

A Comprehensive Investigation into the Use of Behavior Trees in the Coordination and Control of Swarm Robotics Systems

Professor: Nicola Conci

Student: Zaida Brito Triana MAT. 230400

Department of Industrial Engineering - University of Trento

Project Repository: <https://github.com/zaidabri/BehaviorTreeSwarmRobotics>

Abstract - This paper explores the implementation of Behavior Trees (BTs) in the field of Swarm Robotics. Behavior Trees are a type of hierarchical decision-making structure that can be used to create complex autonomous behaviors. This research begins by providing a comprehensive overview of the theoretical foundations of BTs and their role in the design and control of robotics systems. This includes a review of the current state of the art in BT implementation in robotics, highlighting the key challenges and limitations that have been identified in previous research, as well as the most promising approaches and strategies that have been developed to overcome these challenges. Then, it continues with an overview of the advantages and disadvantages of BTs over other traditional systems like Finite State Machines (FSMs). Also, it discusses and explains the results obtained and the methodology and framework used in the example code implementation developed for this paper. Through this analysis, the research aims to provide a clear understanding of the key considerations and issues that must be taken into account when using BTs in swarm robotics, as well as the implementation possibilities that BT offers. The general conclusion of this paper is that BTs offer a viable and robust approach to swarm robotics due to their flexibility and scalability.

The simulation scenario of swarm robotics using behavior trees has been developed in C++.

Keywords - Behavior Trees (BTs), Finite State Machines (FSMs), Swarm Robotics, Autonomous Robots

1 Introduction

Behavior Trees were originally designed to describe the behavior of autonomous non-player characters (NPC) in computer games. Non-player characters, like autonomous robots, have the ability to respond and make choices in situations that are both complex and uncertain. (Heckel et al., 2010). Their definition and key concepts for their application were initially published by R. G. Dromey in 2001, (Dromey, 2001) but one of the first significant projects to implement behavior trees was in the development of the Halo 2 and Halo 3 video games (Isla, 2005) (Isla, 2008). Despite of the origin of Behavior Trees, lately, their potential in the robotics and control community has been gaining attention (Ghzouli et al., 2020). This paper examines the potential of behavior trees in swarm robotics. However, the focus of this paper will also be on broader advancements in the field, as these developments will facilitate future improvements in the implementation of swarm robotics architecture.

Before proceeding, it is crucial to provide a succinct explanation of the fundamental concepts and principles of swarm robotics. Swarm robotics is a field of robotics that studies the behavior of groups of robots, called swarms, that are able to coordinate their actions to accomplish tasks (Legarda Herranz, 2021). The core principles of a robot's swarm are based upon the assumptions that the agents are able to freely move within an environment, either UAVS, as seen for example in (Klockner, 2013), or mobile robots, for instance.



Applications of swarm robotics include search and rescue, environmental monitoring, transport (Legarda Herranz, 2021), and warehouse automation, like the example implemented in this paper.

The way the robot swarm behaves as a community is based on the interactions of each robot with its peers and the environment (Kuckling et al., 2021). Generally, swarms are made up of same type robots, though there are examples of swarms with different types of robots (Dorigo et al., 2013).

1.1 Problem formulation

Building upon the previously discussed information, this paper aims to investigate the following research question:

To what extent does the use of Behavior Trees improve the coordination and control of swarm robotics systems?

The solutions presented in this paper aim to showcase the current state-of-the-art in the development of swarm robotics by utilizing behavior trees as the foundation to develop the coordination and control of these systems. Additionally, a demonstration of the practical application of this approach will be included through a simulated implementation of the logic governing an autonomous agent responsible for operating orders within a warehouse environment. This will help to showcase the potential capabilities and real-world usefulness of utilizing behavior trees in swarm robotics systems.

1.2 Roadmap

Two Ph.D. theses will be used as a baseline for analysis and discussion in this research paper. The first thesis, titled "Onboard Evolution of Human-

Understandable Behaviour Trees for Robot Swarms," was written by Jones.SW., and the second titled "Behavior Trees in Robotics" was written by Colledanchise. M. This second thesis served as a precursor to the book "Behavior Trees in Robotics and AI," which is highly regarded and widely studied in the field of behavior trees in robotics and artificial intelligence. This book will also be referenced and incorporated into the research.

