," in *Aiaa guidance, navigation, and control conference*, p. 4458, 2012.

[4] M. Colledanchise and P. Ögren, "How behavior trees modularize robustness and safety in hybrid systems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1482–1488, IEEE, 2014.

[5] M. Colledanchise and P. Ögren, "How behavior trees generalize the teleo-reactive paradigm and and-or-trees," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 424–429, IEEE, 2016.

[6] R. H. Abiyev, S. Bektas, N. Akkaya, and E. Aytac, "Behavior trees based decision making for soccer robots," *Recent Advances in Mathematical Methods, Intelligent Systems and Materials*, 2013.

[7] F. G. Rosas, F. Hoeller, and F. E. Schneider, "Autonomous robot exploration and measuring in disaster scenarios using behavior trees," in *2020 IEEE 10th International Conference on Intelligent Systems (IS)*, pp. 469–474, IEEE, 2020.

[8] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3868–3875, IEEE, 2015.

[9] S. Bhat and I. Stenius, "Controlling an underactuated auv as an inverted pendulum using nonlinear model predictive control and behavior trees," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12261–12267, IEEE, 2023.

[10] G. Kokotinis, G. Michalos, Z. Arkouli, and S. Makris, "A behavior trees-based architecture towards operation planning in hybrid manufacturing," *International Journal of Computer Integrated Manufacturing*, pp. 1–26, 2023.

[11] A. De Luca, L. Muratore, and N. G. Tsagarakis, "Autonomous navigation with online replanning and recovery behaviors for wheeled-legged robots using behavior trees," *IEEE Robotics and Automation Letters*, 2023.

[12] M. Behery, M. Trinh, C. Brecher, and G. Lakemeyer, "Assistive robot teleoperation using behavior trees," *arXiv preprint arXiv:2303.05177*, 2023.

[13] S. Drolshagen, M. Pfingsthorn, and A. Hein, "Context-aware robotic assistive system: Robotic pointing gesture-based assistance for people with disabilities in sheltered workshops," *Robotics*, vol. 12, no. 5, p. 132, 2023.

[14] F. G. Rosas, F. Hoeller, and F. E. Schneider, "Automated environmental mapping with behavior trees," *Advances in Intelligent Systems Research and Innovation*, pp. 61–85, 2022.

[15] G. Best, R. Garg, J. Keller, G. A. Hollinger, and S. Scherer, "Multi-robot, multi-sensor exploration of multifarious environments with full mission aerial autonomy," *The International Journal of Robotics Research*, p. 02783649231203342, 2023.

[16] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wasowski, "Behavior trees in action: a study of robotics applications," in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, pp. 196–209, 2020.

[17] M. Colledanchise and L. Natale, "On the implementation of behavior trees in robotics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5929–5936, 2021.

[18] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[19] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior trees and state machines in robotics applications," *IEEE Transactions on Software Engineering*, 2023.

[20] D. Stonier, "py_trees." `https://py-trees.readthedocs.io/`. Last accessed on 2023-12-12.

[21] D. Faconti, "BehaviorTree.CPP." `https://www.behaviortree.dev/`. Last accessed on 2023-12-12.

[22] B. Merrill, "Building utility decisions into your existing behavior tree," *Game AI Pro 360: Guide to Architecture*, pp. 81–90, 2019.

[23] M. Colledanchise, A. Marzinotto, and P. Ögren, "Performance analysis of stochastic behavior trees," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3265–3272, IEEE, 2014.

[24] M. Colledanchise and L. Natale, "Improving the parallel execution of behavior trees," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7103–7110, IEEE, 2018.

[25] M. Colledanchise and L. Natale, "Handling concurrency in behavior trees," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2557–2576, 2021.

[26] Y. Ma, J. Zhang, Y. Wu, and Y. Wang, "Follow me: Hierarchical parallel execution synchronization in behavior trees," in *2021 IEEE*

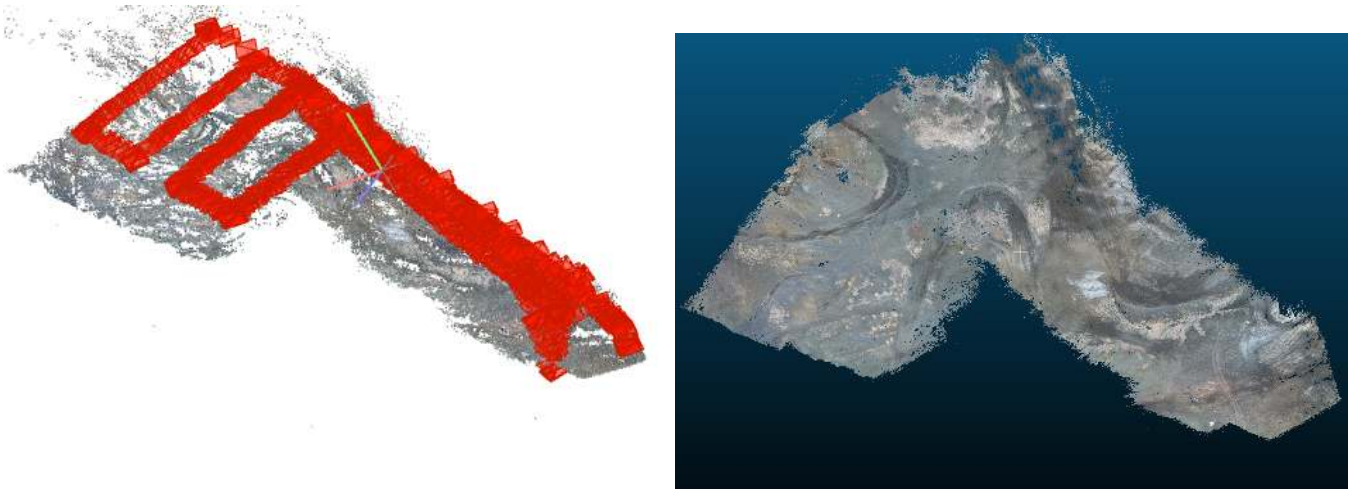

Fig. 6: Photogrammetry-based 3D mapping of the area of interest was performed with the date collected during the experiment shown in Fig. 5. The sparse point cloud is depicted on the left, while the dense 3D map is showcased on the right.

International Conference on Robotics and Biomimetics (ROBIO), pp. 613–618, IEEE, 2021.

[27] G. S. Avellar, G. A. Pereira, L. C. Pimenta, and P. Iscold, "Multi-uav routing for area coverage and remote sensing with minimum time," *Sensors (Switzerland)*, vol. 15, pp. 27783–27803, 11 2015.

[28] K. Y. Samarakoon, G. A. Pereira, and J. N. Gross, "Impact of the trajectory on the performance of rgb-d slam executed by a uav in a subterranean environment," in *2022 International Conference on Unmanned Aircraft Systems, ICUAS 2022*, pp. 812–820, Institute of Electrical and Electronics Engineers Inc., 2022.

[29] "Blue UAS - Defense Innovation Unit." Available: `https://www.diu.mil/blue-uas`. Accessed: January 15, 2024.

[30] A. Sarabakha and P. N. Suganthan, "anafi_ros: from Off-the-Shelf Drones to Research Platforms," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1308–1315, 2023.

[31] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.