Handling with AI-enhanced Robotic Technologies for flexible ManUfacturing

# D2.4
# HARTU-APP-BUILDER Open Tool

| | |
|---|---|
| **Deliverable ID:** | D2.4 |
| **Project Acronym:** | HARTU |
| **Grant:** | 101092100 |
| **Call:** | HORIZON-CL4-2022-TWIN-TRANSITION-01 |
| **Project Coordinator:** | TEKNIKER |
| **Work Package:** | WP2 |
| **Deliverable Type:** | OTHER |
| **Responsible Partner:** | TEK |
| **Contributors:** | TEK |
| **Edition date:** | 28 June 2024 |
| **Version:** | 03 |
| **Status:** | Final |
| **Classification:** | PU |

This project has received funding from the European Union's Horizon Europe - Research and Innovation program under the grant agreement No 101092100. This report reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

# HARTU Consortium

HARTU "Handling with AI-enhanced Robotic Technologies for flexible manufactUring" (Contract No. 101092100) is a collaborative project within the Horizon Europe – Research and Innovation program (HORIZON-CL4-2022-TWIN-TRANSITION-01-04). The consortium members are:

| | | | |
|---|---|---|---|
| **1** | Tekniker | FUNDACION TEKNIKER (TEK)<br>20600 Gipuzkoa \| Spain | Contact: Iñaki Maurtua<br>inaki.maurtua@tekniker.es |
| **2** | German Research Center for Artificial Intelligence | DEUTSCHES FORSCHUNGSZENTRUM FUE KUENSTLICHE INTELLIGENZ GMBH (DFK<br>67663 Kaiserslautern \| Germany | Contact: Vinzenz Bargsten<br>vinzenz.bargsten@dfki.de |
| **3** | aimen | ASOCIACIÓN DE INVESTIGACIÓN METALÚRGICA DEL NOROESTE (AIMEN)<br>36418 Pontevedra\| Spain | Contact: Jawad Masood<br>jawad.masood@aimen.es |
| **4** | eng | ENGINEERING INGEGNERIA INFORMATICA S.P.A. (ENG)<br>00144 Rome\| Italy | Contact: Riccardo Zanetti<br>riccardo.zanetti@eng.it |
| **5** | TOFAŞ | TOFAS TURK OTOMOBIL FABRIKASI ANONIM SIRKETI (TOFAS)<br>34394 Istanbul \| Turkey | Contact: Nuri Ertekin<br>nuri.ertekin@tofas.com.tr |
| **6** | PHILIPS | PHILIPS CONSUMER LIFESTYLE BV (PCL)<br>5656 AG Eindhoven \| Netherlands | Contact: Erik Koehorst<br>erik.koehorst@philips.com |
| **7** | ULMA | ULMA MANUTENCION S. COOP. (ULMA)<br>20560 Gipuzkoa \| Spain | Contact: Leire Zubia<br>lzubia@ulmahandling.com |
| **8** | deepblue | DEEP BLUE Srl (DBL)<br>00193 ROME \| Italy | Contact: Erica Vannucci<br>erica.vannucci@dblue.it |
| **9** | FMI ImProvia | FMI HTS DRACHTEN B.V. (FMI)<br>NL-4622 RD Bergen Op Zoom, Netherlands | Contact: Floris goet<br>floris.goet@fmi-improvia.com |
| **10** | TECNOALIMENTI | TECNOALIMENTI S.C.p.A (TCA)<br>20124 Milano \| Italy | Contact: Marianna Faraldi<br>m.faraldi@tecnoalimenti.com |
| **11** | | POLITECNICO DI BARI (POLIBA)<br>70126 Bari \| Italy | Contact: Giuseppe Carbone<br>giuseppe.carbone@poliba.it |
| **12** | OMNIGRASP | OMNIGRASP S.r.l. (OMNI)<br>70124 Bari \| Italy | Contact: Vito Cacucciolo<br>vito.cacucciolo@omnigrasp.com |
| **13** | ITRI Industrial Technology Research Institute | INDUSTRIAL TECHNOLOGY RESEARCH INSTITUTE INCORPORATED (ITRI) | Contact: Curtis Kuan<br>curtis.kuan@itri.org.tw |
| **14** | INFAR INFAR INDUSTRIAL CO., LTD. | INFAR INDUSTRIAL Co., Ltd (INFAR)<br>504 Chang-hua County \| Taiwan | Contact: Simon Chen<br>simon@infar.com.tw |



Funded by the European Union

Document history

| Date | Version | Status | Author | Description |
|---|---|---|---|---|
| **02/05/2023** | 01 | Draft | TEK | Document template |
| **12/06/2023** | 02 | Draft | TEK | First version |
| **28/06/2023** | 03 | Draft | TEK | Version submitted |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |



Funded by the European Union

# Executive Summary

This document describes the main concepts of HARTU-APP-MANAGER, the tool that HARTU makes available to end users and system integrators to create and control robotic applications, thus giving rise to two different functionalities: HARTU-APP-CREATOR and HARTU-APP-CONTROL.

The core of HARTU-APP-MANAGER is based on the concept of behaviors trees and its implementation using BehaviorTree.CPP 3.8, a C++ library for building BehaviourTrees. The GUI is inspired and based GROOT, an advanced open IDE for creating and debugging BehaviourTrees.

HARTU-APP-MANAGER is an open tool that will be made accessible through the GITLAB infrastructure created by HARTU.



Funded by the European Union

# 1 Table of contents



# List of figures





Funded by the European Union

# List of tables

**No se encuentran elementos de tabla de ilustraciones.**



Funded by the European Union

# Acronyms

| List of the acronyms | |
|---|---|
| **HARTU** | Handling with AI-enhanced Robotic Technologies for flexible manufactUring |
| **BT** | Behaviour Tree |
| **UI** | User Interface |
| **GUI** | Graphical User Interface |
| **CAD** | Computer Aided Design |



Funded by the European Union

# 1 Introduction

## 1.1 Objective

The objective of "Task 2.4 Application development support tool" is to provide a tool that facilitates the **definition** of a robotic application and its **control** by end users.

Initially the use of Node-RED was considered, an open programming tool based on Node.js (non-blocking event-driven model), whose browser-based editor would facilitate the connection of flows using the wide range of nodes already available or by creating custom nodes using the available editor. Thus, it would be possible to reuse functions, templates or flows.

However, after an analysis of other alternatives, it was decided to develop HARTU-APP-BUILDER based on the concept of Behaviour Trees, in particular the implementation based on BehaviorTree.CPP (see section 1.2).

It uses a similar concept, so it is possible to create the set of Nodes for each of the components developed in the project and integrate with ROS2 based functionalities such as global and local trajectory planning and controllers.

By means of the HARTU-APP-MANAGER, system Integrators / end users will create and control the application for the 7 use cases defined for the 5 industrial end users.

**IMPORTANT**: Although the title of this document refers to HARTU-APP-BUILDER, it is more appropriate to name it **HARTU-APP-MANAGER**, as this software offers two different functionalities:

- **HARTU-APP-CREATOR**, used to create and configure the robotic application
- **HARTU-APP-EXECUTOR**, used to control and monitor the execution of the robotic application.

## 1.2 Codeless programming

Codeless robot programming refers to the development and control of robotic systems using visual, intuitive, and user-friendly interfaces rather than traditional text-based coding. This approach aims to make robot programming more accessible to non-experts, allowing users to create, modify, and deploy robot behaviors and tasks without needing to write complex code.