The example presented in this paper was designed and implemented using the "BehaviorTree.cpp" framework (Faconti, 2022), written in C++ 17. Additionally, the "Groot" editor (Faconti, 2019) was used to graphically create the behavior trees. Both of these programs, maintained and regularly updated by Faconti.D, have been extremely beneficial in the implementation of behavior trees in this field.

2 Literature Review

This section aims to provide an overview of the current state-of-the-art in the field of swarm robotics, followed by an examination of the theory and principles of behavior trees and a comparison with other commonly used methods in the coordination and control of swarm systems.

2.1 Behavior Tree Theory

Behavior trees are a type of decision tree used in artificial intelligence and robotics to specify the behavior of a system. They are composed of a hierarchy of nodes, each of which represents a specific behavior or action. The tree is traversed from the top down, and the actions of the nodes are executed in order based on the results of

previous actions and the conditions specified in the tree.

Behavior trees have been gaining attention in robotics because they allow the robot to make decisions based on its current situation and available information. For example, like in the demo developed, a behavior tree might specify that a robot should move to a particular location, but if the robot encounters an obstacle, it should try to go reroute and go around or stop moving.

Behavior trees are a useful tool for specifying complex behaviors because they allow for clear and concise representation of the decision-making process and enable easy modification of the behavior by modifying the tree structure. They are also easy to understand and debug, making them a feasible choice for specifying the behavior of robots and other intelligent systems

Corresponding to the matter in this research we can define that a Behaviour Tree is a hierarchical tree of nodes, where the leaf nodes correspond to executable programs that correspond to robot actions, such as pick up an object, stop, or perform a gesture. A behavior tree (BT) begins by running its root node, which sends out signals at a specific frequency called *ticks*. These ticks are then passed on to the root node's children. A node can only be executed if it receives these ticks. When the root of a tree sends a *tick* signal, it travels towards the leaves of the tree. Each *TreeNode* that receives the signal will then execute a callback function. This callback can return one of three possible results: success, failure, or running. If the running result is returned, it indicates that the action is still in progress and needs more time to be completed.

Tree nodes with children are responsible for passing on the *tick* signal to them. The rules for when and how many times the *tick* is passed down to the

children may vary between tree nodes. The actual commands of the behavior tree are found at the leaf nodes, and it is there that the tree interacts with the rest of the system. Commonly, these leaf nodes are Actions nodes.

2.1.1. Node types

A review of the most commonly used node types in industry projects and research is presented below.

Outline of the most common control flow nodes:

Sequence nodes: A sequence node is the most common type of node in a behavior tree. It is used to execute a series of actions consecutively. When a sequence node with more than one child receives a tick, it will first *tick* its first child; if that child returns a result of success, the sequence node will then *tick* its next child, continuing this process until all of its children have been *ticked*.

The sequence node will return a result of success if all of its children return a result of success. If any of the children return a result of failure, the sequence node will immediately return a result of failure and stop executing the behavior tree.

Algorithm 1 Sequence node

```

1: for  $i \leftarrow 1$  to  $N_{child}$  do
2:    $childReturn \leftarrow Tick(C_i)$ 
3:   if  $childReturn = RUNNING$  then
4:     return  $RUNNING$ 
5:   if  $childReturn = FAILURE$  then
6:     return  $FAILURE$ 
7:   end if
8: end if
9: end for
10: return  $SUCCESS$ 

```

Fallback or Selector nodes: The series of actions specified by these nodes is *ticked* in order. If a child returns failure, it ticks the next one. If success or running is returned by one child, the selector does not *tick* any more of its children, and the same result is returned. In the event that all of the actions fail, a failure result is returned by the selector node.

Algorithm 2 Selector or Fallback node

```

1: for  $i \leftarrow 1$  to  $N_{child}$  do
2:    $childReturn \leftarrow Tick(C_i)$ 
3:   if  $childReturn = RUNNING$  then
4:     return  $RUNNING$ 
5:   if  $childReturn = SUCCESS$  then
6:     return  $SUCCESS$ 
7:   end if
8: end if
9: end for
10: return  $FAILURE$ 

```

Parallel nodes: These nodes indicate that a series of actions should be carried out at the same time. If any of the child nodes are still in progress, the node will return a running status. If the number of child nodes that have successfully completed is above a certain threshold, which is set by the user, the node will return a success status. If it is below the threshold, it will return a failure status. Although they were not used in this research, parallel nodes are mentioned here for completeness.

