Behavior Tree as a Decision Planning Algorithm for Industrial Robot

Martina Hutter-Mironovova[1], Benjamin Blumhofer[2], Christopher Schneider[1], Achim Wagner[3]
[1]YASKAWA Europe GmbH, Yaskawastraße 1, 85391 Allershausen, Germany
martina.hutter@yaskawa.eu, christopher.schneider@yaskawa.eu
[2]Technologie-Initiative SmartFactory KL e. V., Trippstadter Str. 122, 67663 Kaiserslautern, Germany
benjamin.blumhofer@smartfactory.de
[3]German Research Center for Artificial Intelligence GmbH (DFKI), Trippstadter Str. 122, 67663 Kaiserslautern, Germany
achim.wagner@dfki.de

Abstract. In traditional industrial settings, robot execution planning is typically governed by pre-programmed instructions, with cell logic predominantly managed by PLC (Programmable Logic Controller) systems. However, the rapid advancements in artificial intelligence have unlocked new possibilities, enabling the deployment of robots in previously unautomated sectors. Enhanced machine vision, advanced data processing, and increased adaptability to dynamic environments are now within reach. These developments necessitate a re-evaluation of conventional approaches to robot and cell programming. This paper explores the implementation of behavior trees as an alternative execution planning algorithm, specifically applied to an industrial YASKAWA robot, demonstrating their potential to optimize performance and flexibility in complex industrial applications in dynamic environment.

1 Introduction

In a typical industrial application, robot's moves and interaction with other devices are executed by the robot program. Execution of the cell functionality is typically driven by a PLC control, either separated or integrated within the robot controller. This approach is based on the principle that the robot consistently performs the same movements and actions to meet a predefined cycle time and maintain an optimal workflow as defined by the programmer. However, it does not account for changes in the working process, variations in operating conditions, environmental shifts, or the need for flexibility within the work cell [1].

With decreasing amount of available human workforce and increasing demands of manufacturers, also areas of manufacturing or production, that were never considered for automation, are recently considered for deployment of robots. In comparison to fully automated solution, robots offer flexibility due to multi degrees of freedom and can be used in production, where conditions are changing. However, until now, such automation has been challenging to achieve and has proven costly to implement and maintain. Adding new products and functionality required an expert in robot programming or machine vision and new learning for operators.

The recent advancements in artificial intelligence have enabled the deployment of robots in industries that have historically depended on human labor due to the necessity of cognitive abilities and creativity. These sectors include food processing and bakery operations, construction sites, healthcare facilities, biomedical and chemical industries, waste management systems, and other domains where automation was previously considered infeasible. Thanks to new development in artificial intelligence (AI), machine vision and data processing can be enhanced, and robots can be used in dynamic environments with adaptive capabilities. These use cases frequently demand more adaptive robot behavior, making it essential to explore new execution planning approaches for robots and cells to manage the increased complexity. Behavior trees offer modularity and scalability as well as clearer decision making with clear visualization. Such advantages offer a different approach in execution planning compared to traditional procedural programming approaches, where varying situations are typically solved using conditions and loops, which make system much more difficult to scale up.

This paper presents the implementation of behavior trees on an industrial YASKAWA robot from the NEXT generation and shows advantages of used execution planning algorithms.

2 State of the Art

Execution planning can be implemented through three primary approaches: traditional routine programming on a CPU, enhanced programming with an external PC or GPU, and programming utilizing IoT and big data technologies [2].

With a classical programming, robot job runs directly in the robot controller. This programming is typically used in a predefined environment with known robot paths and positions, which robot needs to reach. It does not involve changing environment or need for machine vision. These programs are easy to implement and debug, typically being written in a language specific to the manufacturer.

On the other hand, when the environment around the robot is changing or the robot task is varying, it is necessary to use additional sensors to achieve required performance of the system. Such sensors include cameras, lidars, radars, tactile sensors and force and torque sensors as most common ones. Usage of those devices enlarges the capabilities of the robot and if are combined with methods of artificial intelligence, they can level up the device in its performance [3]. Methods of classical machine vision are boosted up with use of classification or other forms of machine learning algorithms [4]. However, such AI algorithms require more computation power and therefore, CPU of the robot controller is no longer capable of performing such calculations in reasonable time. Usually, such algorithms would need to run on an additional machine (industrial PC) connected to the robot controller. Adding another computer adds complexity to the hardware architecture and complicates robustness of the system.