The main features of codeless robot programming are the following:

1. **Visual Programming Interfaces:**

   Users interact with a graphical user interface (GUI) where they can drag and drop pre-defined blocks or nodes that represent various actions, conditions, and control structures. These blocks can be connected to form a flowchart or diagram that represents the robot's behavior and logic.

2. **Pre-defined Components:**



Funded by the European Union

Libraries of pre-built components or modules are available, covering common robot actions (e.g., moving, picking objects, sensing). Users can easily select and configure these components to fit their specific needs.

3. **Simulation and Testing:**

   Many codeless programming environments include simulators that allow users to test and validate their robot programs in a virtual environment before deploying them to the actual robot. This helps in identifying and fixing issues without risking damage to the physical robot.

4. **Interactive Debugging:**

   Tools for real-time monitoring and debugging of robot behavior are often integrated, allowing users to visualize the execution flow and identify problems interactively.

5. **Cross-platform Compatibility:**

   Codeless programming tools often support multiple robot platforms and hardware, providing flexibility and reducing the need for platform-specific expertise.

Examples of codeless programming are **Flowbotics Studio** (https://wiki.lynxmotion.com/info/wiki/lynxmotion/view/ses-software/flowbotics/) and **Intrinsic Flowstate** (https://www.intrinsic.ai/) to name a few.

Codeless robot programming democratizes the field of robotics by making it more accessible and easier to use. It leverages visual programming techniques to enable users to create, modify, and deploy robot behaviors without writing traditional code, thus accelerating development, reducing errors, and fostering collaboration.

HARTU has adopted this approach to develop the HARTU-APP-MANAGER tool.

## 1.3 Node-RED vs Behaviour Trees

As said above, a comparative analysis between Node-RED and Behaviour trees was carried out at the beginning of the project.

Node-RED and BehaviourTrees are both tools used for building and managing logic flows, but they serve different purposes and have distinct differences in their design, application, and use cases. Here's a breakdown of the main differences between the two:

**Node-RED:**

Node-RED is a flow-based development tool for visual programming, mainly used to connect hardware devices, APIs, and online services. It is commonly used in IoT (Internet of Things) applications, home automation, and integrating different systems and services.

Node-RED uses a directed graph where nodes represent operations or functions, and edges represent the flow of data between these nodes. The flow is built using a visual editor where nodes are dragged and dropped to create the logic flow. Each node performs a specific task and can be connected to multiple other nodes.



Funded by the European Union

Node-RED operates on an event-driven model where nodes react to incoming messages and pass messages to the next node. The flow is continuous and dynamic, based on events and data streams. State is often managed within the nodes or through external storage, and flows can be stateless or stateful depending on the design.

Node-RED is highly flexible, allowing for custom nodes to be created and integrated.

**Behaviour Trees:**

Behaviour trees are a hierarchical control structure used to define the logic of autonomous agents, primarily in robotics and game development.

They are used for AI behaviours in games, robotics control systems, and any application requiring complex decision-making and behaviour modelling.

Behaviour trees are structured hierarchically with nodes representing tasks or behaviours, connected in a tree structure. There are typically three types of nodes: control flow nodes (sequence, selector), decorator nodes, and leaf nodes (actions, conditions). The tree starts from a root node and branches out to child nodes. The execution flows from the root to the leaves, with control flow nodes determining the order of execution based on conditions and states.

Behaviour trees operate on a tick-based model where each node is ticked periodically, and the status (success, failure, running) is propagated up the tree. This allows for reactive and adaptive behaviour. State is managed within the tree structure, with nodes remembering their status between ticks, which is crucial for complex behaviour modelling. To learn more, visit the page Introduction to BTs.

Behaviour trees offer flexibility in defining behaviours through combinations of different node types. They are modular and reusable, making it easy to design and test complex behaviours. Custom behaviours can be created by defining new types of leaf nodes or decorators, allowing for specific actions and conditions to be integrated into the tree.

In summary, each tool has its strengths and is designed to address specific types of problems, so the choice between Node-RED and behaviour trees depends on the particular requirements and context of the project. Node-RED is best suited for IoT, system integration, and event-driven applications, while behaviour trees is best for AI behaviours in robotics, offering structured and hierarchical control over complex decision-making processes.

HARTU has selected Behaviour Trees to implement HARTU-APP-MANAGER, in particular its implementation using BehaviorTree.CPP 3.8, a C++ library to build Behaviour Trees. The GUI is inspired and based on GROOT, an advanced open IDE for creating and debugging Behaviour Trees.

# 2 HARTU-APP-MANAGER: common entry point

Through the **HARTU-APP-MANAGER** the user selects which functionality wants to access. In both cases, the user must first select the functionality and the press the START button. The corresponding GUI (HARTU-APP-CREATOR or HARTU-APP-EXECUTOR) will then appear.



Funded by the European Union


*Figure 1. Common access point of the HARTU-APP-MANAGER*

# 3 Robotic application builder: HARTU-APP-BUILDER

## 3.1 HARTU-APP-BUILDER: Main interface description

The main interface is shown in Figure 2

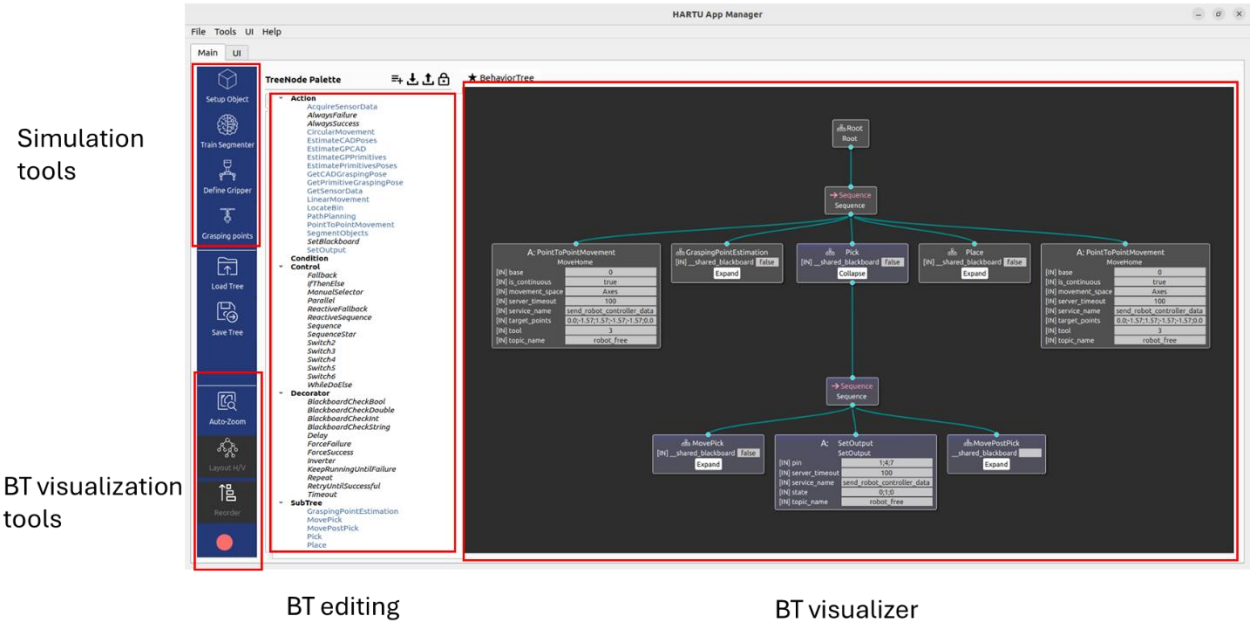



*Figure 2. HARTU-APP-CREATOR GUI*

The GUI consists of 3 basic panels:

- On the left side there is an icon bar to access to the simulation tools, manage the BTs and configure the way BTs are displayed
  - Simulation tools
  - Management of Trees
  - BT Visualization
- In the centre is a palette of components for the creation of the tree. It includes 4 categories: Action, Condition control, Decorator and SubTree
- On the right is the BT visualizer, where the tree is presented graphically



Funded by the European Union

The red/green icon in the lower left corner indicates if there is an error in the BT definition. At the top there are 4 additional icons (see Figure 3) for adding custom nodes, loading a tree, saving the current tree and locking/unlocking BT editing.