Decorator nodes: These nodes have a single child node, and they modify the return status of the child node according to a rule specified by the user. They also selectively execute the child node based on a predetermined rule. In every case, if the child node returns a running status, the decorator node will also return running. Algorithm 3 presents the pseudocode of the commonly used decorator node *RetryUntilSuccessful*.

Algorithm 3 Decorator node

RetryUntilSuccessful with N attempts

```

1: for  $i \leftarrow 1$  to  $N_{child}$  do
2:   for  $n \leq N$  do
3:     if  $childReturn = FAILURE$  then
4:        $n = n + 1$ 
5:     if  $childReturn = SUCCESS$  then
6:        $n = N$ 
7:       return  $SUCCESS$ 
8:     end if
9:   end if
10: end for
11: end for
12: return  $FAILURE$ 

```

Outline of the general characteristics of control execution nodes:

Action nodes: These nodes are the most common type of leaf node, and they represent a specific action or behavior that should be performed. When it receives a tick, an action node will execute a command. If the action is completed successfully, it will return a success status. If the action fails, it will return a failure status. While the action is in progress, it will return a running status. An action node may read from the blackboard (as described in Section 2.1.2) and modify the values of certain variables by writing to them.

Condition nodes: Control execution nodes determine the success or failure of a particular action based on the evaluation of a specific condition. They are frequently used with decorator nodes to specify the circumstances under which a specific behavior should be executed. It is important to note that a control execution node will never return a running status. They may read from the blackboard, but they cannot modify its contents.

The table above, inspired by (Colledanchise & Ögren, 2017), summarizes the characteristics of the most common node types







Node type	Symbol	RUNNING	SUCCESS	FAILURE
Sequence		If one child returns RUNNING	If all children return SUCCESS	If one child returns FAILURE
Fallback		If one child returns RUNNING	If one child return SUCCESS	If all child returns FAILURE
Parallel		Any other situation	If $\geq N$ children return SUCCESS	If $> n - N$ return FAILURE
Decorator		Custom	Custom	Custom
Action		During completion	On conclusion	If unfeasible
Condition		Never	If true	If false

Table 1. Most frequently encountered node types in a Behavior Tree, outlining their main characteristics

2.1.2. Blackboard

In a behavior tree, leaf nodes interact with the environment through a set of variables known as the blackboard. The blackboard mechanism facilitates the integration of events by allocating memory for specific tasks or by translating events into lower-level mission plan execution status. These statuses can be stored in the blackboard and reset after processing appropriate behavior tree tasks. (Agis et al., 2020)

In the swarm robotics implementation seen in this research, the leaf nodes within the behavior tree interact with the external environment through the use of a blackboard. Each robot in the swarm has its own version of the blackboard, where all the necessary data for the robot is stored in various types. Each entry on the blackboard has an access level, which determines how the behavior tree can interact with it.

In the shapes project (Cooper & Lemaignan, 2022), we can see a basic example of the use of blackboard when the robot outputs a string with

the name of the identified person after recognizing it.

Blackboards are one of the key elements in the real implementation of behavior trees; it is possible to see it for example in (Jones, 2020) in the explanation of the controller of its system "A behavior tree consists of a tree of nodes and a blackboard of variables which comprise the interface between the controller and the robot. At every controller update cycle, the tree of each robot is evaluated, with sensory inputs resulting in actuation outputs. Evaluation consists of a depth-first traversal of the tree until certain conditions are met. Each agent has its own blackboard, state memory and tree."

2.2 BTs vs. FSM

FSMs have been for long time the standard choice for constructing structures for task switching, in 2013, Klockner highlighted the current prevalent use of Finite State Machines (FSMs) as a means of

managing the diverse capabilities of an Unmanned Aerial Vehicle (UAV), quoting in his paper "The state-of-the-art approach for managing different capabilities of a UAV is to use Finite State Machines (FSMs)." (Klockner, 2013)

In a comprehensive manner, FSMs are a computation model that can be in only one state at any one time. The state changes when specific conditions are met, which are usually linked to sensor inputs. The outputs of actuators are usually dependent on the current state.

Despite the multitude of possibilities that finite state machines offer, it has been noted that they encounter difficulties in satisfying two essential characteristics that are often sought-after in autonomous agents: the ability to exhibit both reactive and modular capabilities.

The term reactive refers to the ability to rapidly and effectively respond to changing circumstances. This can manifest in various ways as for example shown in the demo developed in this paper, when a robot reroutes to avoid a collision. The modularity allows for a more hierarchical and adaptable approach in designing, which can lead to more flexibility and ease of modification of the agent's behavior.