Third method mentioned above relates usage of big data, such as databases of learned models or patterns. The data is stored in a cloud, which brings not only networking delays, but also feared loss of data or cyber security related issues.

Intelligent robots are supposed to work in less-structured environments together in collaboration with humans. Such workflows or tasks are composed of sub-tasks, that can be executed independently [5]. Example would be a grasping skill of the robot, based on the object and its position. Behavior trees simplify the composition of these sub-tasks in the whole application and the order in which the sub-tasks are executed is independent from their implementation, as sub-tasks can be designed, tested, and replaced independently. Such approach allows an easy composition of trees and creation of larger trees. In traditional programming, behaviors can be created as sequences of function calls, making it more difficult to implement, reuse, or maintain individual components, which can introduce bugs, make debugging more difficult and make the system less predictable [5]. Another huge benefit of behavior tree is parallelism, where behaviors can run simultaneously within the node. Implementation of parallelism in procedural programming often requires explicit threading or concurrent operations, which adds complexity and potential synchronization issues. Implementation of behavior trees also brings ease of use for non-skilled programmers. Robots can be deployed faster, therefore lowering costs for the companies [1].Behavior tree should be designed in the way, that is utilizing modularity of the sub-tasks as much as possible. Behavior tree must fulfil all requirements, all nodes shall be executed in correct workflow and all behaviors must be covered. In case of very complex trees, this could be more difficult to perform [5]. More standardized approach to understanding of robot skills can simplify the implementation of the behavior tree and interconnectivity of individual nodes [6]. Each skill should be designed that it receives and provides information in a structured way for universal connection to other skills. Another aspect is to achieve adaptable movement of the robot, so it would simply move between positions without collisions with its environment. Working area of the robot can be monitored (with radar, lidar, camera, etc.) and collected data transformed into virtual environment. In such digital twin, robot paths can be automatically generated based on the obstacles or other equipment in the workcell [6].

For practical use, automatic generation of the BT architecture from classical workflow description would be enormous simplification for project development and can be considered as a further work [7]. Research in this field have been performed with promising results, especially with use of reinforcement learning methods [8], simulation and reality [9].

3 Approach to Execution Planning with Behavior Trees

Behavior trees started to be used in computer game industry for modelling of NPC characters' behavior (Non-player character). In the last few years, usage of this programming method spread also to robotics area as a more expressive tool to model behavior of autonomous agents [10].

Behavior tree (BT) is designed as a composition of behaviors, that are independent from each other. Execution (tick) of a BT starts from the root (node without a parent) and propagates through top to bottom and left to right. Execution ends at leaf, which has no other child nodes underneath. The rules are set how behaviors occur, and in which order they will be executed. BT follows traditional way of data structure, where execution starts at the root in fixed direction, so it does not look back or repeat itself.

Internal nodes are called control flow nodes and leaf nodes are called execution nodes. The control flow node must have at least one child and each node has one parent. Tick is a trigger to execute the node, which returns either running, failure or success. Control flow nodes are of four types: Sequence, Fallback, Parallel, Decorator and execution nodes are either Condition or Action [10]. Various node types are explained in the Table 1.

BT used in this example is BT.CPP, a C++17 library, that provides a framework for construction of a behavior tree. It is written in C++ language and tree can be defined using a scripting language based on XML. Created behavior tree can be visualized and edited via GUI called Groot 2 [11]. There are variety of different libraries in C++ or Python languages available for the user. List of those can be found in this publication [12] with description and comparison analysis.