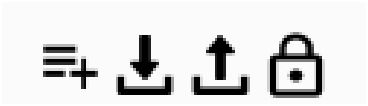


*Figure 3. From left to right: Icons for adding custom nodes, loading a tree, saving the current tree and locking/unlocking BT editing*

The BT is created by dragging and dropping the components on the palette. If the component is an action or a subtree, the user must update the values of its parameters.

## 3.2 Icon bars

### 3.2.1 Simulation tools

#### *3.2.1.1 Setup Object*

It opens the GUI to configure the characteristics that define the visual appearance and some physical properties (weight and friction) of the object. The first ones are needed for the generation of image datasets, and the second ones for testing the grasping points.

The description of these interfaces is included in "D2.3 Simulation infrastructure for handling component training".

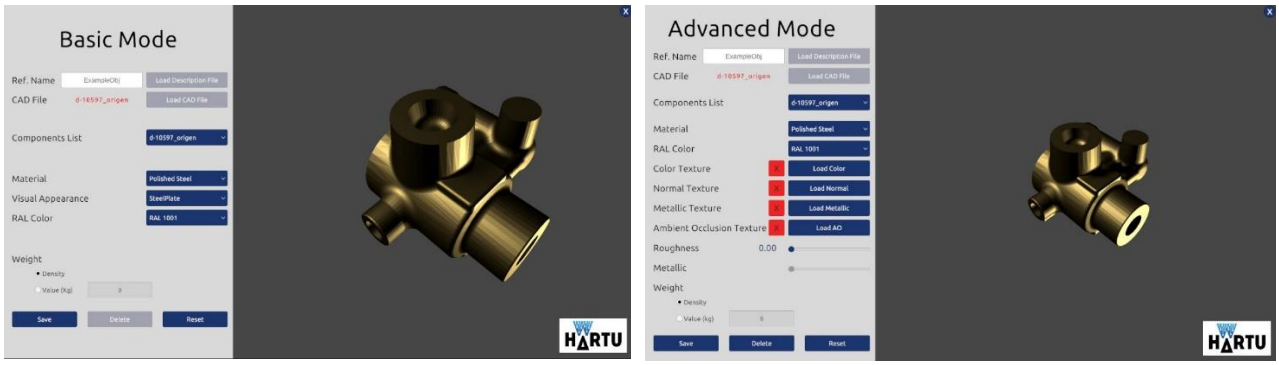



*Figure 5. Interface for defining the visual appearance and physical characteristics of parts: Basic and Advanced Mode*

#### *3.2.1.2 Train Segmenter*

It give access to two different interfaces: one to define scenes and one to define the dataset characteristics.

The description of these interfaces is included in "D2.3 Simulation infrastructure for handling component training".



Funded by the European Union

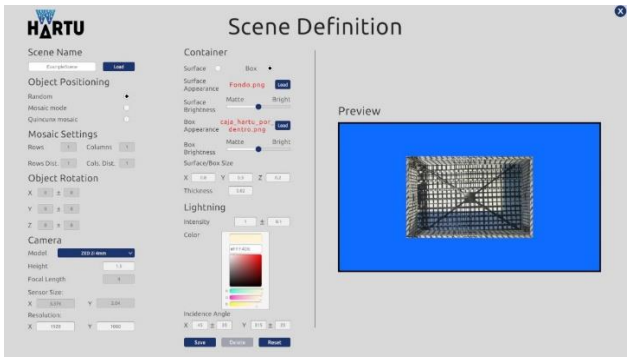



*Figure 4. Interface to define a scene*


*Figure 5. Interface to define the dataset of images to be generated*

Once the dataset is generated the user call the python script to train the object detection model (YOLOv5).

### *3.2.1.3 Define gripper*

It displays an interface to select/define the gripper that will be used by the LocalGraspPlanner to identify the grasping points of a new object:

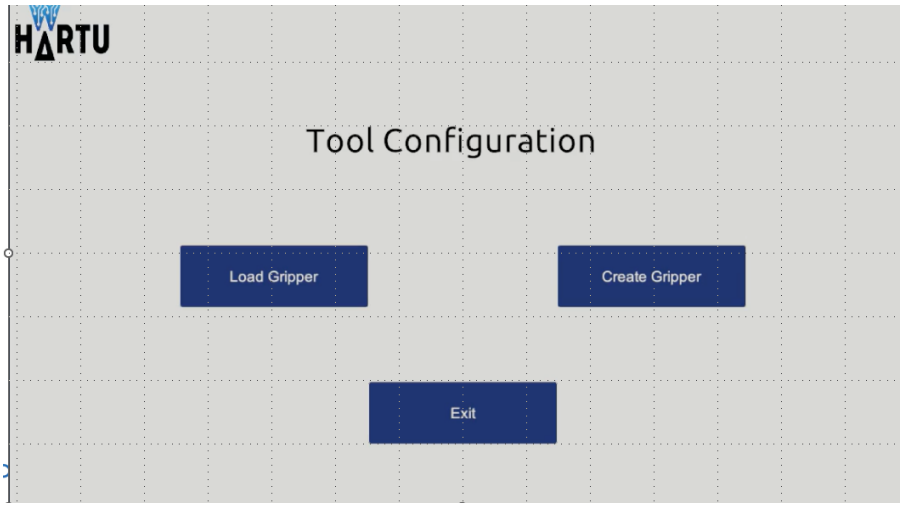



*Figure 6. Interface to configure the gripper for the LocalGraspPlanner*

The user can load an existing one of create a new one:

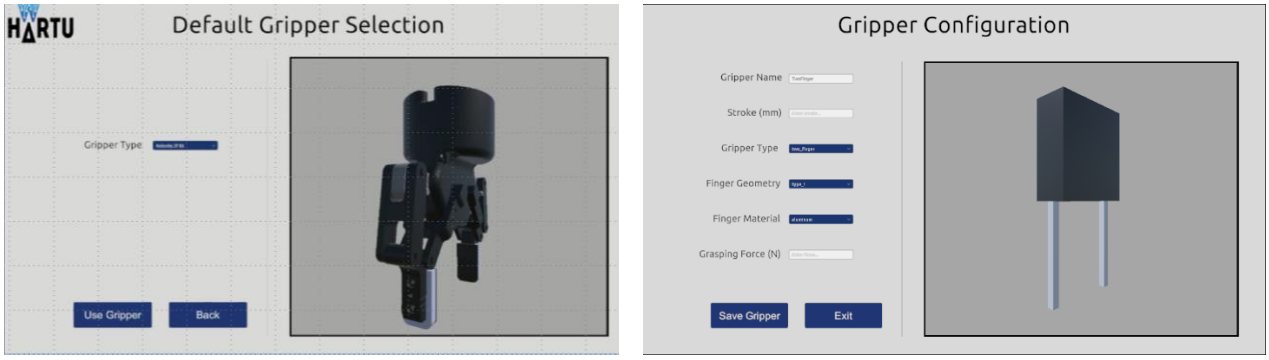



*Figure 7. Interfaces to select an existing gripper (left) or define a new one (right)*

### *3.2.1.4 Grasping points*

It executes a python script that launches a ROS2 node with the pipeline to estimate the grasping points (*LocalGraspPlanner*).