As explained in (Colledanchise & Ögren, 2017), the problem lies in FSM's trade-off between reactivity and modularity. This compromise can be comprehended by referencing the traditional Goto statement, heavily criticized by Edsger Dijkstra (Dijkstra, 1968), that was utilized in early programming languages. The Goto statement serves as an exemplar of a one-way control transfer, in which the execution of a program diverts to another section of code and continues from that point. In contrast to one-way control transfers, contemporary programming languages tend to favor two-way control transfers, as exhibited in constructs such as function calls. In order for the

system to be reactive, many transitions between components are necessary. However, a high frequency of transitions results in an abundance of one-way control transfers. For instance, if a component is removed, every transition to that component must be updated, affecting lastly the modularity. BTs, on the contrary, utilize two-way control transfers regulated by the internal nodes of the tree, which has been shown to mitigate this trade-off.

As the number of capabilities or inputs that a finite state machine is required to process increases, the complexity of the FSM also increases in terms of the number of states and transitions. This results in a scalability issue, as the FSM becomes increasingly difficult to design, analyze, understand, maintain, and verify as the number of states and transitions increases, making it challenging to ensure that the FSM is working as intended.

It is also worth noting that behavior trees offer advantages over finite state machines in terms of reusability. The hierarchical structure of behavior trees enables the reuse of subtrees, allowing for more efficient creation of new behaviors or the extension of existing ones. This modular design allows for a more streamlined development process, as it enables developers to easily add or modify behaviors without having to make extensive changes to the overall structure of the system.

It can be highlighted that while behavior trees provide an advantage over finite state machines, it has been demonstrated through the research of Colledanchise [2017] and Jones [2020] that both BTs and FSMs are mathematically equivalent in terms of their capabilities.

The validity of these advantages has been substantiated through various contemporary studies, including (Cooper & Lemaignan, 2022) (Tadewos et al., 2019), and more insights on the

topic will be examined in greater detail within section [2.1].

2.1 State of the art

Even though BTs have been around for more than a decade now, not too many years ago the complaint was that in the field of Behavior Trees "The available literature lacks the consistency and mathematical rigor required for robotic and control applications" (Marzinotto et al., 2014). Since then, attempts were made to standardize the basis of behavior trees, as seen in (Champandard & Dunstan, 2019). The purpose of this review is to provide a comprehensive overview of the current state of the art in the application of behavior trees in swarm robotics focusing on three recent researches.

In this research conducted in 2021 by Legarda.H studied a methodology for the collective transport of heavy objects through the utilization of an industrial swarm in a simulated setting is presented. The method was developed and tested in a simulated environment, using a combination of freely available open-source libraries. The robots employed controllers that incorporate a behavior tree architecture, which were optimized using genetic programming (GP) techniques to generate actuator commands. Coordination was achieved through a decentralized negotiation strategy that employed direct communication. The findings indicated that the genetic programming algorithm is capable of generating controllers that are able to safely lift and convey multiple loads concurrently towards a designated area, referred in the project as nest region. However, further refinement would be needed to implement these advancements in a real-world environment in order to assess its sensitivity to the reality gap.

The first research (Jones, 2020), linked as well with another earlier publication (S. Jones et al., 2018), puts the focus on the issues of designing optimal and readable systems. In order to design swarm robot systems with desired collective behavior, the challenge lies in designing controllers for individual robots that will produce the desired communal behavior through interaction. The current solutions studied for the mentioned research often relied on offline automatic discovery using artificial evolution of robot controllers, which are then transferred to the swarm, but that approach had limitations such as the need for additional supporting infrastructure for evolution and communication and also the evolved controllers being opaque and difficult to understand which could compromise safety and explainability. This research addresses both these issues by using behavior trees. Behavior trees were used as the individual robot controller architecture with great results, focusing on them being modular, hierarchical, and human-readable. Automatic tools were developed to simplify large evolved trees, making it possible to understand, explain, and improve the evolved controllers.