Table 1. Explanation of nodes with labelling convention. [10]

Node	Labelling	Success	Failure
Sequence (AND)	→	All children must return success	At least one child return failure
Fallback (OR)	?	One child returns success	All children return failure
Parallel	⇉	When at least M child nodes succeed	When all child nodes fail
Decorator	δ	According to user defined policy	According to user defined policy
Condition	Condition	If is true	If is false
Action	Action	When completed	If not possible to complete

3.1 Hardware and Software Architecture

For the implementation purposes, new generation of intelligent robots MOTOMAN NEXT of YASKAWA is used. In addition to the robot controller CPU this robot is expanded with an integrated edge computer equipped with CPU and GPU. CPU of the robot controller takes care of standard robot functionalities, while GPU allows user to run expanded functions, such as robot control service, AI service, machine vision service, path planning and obstacle avoidance service and user defined skills, tasks or services. Architecture of the NEXT controller is illustrated in Figure 1.

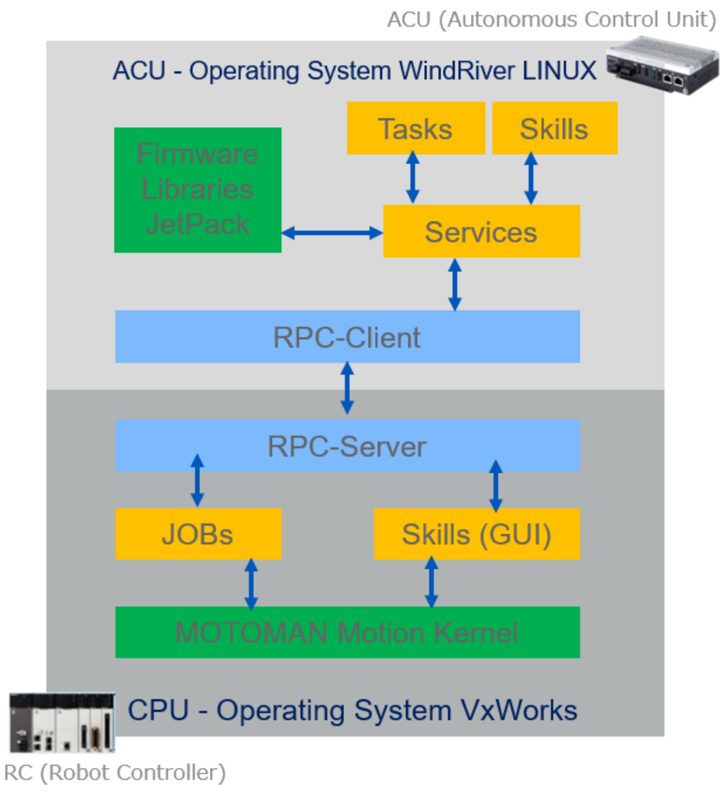


Fig. 1. Architecture of the NEXT controller. RC (Robot Controller) operating system is VxWorks, it hosts the motion kernel, which is core functionality for the robot functionality. On RC run JOBs (robot native programs) and skills user interfaces. RPC-Server serves for transferring tasks between CPU and ACU.

With this unique and novel architecture, it is easy to achieve optimal performance and develop custom applications, that can be deployed directly inside the robot controller without the need of additional PC. Services offer APIs for easy and direct implementation of functions into the user's own code, which can be then uploaded onto the ACU (Autonomous control unit) as a containerized application (Docker container) via graphical user interface.

3.2 Description of the Robot Task

The demonstration of the pick skill designed by behavior tree is shown in Figure 2. A Machine vision service provides the product type and 3D pose of the robot based on the recognized product type and its position in the cameras coordinate system. 3D pose is directly converted and written into the position variable in the RC as three positions and three rotations in robot coordinate system. Product type is written in variable in RC as well. Recognized products, their type and position are written in a table that is checked by the tick in decision tree. Based on the type of the product written in the table on i-th position during the i-th iteration, the gripping method is selected. Either a finger gripper or a vacuum suction cup gripper is used. This is simplified Pick Skill without error handling. Behavior tree can be either visualized in the graph form (Figure 2) or written in XML format.