Funded by the European Union

### 3.2.2 Programming by demonstration

It allows calling two ROS services:

- Perform recording. The data (trajectories) are saved in a file.
- Learning the model. The learned model is also saved in a file.

### 3.2.3 BT management

The two icons allow loading an existing Tree from the repository or saving the current tree.

### 3.2.4 BT Visualization

It includes three icons that allow modifying the way the BT are presented in the visualization area:

- Auto Zoom: It adjusts the complete BT in the visualization area
- Layout H/V: as sometimes the BT may grow horizontally or vertically, the user can choose the way it is presented:

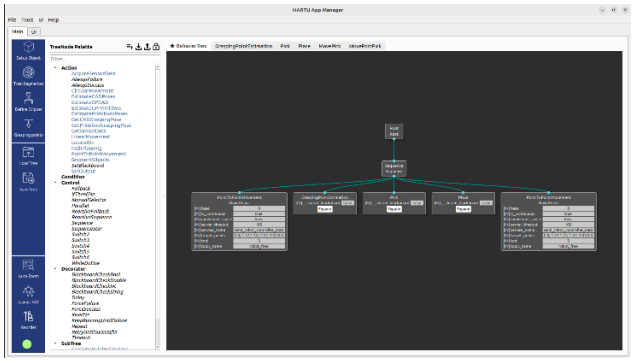


*Figure 8. BT arranged horizontally*

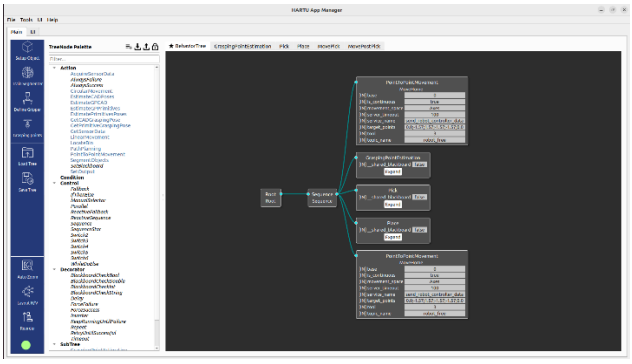


*Figure 9. BT arranged vertically*

- Reorder: Reorders all nodes of the BT

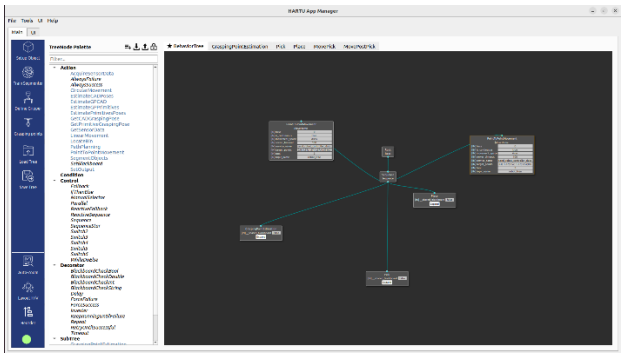


*Figure 10. BT Original non-ordered BT*

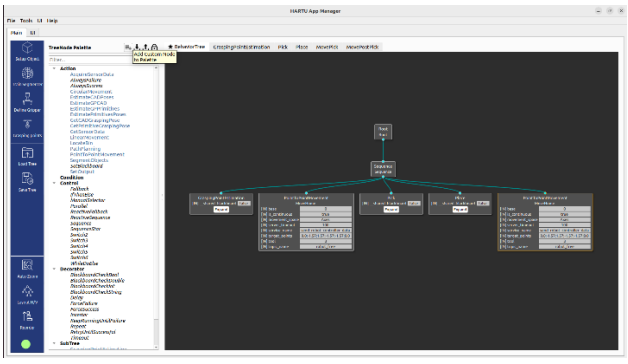


*Figure 11. BT after reordering*



Funded by the European Union

## 3.3 Palette of components

The interface includes the basic palette available in GROOT with the additional ones developed in HARTU.