The research highlights three main topics to research in the future; firstly, adaptability represents a crucial aspect to be considered, for the system described to possess the ability to better adapt its behavior in response to changes in the environment. Another important area of investigation is the reality gap and the impact of the chosen architecture. Additionally, expanding the system to operate in three-dimensional environments is also worth exploring, since the research was made in a two-dimensional setting. The videos of the implementation of this research can be consulted¹

¹ youtube.com/user/simonj23/videos

The third study (Ligot et al., 2020) examines the potential, challenges, and limitations of using behavior trees in the automatic modular design of collective behaviors in swarm robotics. An automatic design method called Maple is introduced, which combines predefined modules such as low-level behaviors and conditions into a behavior tree that encodes the individual behavior of each robot in the swarm. In the research is stated that in comparison to FSMs, BTs offer a number of appealing features. However, the majority of these characteristics are only realized if the action nodes return their states of execution, meaning that the robot is capable of determining whether the low-level behavior it is executing has been successfully terminated, cannot execute normally, or requires further time to conclude. In the context of swarm robotics, the simple and reactive robots that are commonly employed lack the ability to determine the state of execution of the low-level behaviors they operate. A comparison of Maple with Chocolate, a previously proposed automatic design method utilizing probabilistic finite state machines, is conducted to examine the strengths and weaknesses of using behavior trees as a control architecture. The first study evaluates Maple's ability to produce control software that successfully crosses the reality gap. These findings support the conjecture of Francesca et al. (2014) that the automatic modular design (AutoMoDe) approach is robust to the reality gap due to its modular nature. The second study examines Maple's performance in relation to the design budget, or the maximum number of simulations runs allowed in the design process. The third study explores various variants of Maple with varying constraints on the structure of the behavior trees generated. The results of these studies suggest that while behavior trees may be appealing in the context of swarm robotics, they do not consistently produce superior solutions compared to finite state machines, further research needs to be made. Future research could focus on two main areas, one

being the further examination of the use of low-level behaviors as action nodes of behavior trees, including evaluating the control software generated by Maple with different design budgets in robot experiments and exploring further variants by relaxing the restrictions on the number of levels, branches, and leaves. The second area of focus could be the development of a tailored optimization algorithm that leverages the inherent modularity of behavior trees, using local search algorithms such as iterative improvement and simulated annealing as starting points.

3 Experiments

The demonstration presented in this section aims to provide a high-level overview of the logic utilized by an autonomous robot within a swarm robotics environment.

It is important to note that the demonstration created is intended to serve as a structure that illustrates the implementation of behavior trees in swarm robotics, so several assumptions about the physical embodiment of the system model had to be made. Even though the level of the battery of the robots is taken into account, the focus is solely on the development of software capabilities, so hardware constraints such as environment mapping through techniques such as SLAM or position monitoring through GNSS and IMU sensors are outside the scope of this paper, as due to scalability issues. For these reasons, it is assumed that the environment is already mapped and known to each agent, and they know the position of transfer stations and pickup stations. Moreover, it is assumed that each agent is a holonomic robot when modeling the motion in the warehouse. Similarly, the part related to the Environment sensors and actuators will not be analyzed. Conversely, in this paper, the Autonomous

capabilities are strongly emphasized, and the proposed solutions mainly focus on this domain, such as the ability to independently or cooperatively with another agent execute single tasks. The robots are able to interact and have intra-cooperation and networking capabilities.

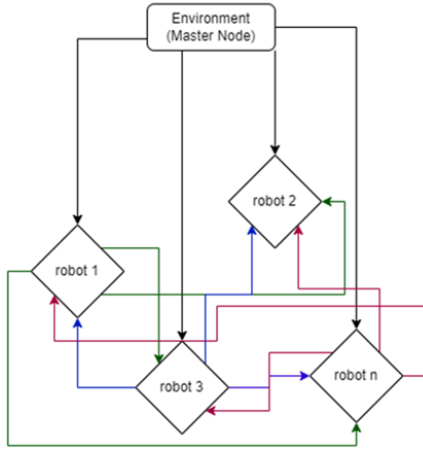


Figure 1. Schema of the proposed communication connections in the demo

The present study demonstrates the implementation of a system in which the environment and autonomous robots engage in mutual communication, as depicted in Figure 1. This experiment can be found in the GitHub repository associated with this research. Specifically, the experiment is set in a warehouse context, wherein the facility is divided into two distinct sections. As a result, the robots on one side are unable to traverse to the other side to perform package deliveries. To mitigate this limitation, the robots on one side collaborate with their counterparts on the other side through transfer stations to achieve efficient delivery of goods. In this demonstration, the environment disseminates directives to all robots situated on one side of the warehouse, which comprise of order requirements and the relevant information pertaining to the selected pickup station. Subsequently, each robot

assesses the availability of orders and determines the pickup station to which it should proceed. If a robot selects an order, it subsequently communicates this information to the other robots within the system, thereby enabling collaborative task execution. In this study that the blackboard mechanism is utilized as a means of illustrating the capability of the robot to acquire orders from various pickup points on an ongoing basis.