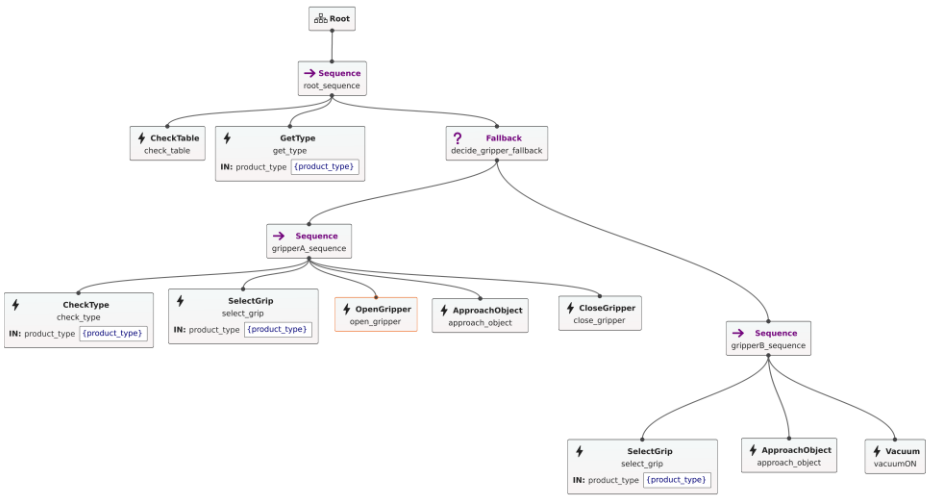


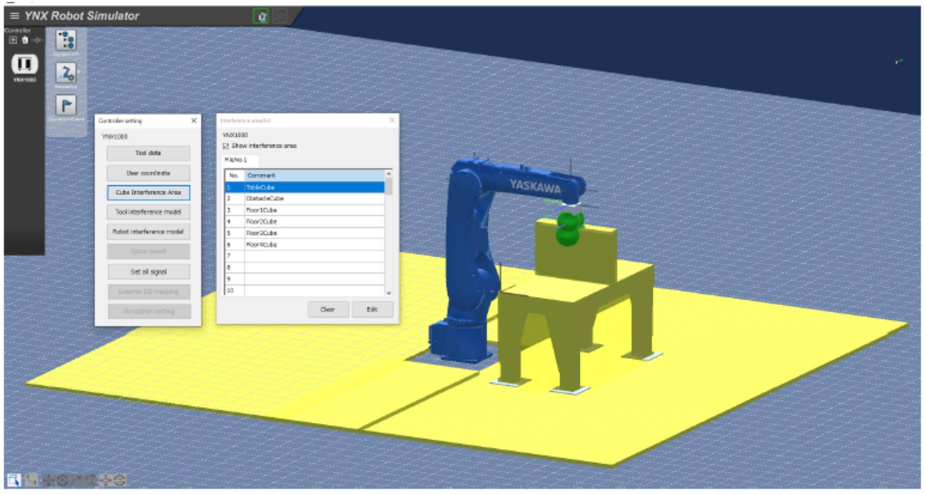
Fig. 2. Structure of the behavior tree. Individual nodes are represented as rectangles, arrows connect child nodes and parent nodes.

Behavior tree practically describes the workflow and actions performed by the robot. In its visual form, it is easy to understand and follow the job flow, as well as debug the relations between nodes. When the BT is created, it is necessary to pay attention to correct workflow and execution, because complex tasks can lead to complex structures of behavior trees. In such complexity, it might be difficult to cover all possible situations or behaviors [5]. However, one advantage of behavior trees is their ability to nest multiple trees within a single structure.

4 Implementation into the MOTOMAN NEXT

In the previous section, the structure of the behavior tree was established and executed in XML format (or visualized by Groot 2 software). In this section, an implementation of the behavior trees to the robot controller will be discussed.

Typically, behavior trees use the framework ROS (Robot Operating System) as the most common open source software development kit among companies or institutions in research and industry in robotics and automation providing an interface between the applications and the hardware. It is modular and reusable, allowing developers to create independent programs (called nodes), that can communicate with each other and be therefore re-used or shared freely among large community.