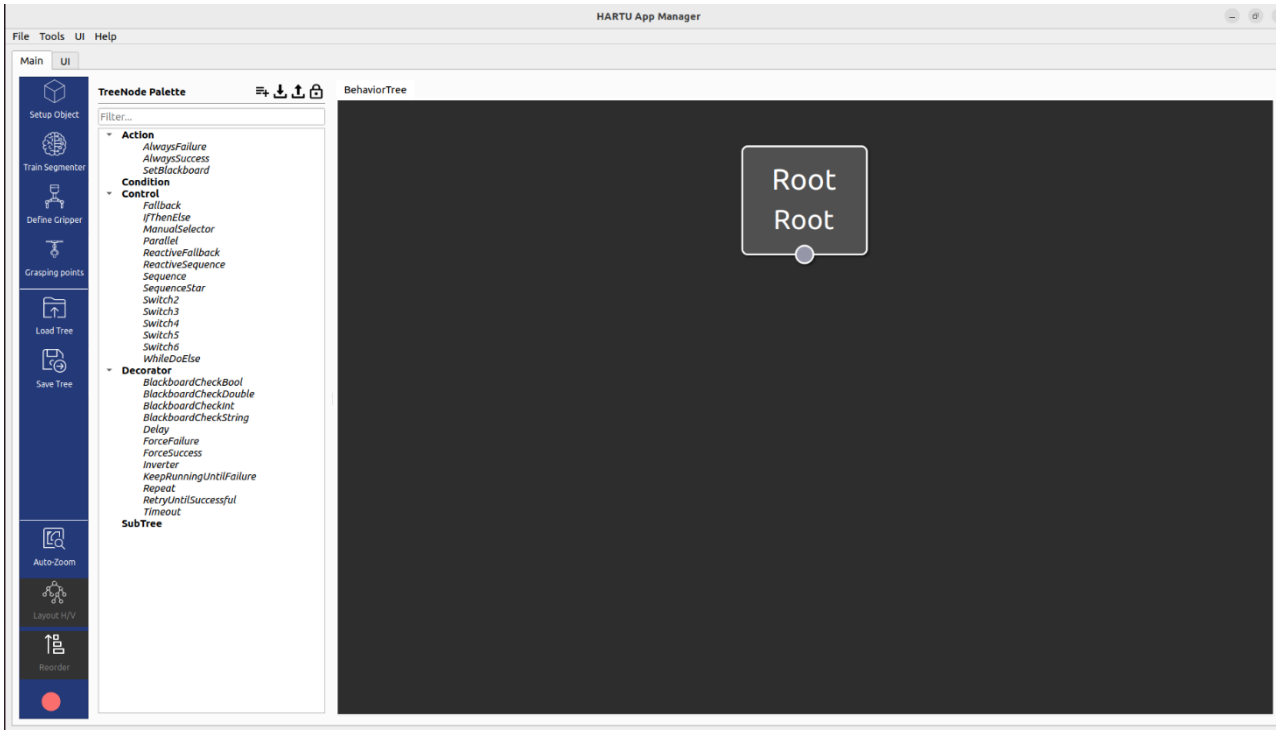



*Figure 12. Basic elements of the GROOT pallete*

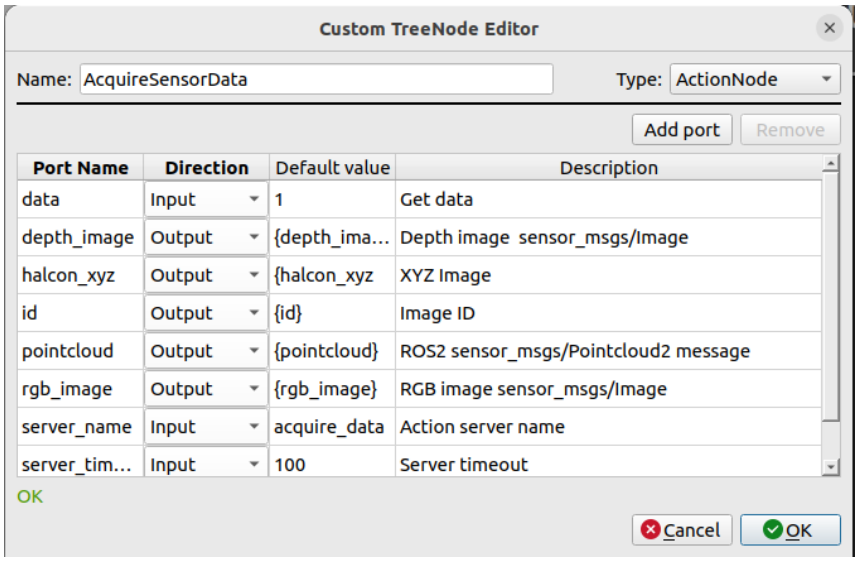
### 3.3.1 Action

HARTU-APP-CREATOR offers a library of BT nodes for the most common functionalities needed to build a robotic application.

The description of the parameters of each node are accessible by pressing the right buttom of the mouse on the corresponding action as in the example for the AcquireSensorData:



Funded by the European Union




Advanced users can create their own actions using BehaviorTree.CPP 3.8

#### *3.3.1.1 AcquireSensorData*

Calls to the ROS2 node that implements the acquisition of an image. Currently it is available for Photoneo and ZEDi.

When the *data* parameter is set to 1 it is called synchronously and doesn't return until the image is available. With the *data* set to 0 is called asynchronously and returns the control immediately. In this later case, the image can be read using the GetSensorData function (see section 3.3.1.11).

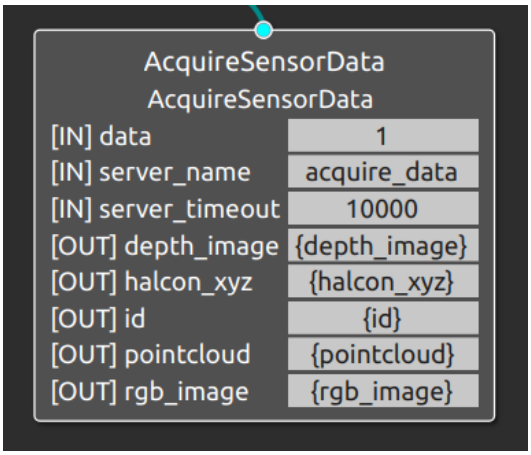



*Figure 13. AcquireSensorData node*

#### *3.3.1.2 ChangeFrame*

Changes the frame of the camera to that specified in the node.



Funded by the European Union

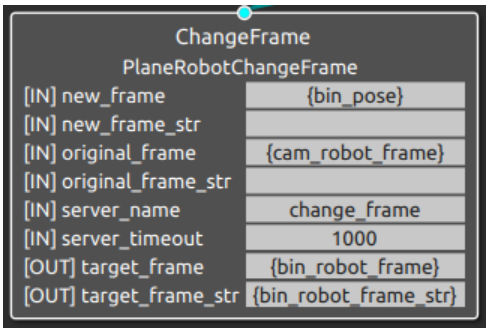



*Figure 14. ChangeFrame node*

### *3.3.1.3 CircularMovement*

Calls the ROS2 node that generates a circular movement.

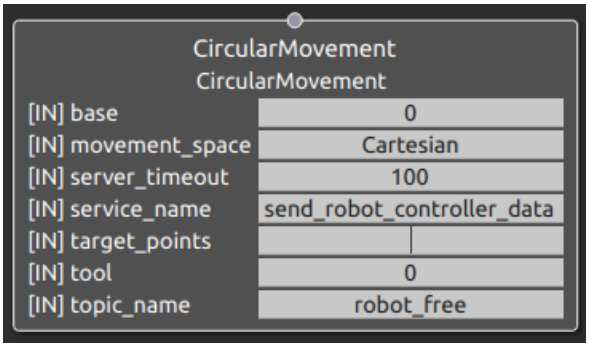



*Figure 15. CircularMovement node*

### *3.3.1.4 EstimateCADPoses*

Calls the ROS2 node that provides the pose of the segmented objects in the image using their corresponding CAD model.

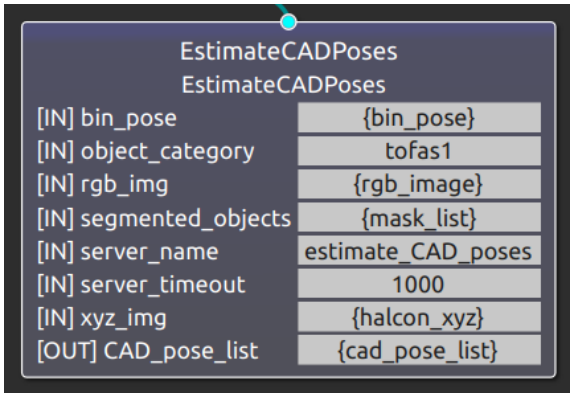



*Figure 16. EstimateCADPoses node*

### *3.3.1.5 EstimateGPCAD*

Calls the ROS2 node that provides the pose of the grasping points on the segmented objects in the image, when the CAD is available.



Funded by the European Union

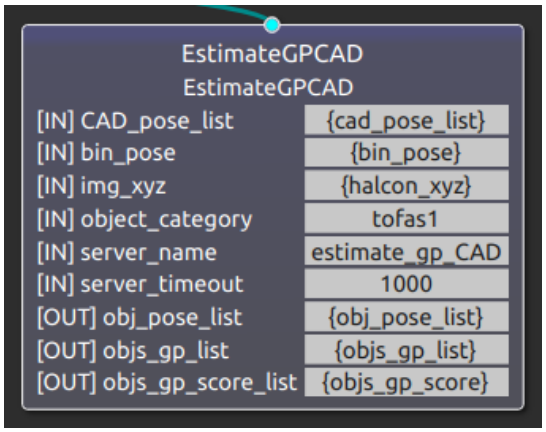



*Figure 17. EstimateGPCAD node*

#### *3.3.1.6 EstimateGPPrimitives*

Calls the ROS2 node that provides the pose of the grasping points on the segmented objects in the image (the corresponding to the primitives), when there is not CAD is available.

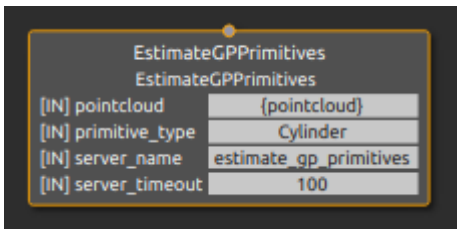



*Figure 18. EstimateGPPrimitives node*

#### *3.3.1.7 EstimatePrimitivesPoses*

Calls the ROS2 node that provides the pose of the segmented objects (their corresponding primitives) in the image when there is not CAD model available.