Upon determining the location of the pickup station, the robot proceeds to calculate a route, which can be adjusted in the event of encountering obstacles. Once the robot selects a delivery station, it broadcasts this information to its paired robot on the opposite side of the warehouse, in a manner similar to the communication protocol described in Legarda Herranz (2021). Other examples of this type of communication can be found in the literature, such as in the work of Klockner (2013).

After determining the transfer station, the robot calculates a route, which, again, can be modified as necessary. Simultaneously, the collaborative robot proceeds to the transfer station to retrieve the order from the opposite side. Upon completion of the delivery, the initial robot marks itself as available, thereby enabling potential interactions with humans or the environment for maintenance or other purposes.

3.1 Implementation & results

In this study, the C++ implementation of BTs employs the BehaviorTree.CPP library developed by Faconti and Colledanchise. It is worth noting that the compatibility of Groot 1.0 is exclusively with BehaviorTree.CPP version 3.8.x, thus the demonstration presented in this paper, which is available in the GitHub repository², is constructed

² <https://github.com/zaidabri/BehaviorTreeSwarmRobotics>

upon the framework provided by BehaviorTree.CPP 3.8.x.

It is significant to point out characteristics of the library under examination. In addition to being compliant with Groot, it also exhibits compatibility with both ROS1 and ROS2, enabling the seamless deployment of controllers constructed using this library on the corresponding physical platform. Furthermore, the behavior trees are represented in XML, which enhances the readability and comprehension of the trees. Additionally, the library provides a diverse array of pre-built common nodes that can be utilized during the development process, further expanding the functionality of the library.

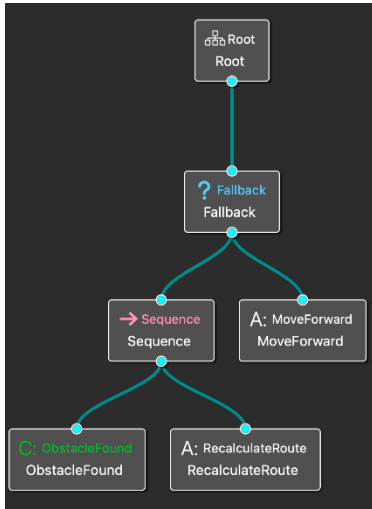


Figure 2. Obstacle avoidance subtree

The implemented logic was developed based on the principles outlined in section 3 of Colledanchise and Ögren (2017). Graphical illustration implemented in Groot is depicted in Figure 3. Behavior Tree implementation

As previously discussed in the preceding section, the limitations of the physical model entail that the signals received and transmitted by the robot are currently hardcoded. However, in a real-world implementation, it would be imperative to

incorporate the updating of the blackboard with data obtained from the sensors and the overall system in order to enhance the functionality of the system.

At each iteration, the robot evaluates the current battery level. In the event that the fallback node experiences a failure, the robot initiates the action to proceed to the charging station. When the battery tree returns a success status, the robot queries the environment to determine the availability of orders. In the event that an order is available, the robot retrieves from the environment through the blackboard the designated pickup station and subsequently sends a signal to other robots, through the blackboard, indicating the reservation of the order. The robot is programmed to track down the pickup station visually once it receives its location. Thinking about the case in which the pickup station is not within an accessible visual range, the robot is given five attempts. Upon the identification of the pickup station, the obstacle subtree, as depicted in Figure 2. Obstacle avoidance subtree, is activated in order to conduct an obstacle detection process. In the event that obstacles are detected, the robot engages in the recalculation of its route. Conversely, if no obstacles are present, the robot proceeds forward without deviation. After picking up the package, the robot engages in the process of identifying a transfer station. Upon successful identification, an output signal is transmitted and recorded on the blackboard, enabling other robots to detect its location and initiate movement towards the transfer station.

Subsequently, the obstacle subtree is reactivated to detect any potential obstacles in the path. Once the route is calculated, the robot proceeds to move forward and deliver the order. Finally, the robot sends a signal to the system indicating that it is available for further tasks, which could be useful for human supervision or maintenance purposes.