No.	Comment
1	TableCube
2	ObstacleCube
3	Floor1Cube
4	Floor2Cube
5	Floor3Cube
6	Floor4Cube
7	
8	
9	
10	

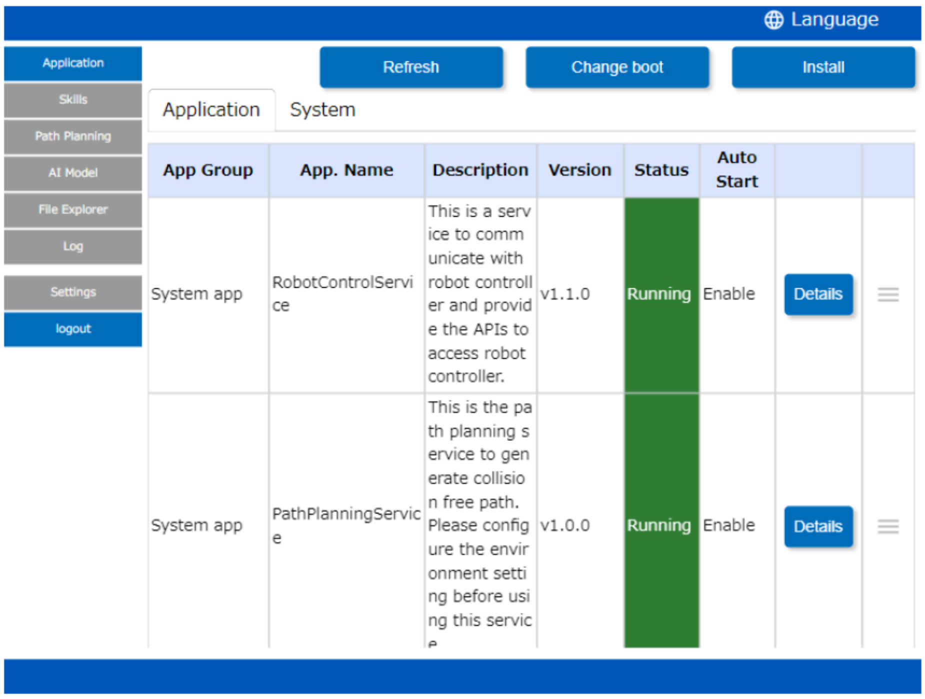
Fig. 3. Digital twin in the simulation environment YNX Robot Simulator and the surrounding environment with defined obstacles.

The NEXT controller from YASKAWA provides a comprehensive set of services (APIs) that offer a direct interface to the robot controller, eliminating the need for additional interface setup. This enables users to implement a library of functions and seamlessly exchange data between their applications and the robot controller, as well as across multiple applications. Users can manage variables (such as byte, integer, string, positions, etc.), initiate robot motions (linear, joint), execute robot JOBs (from the RC controller), and read or write system variables (such as servo on, start, hold, etc.). These services are fully compatible with one another, allowing for integrated functionality; for instance, the Machine Vision service can provide position data of detected objects, which the Robot Control service can process and send to the RC unit. Simultaneously, the Path Planning service creates an optimal trajectory for the robot between two points in space, taking into account environmental factors like obstacles, other machines, and parts. The robot follows this automatically generated collision-free path, and the path can be recalculated with each iteration of the program, enabling the robot to adapt to changing conditions in its environment (Figure 3).

An example of such implementation of functions, that write to the robot controller variables (byte and position) and execute the robot motion are shown in the pseudo code. It shows one node implementation (this node executes a motion of the robot), creation of the tree from XML file and node registration to the tree structure. Every node has to be written separately as a function, that implements the node behavior. I.e., opening the gripper would set the variable that would trigger the external output (24V signal) to the gripper.

```
// Import used libraries incl. APIs from YASKAWA
Import ...

// Define individual nodes and their functionality
class PickFruitBehaviourNode extends BehaviourNode {
...
virtual BehaviourNodeStatus execute() override {
  //YASKAWA functions for moving the robot,
  //writing variables, starting JOBs, etc.
  robot_controller_client_.robot_motion(...);
  robot_controller_client_.variable_byte_value_set(...);
  robot_controller_client_.job_start(...);
  return BehaviourNodeStatus();
}

// define the client for the communication with RC unit
RobotControllerClient robot_controller_client_;

}

function void main() {
// start the client for the communication with RC unit
robot_controller_client = new RobotControllerClient(...);
// generate BT from XML file
behaviour_tree = new BehaviourTree(...);
// register all nodes to BT
behaviour_tree.registerNode<PickFruitBehaviourNode>(robot
_controller_client);
behaviour_tree.execute();
}
```

The user application must be containerized and build within the ACU SDK environment. Once completed, the application can be uploaded and managed through the web-based interface of the ACU (Figure 4).