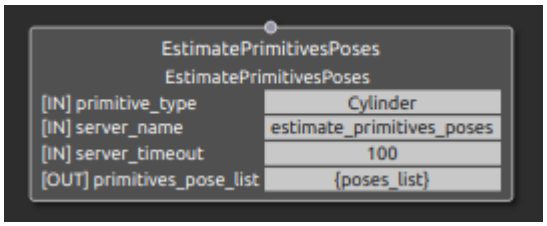



*Figure 19. EstimatePrimitivesPoses node*

#### *3.3.1.8 ExecuteLearnedMovement*

Executes the learned movements during the learning from demonstration.



Funded by the European Union

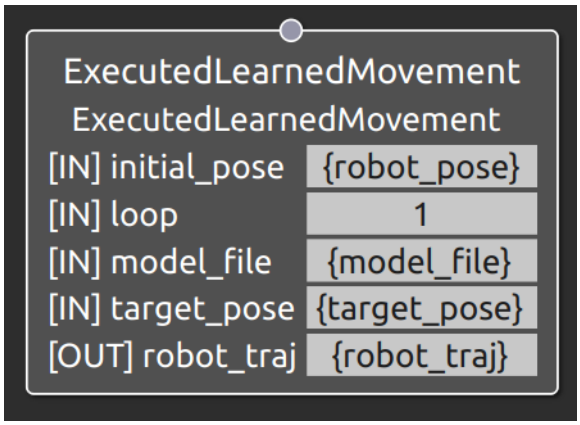



*Figure 20. ExecuteLearnedMovement node*

### 3.3.1.9 GetCADGraspingPose

Calls the ROS2 node that returns the best grasping points between all candidates in the scene, when there is CAD model available.

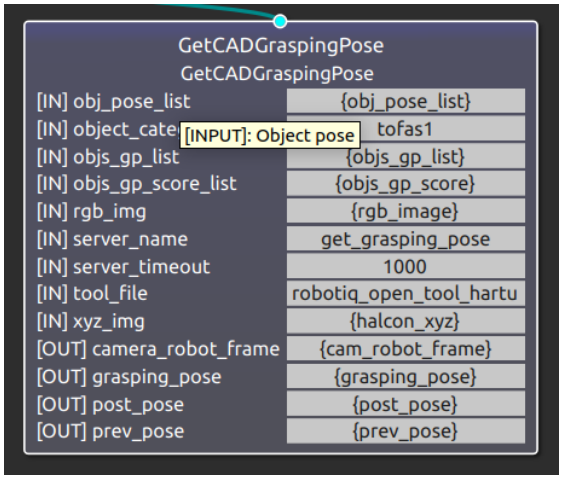



*Figure 21. GetCADGraspingPose node*

### 3.3.1.10 GetPrimitiveGraspingPose

Calls the ROS2 node that returns the best grasping points between all candidates in the scene, when no CAD model available.

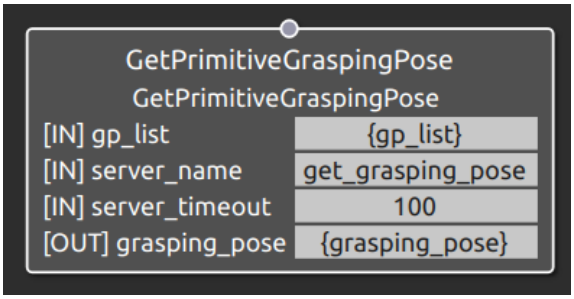



*Figure 22. GetPrimitiveGraspingPose node*

### 3.3.1.11 GetSensorData

Retrieves the image acquired by the corresponding *AcquireSensorData* when in the latter the *data* parameter is set to 0. In this case *AcquireSensorData* is called asynchronously and returns the



Funded by the European Union

control immediately, otherwise, with *data* set to 1 it is called synchronously and doesn't return until the image is available.

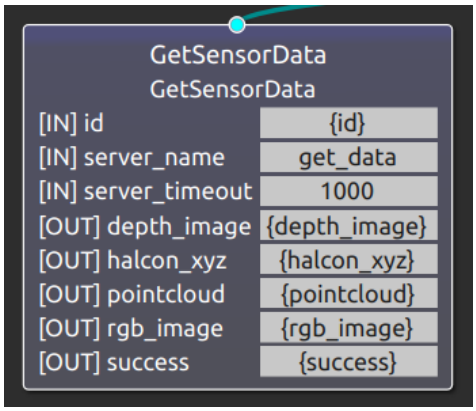



*Figure 23. GetSensorData node*

### *3.3.1.12 LinearMovement*

Calls the ROS2 node that generates a linear movement from the current position to the target position.

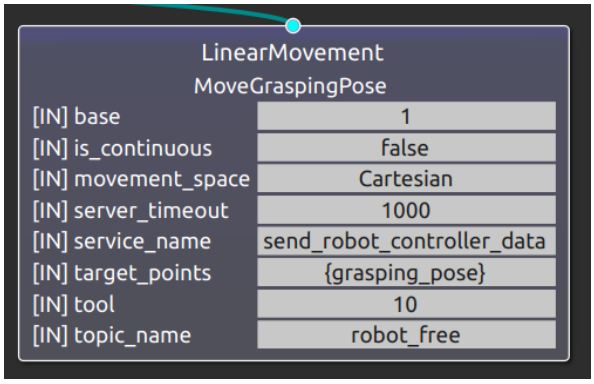



*Figure 24. LinearMovement node*

### *3.3.1.13 LocateBin*

Calls the ROS2 node that provides the location of either the work surface or the bin/container in the scene where parts can be found.

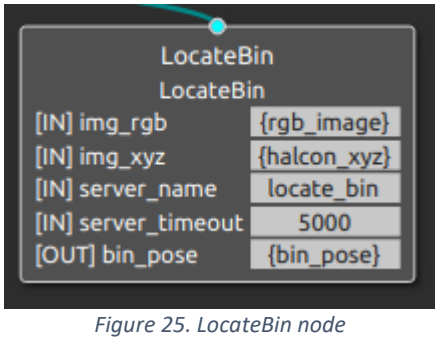



*Figure 25. LocateBin node*

### *3.3.1.14 PathPlanning*

Calls the planner selected in MoveIt, to plan the trajectory given the initial and final points.