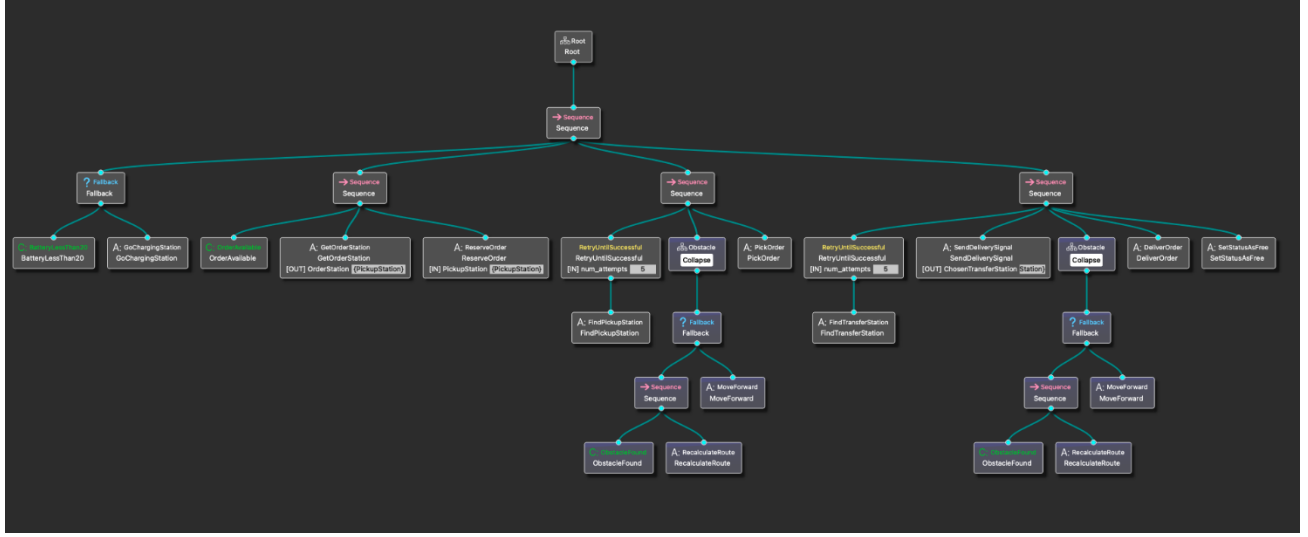


Figure 3. Behavior Tree implementation

4 Conclusions

This paper has explored the implementation of Behavior Trees in the field of Swarm Robotics. The research began by providing a comprehensive overview of the theoretical foundations of BTs and their role in the design and control of robotics systems, including a review of the current state of the art and key challenges and limitations identified in previous research. The paper then discussed the advantages and disadvantages of BTs over other traditional systems like Finite State Machines (FSMs) and finally presented results obtained from an example code implementation developed for this paper. Through this analysis, the research aimed to provide a clear understanding of the key considerations and issues that must be taken into account when using BTs in swarm robotics and the implementation possibilities that BT offers.

The results of this study demonstrate that BTs offer a viable and powerful approach to swarm robotics due to their flexibility, scalability, and readability. This simulation scenario of swarm

robotics using behavior trees has been developed in an open-source library in C++ with great results.

The research question was "To what extent does the use of Behavior Trees improve the coordination and control of swarm robotics systems?" and the conclusion is that behavior trees can serve as a powerful tool in controlling the decision-making of a swarm of robots, enabling them to coordinate and accomplish a given task with efficiency. Furthermore, the use of behavior trees in swarm robotics offers several advantages, such as the ability to clearly represent complex logic and decision-making processes, and the capability to adapt to changing conditions in the environment. The ideal next steps in the development of the method proposed would be to test it in a real-world environment in order to better evaluate its sensitivity to the reality gap.