App Group	App. Name	Description	Version	Status	Auto Start		
System app	RobotControlService	This is a service to communicate with robot controller and provide the APIs to access robot controller.	v1.1.0	Running	Enable	Details	≡
System app	PathPlanningService	This is the path planning service to generate collision free path. Please configure the environment setting before using this service	v1.0.0	Running	Enable	Details	≡

Fig. 4. Graphical user interface in the ACU. Left hand side menu shows Services, Skills and Applications in the ACU, and logging and settings interfaces. Main screen displays list of all uploaded applications, their name, description, version number and status.

5 Conclusions and Further Work

This study demonstrates the effectiveness of integrating behavior trees with industrial robots, offering a flexible and adaptable approach to execution planning in dynamic manufacturing environments. Behavior trees provide a structured and intuitive method for managing complex tasks, and when combined with AI-driven machine vision, they significantly enhance the capabilities of industrial robots. The innovative architecture of the MOTOMAN NEXT robot, with its built-in ACU unit, facilitates the direct implementation of AI methods within the robot controller, eliminating the need for additional external computing resources. The APIs provided by YASKAWA further streamline the integration of native robot skills and services into custom user applications.

The current implementation serves as a foundation for further development, with plans to incorporate additional YASKAWA services, such as Machine Vision and Path Planning, to create a fully operational application. This will allow for a comprehensive evaluation of performance in real-world scenarios. Additionally, the behavior tree framework could be expanded into a user-friendly service, complete with pre-defined nodes for robot motion, gripper control, and other essential functions. This would greatly simplify the process for users to develop their own applications, enhancing the overall user experience and expanding the potential for innovation in industrial robotics.

References

1. Sidorenko, A., Rezapour, M., Wagner, A., Ruskowski, M.: Towards Using Behavior Trees in Industrial Automation Controllers (2024)
2. Naghib, A., Navimipour, N., Hosseinzadeh, M., Sharifi, A.: A comprehensive and systematic literature review on the big data management techniques in the internet of things, Springer, Wireless Networks (2022)
3. Li, Q., Wu, C., Yuan, Y., You, Y.: MSSP : A Versatile Multi-Scenario Adaptable Intelligent Robot Simulation Platform Based on LIDAR-Inertial Fusion (2024)
4. Peters, J., Tedrake, R., Roy, N., Morimoto, J.: Robot Learning (2017)
5. Colledanchise, M.: Behavior Trees in Robotics, Doctoral Thesis Stockholm, Sweden (2017)
6. Herrero, H., Moughlbay, A., Outon, J., Salle, D., Ipina, K.: Skill Based Robot Programming: Assembly, Vision and Workspace Monitoring Skill Interaction, Neurocomputing (2017)
7. Iovino, M., Smith, Ch.: Behavior Trees for Robust Task Level Control in Robotic Applications (2023)
8. Banerjee, B.: Autonomous Acquisition of Behavior Trees for Robot Control (2018)
9. French, K., Wu, S., Pan, T., Zhou, Z., Jenkins, O. Ch.: Learning Behavior Trees From Demonstration (2019)
10. Colledanchise, M., Ögren, P: Behavior Trees in Robotics and AI: An Introduction, CRC Press, kTH., (2022)
11. Faconti, D.: BehaviorTree.CPP [Software]. Available online (on 21.08.2024) at GitHub: https://github.com/BehaviorTree/BehaviorTree.CPP (2019)
12. Ghzouli, R., Berger, T., Johnsen, E. B., Wasowski, A., Dragule, S.: Behavior Trees and State Machines in Robotics Applications (2022)