Funded by the European Union

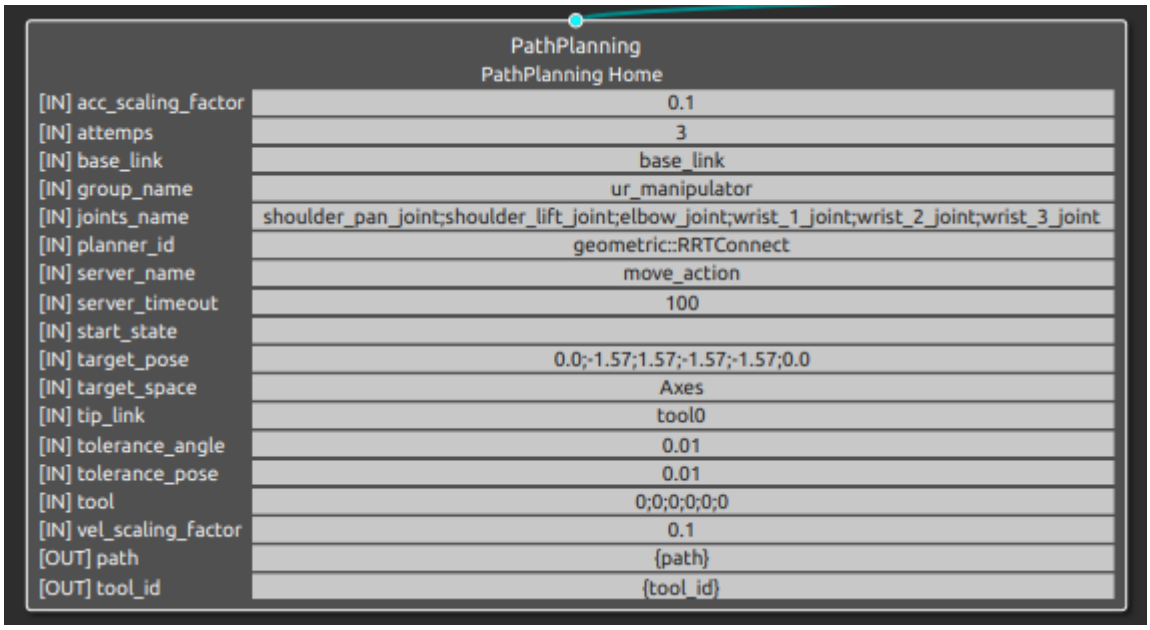



*Figure 26. PathPlanning node*

### *3.3.1.15 PointToPointMovement*

Calls the ROS2 node that generates a point-to-point trajectory either in cartesian or joints.

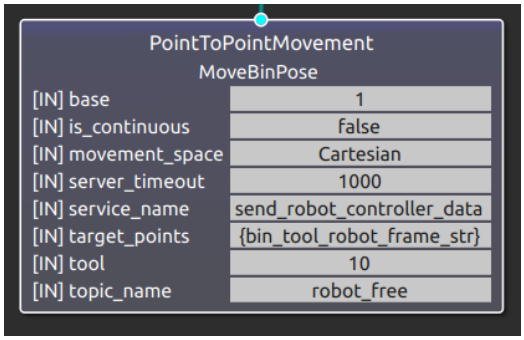



*Figure 27. PointToPointMovement node*

### *3.3.1.16 SegmentObjects*

Calls the ROS2 node that generates the masks of the segmented objects in the RGB image given the trained model.

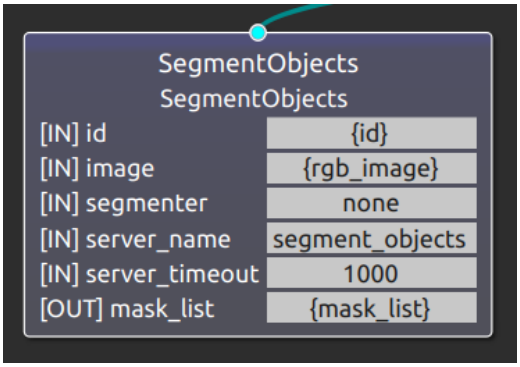



*Figure 28. SegmentObjects node*

### *3.3.1.17 SetOutput*

Calls the ROS2 driver (ROS2 node) that sets the digital output signal to true of false



Funded by the European Union

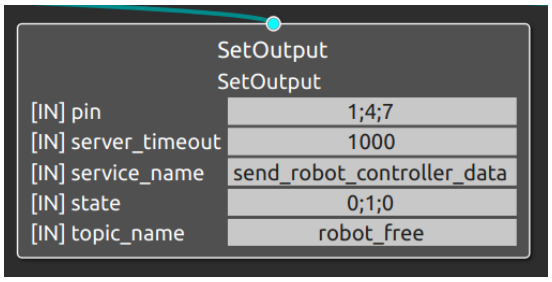



*Figure 29. SetOutput node*

#### *3.3.1.18 SetRobotiqGripper*

Calls the RobotiQ gripper ROS2 driver (implementing MODBUS communication) that opens or closes the tool.

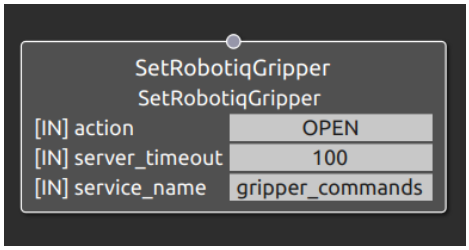



*Figure 30. SetRobotiqGripper node*

### 3.3.2 Condition control

The user has at his/her disposal the condition controls that are common in many programming languages. Details are available at https://www.behaviortree.dev/.

- *Fallback*
- *IfThenElse*
- ManualSelector
- Parallel
- ReactiveFallback
- ReactiveSequence
- Sequence
- SequenceStar
- Switch1 to Switch6
- WhileDoElse

A detailed explanation of this controls is available here:

### 3.3.3 Decorator

Decorators are special types of nodes that modify the behaviour of other nodes. They act as intermediaries, adding additional control logic to the execution flow of the tree. Decorators can change the way a node or subtree is executed based on conditions or rules. For instance, the repeat decorator can be used to execute an action (or subtree) as many times as indicated:



Funded by the European Union

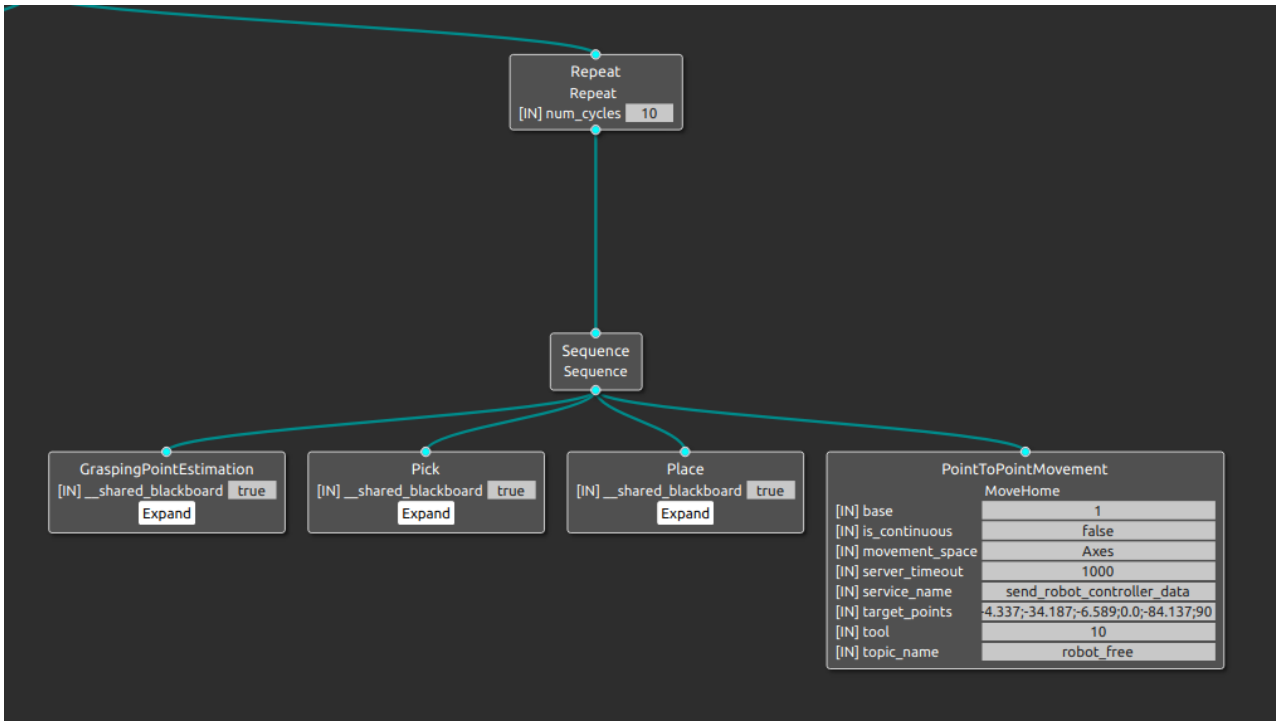



*Figure 31. Example of use of repeat decorator*

Details are available at https://www.behaviortree.dev/.