5 References

- [1] (Agis et al., 2020)
Agis, R. A., Gottifredi, S., & García, A. J. (2020). An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, 155(113457), 113457. <https://doi.org/10.1016/j.eswa.2020.113457>
- [2] (Champanand & Dunstan, 2019)
Champanand, A. J., & Dunstan, P. (2019). The behavior tree starter kit. In *Game AI Pro 360* (pp. 27–46). CRC Press.
- [3] (Colledanchise, 2017)
Colledanchise, M. (2017). Behavior trees in robotics. *Diva-portal.org*. Retrieved January 9, 2023, from <https://www.diva-portal.org/smash/get/diva2:1078940/FULLTEXT01.pdf>
- [4] (Colledanchise & Ögren, 2017)
Colledanchise, M., & Ögren, P. (2017). Behavior trees in Robotics and AI: An introduction. *ArXiv [Cs.RO]*. <https://doi.org/10.48550/ARXIV.1709.00084>
- [5] (Cooper & Lemaignan, 2022)
Cooper, S., & Lemaignan, S. (2022). Towards using behaviour trees for long-term social robot behaviour. 2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 737–741.
- [6] (Dijkstra, 1968)
Dijkstra, E. W. (1968). Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3), 147–148. <https://doi.org/10.1145/362929.362947>
- [7] (Dorigo et al., 2013)
Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Gonzales, J. M., ... Vaussard, F. (2013). Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4), 60–71. <https://doi.org/10.1109/mra.2013.2252996>
- [8] (Dromey, 2001)
R.G.Dromey, (2001). Genetic Software Engineering – Simplifying Design Using Requirements Integration, IEEE Working Conference on Complex and Dynamic Systems Architecture, Brisbane.
- [9] (Faconti, 2022)
Davide Faconti. 2022. BehaviorTree.CPP library Documentation. <https://www.behaviortree.dev>
- [10] (Faconti, 2019)
Faconti, D. Groot. <https://github.com/BehaviorTree/Groot>. 2019.
- [11] (Francesca et al., 2014)
Francesca G, Brambilla M, Brutschy A, Trianni V, Birattari M. 2014. AutoMoDe: a novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence* 8(2):89–112 DOI 10.1007/s11721-014-0092-4.
- [12] (Ghzouli et al., 2020)
Ghzouli, R., Berger, T., Johnsen, E. B., Dragule, S., & Wąsowski, A. (2020). Behavior Trees in action: A study of robotics applications. *ArXiv [Cs.RO]*. <https://doi.org/10.48550/ARXIV.2010.06256>
- [13] (Heckel et al., 2010)
Heckel, F. W. P., Youngblood, G. M., & Ketkar, N. S. (2010). Representational complexity of reactive agents. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*.
- [14] (Isla, 2005)
Isla, D. (2005, March 11). GDC 2005 proceeding: Handling complexity in the Halo 2 AI. *Game Developer*. <https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai>
- [15] (Isla, 2008)
Isla, D (2008). "Building a Better Battle: Halo 3 AI Objectives". In: *Game Developers Conference*. 2008.
- [16] (S. Jones et al., 2018)
Jones, S., Studley, M., Hauert, S., & Winfield, A. (2018). Evolving behaviour trees for swarm robotics. In *Distributed Autonomous Robotic Systems* (pp. 487–501). Springer International Publishing.
- [17] (Jones, 2020)
Jones, S. W. (2020). Onboard evolution of human-understandable behaviour trees for robot swarms. University of Bristol. Retrieved January 9, 2023, from <https://research-information.bris.ac.uk/en/studentTheses/onboard-evolution-of-human-understandable-behaviour-trees-for-rob>
- [18] (Klockner, 2013)
Klockner, A. (2013). Behavior Trees for UAV Mission Management. *Dlr.de*. Retrieved January 9, 2023, from <https://elib.dlr.de/91679/1/kloeckner2013behavior.pdf>
- [19] (Kuckling et al., 2021)

Kuckling, J., van Pelt, V., & Birattari, M. (2021). Automatic modular design of behavior trees for robot swarms with communication capabilities. In *Applications of Evolutionary Computation* (pp. 130–145). Springer International Publishing.

[20] (Legarda Herranz, 2021)

Legarda Herranz, G. (2021). Evolution of behaviour trees for collective transport with robot swarms. Universitat Politècnica de Catalunya.

[21] (Ligot et al., 2020)

Ligot, A., Kuckling, J., Bozhinoski, D., & Birattari, M. (2020). Automatic modular design of robot swarms using behavior trees as a control architecture. *PeerJ. Computer*

Science, 6(e314), e314. <https://doi.org/10.7717/peerj-cs.314>

[22] (Marzinotto et al., 2014)

Marzinotto, A., Colledanchise, M., Smith, C., & Ogren, P. (2014). Towards a unified behavior trees framework for robot control. 2014 IEEE International Conference on Robotics and Automation (ICRA), 5420–5427.

[23] (Tadewos et al., 2019)

Tadewos, T. G., Shamgah, L., & Karimodini, A. (2019). Automatic safe behaviour tree synthesis for autonomous agents. 2019 IEEE 58th Conference on Decision and Control (CDC), 2776–2781.