- *BlackboardCheckBool*
- *BlackboardCheckDouble*
- *BlackboardCheckInt*
- *BlackboardCheckString*
- *Delay*
- *ForceFailure*
- *ForceSuccess*
- *Inverter*
- *KeepRunningUnitlFailure*
- *Repeat*
- *RetryUntilSuccessful*
- *Timeout*

### 3.3.4 Subtrees

It is very common that a set of actions are repeated in different robotic applications. HARTU-APP-CREATOR offers the possibility to define then as a subtree that can be then reused. Advanced users can create their own subtrees.

The subtrees currently available are:



Funded by the European Union

### 3.3.4.1 GraspingPointEstimation

It implements the various actions necessary to select the grasping point of the object that the robot will try to pick up in a given scene: acquiring the image, segmenting it and estimating the pose of the grasping point candidates.

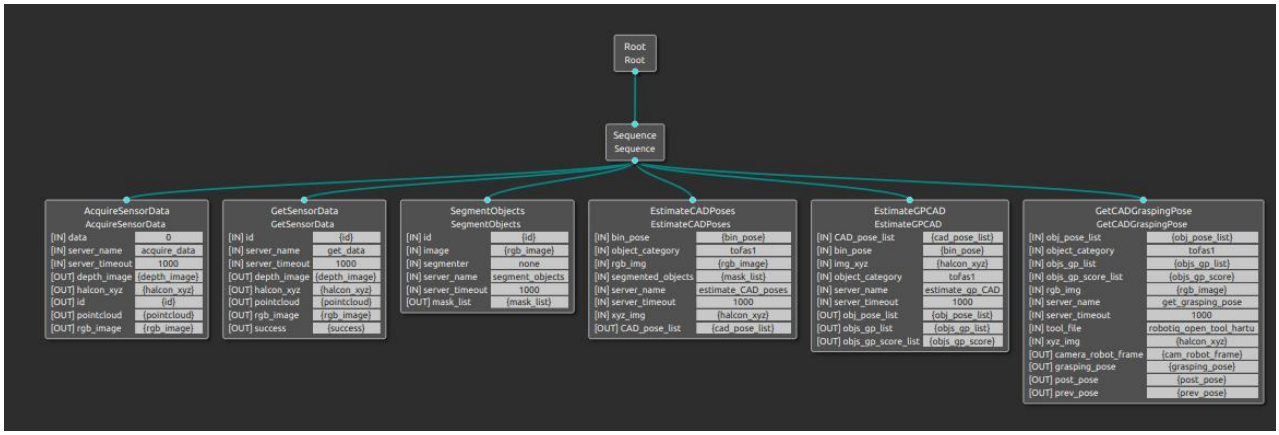



*Figure 32. GraspingPointEstimation Subtree*

### 3.3.4.2 MovePick

It implements a set of movements to safely approach the pick position(just before activating the gripper).

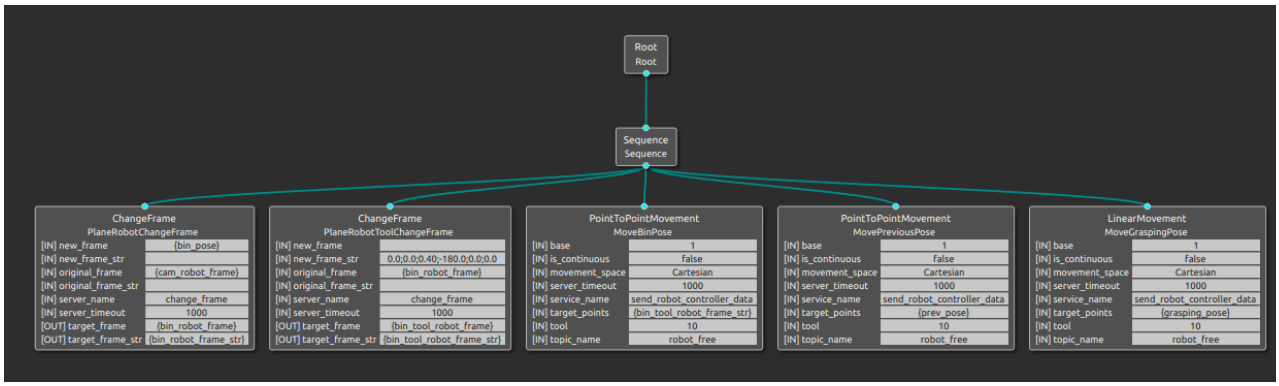



*Figure 33. MovePick Subtree*

### 3.3.4.3 MovePostPick

It implements a set of movements to safely reverse the movements after the picking operation.



Funded by the European Union

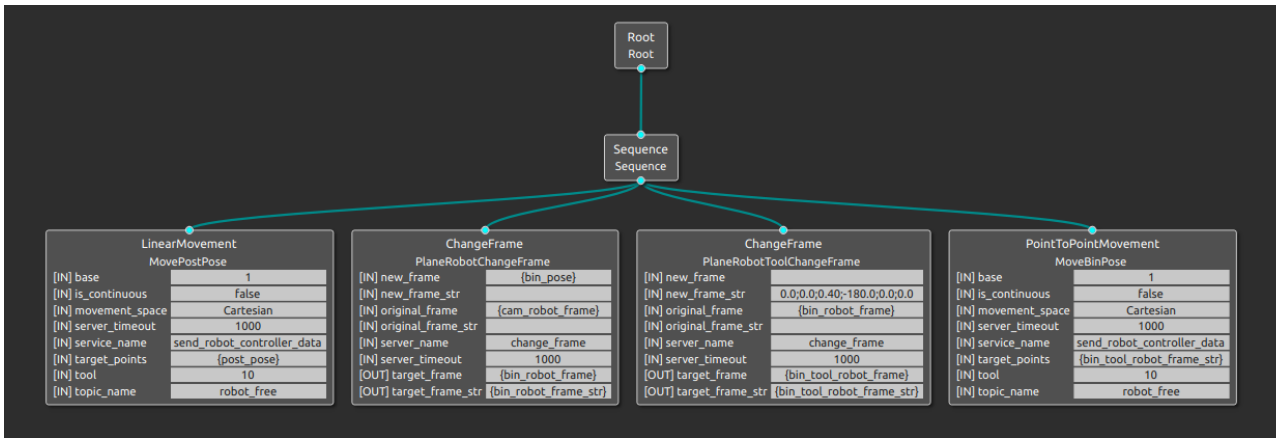



*Figure 34. MovePostPick Subtree*

### *3.3.4.4 Pick*

It implements the complete picking operations as a combination of MovePick, SetRobotiquGrippper (or SetOutput) and MovePostPick

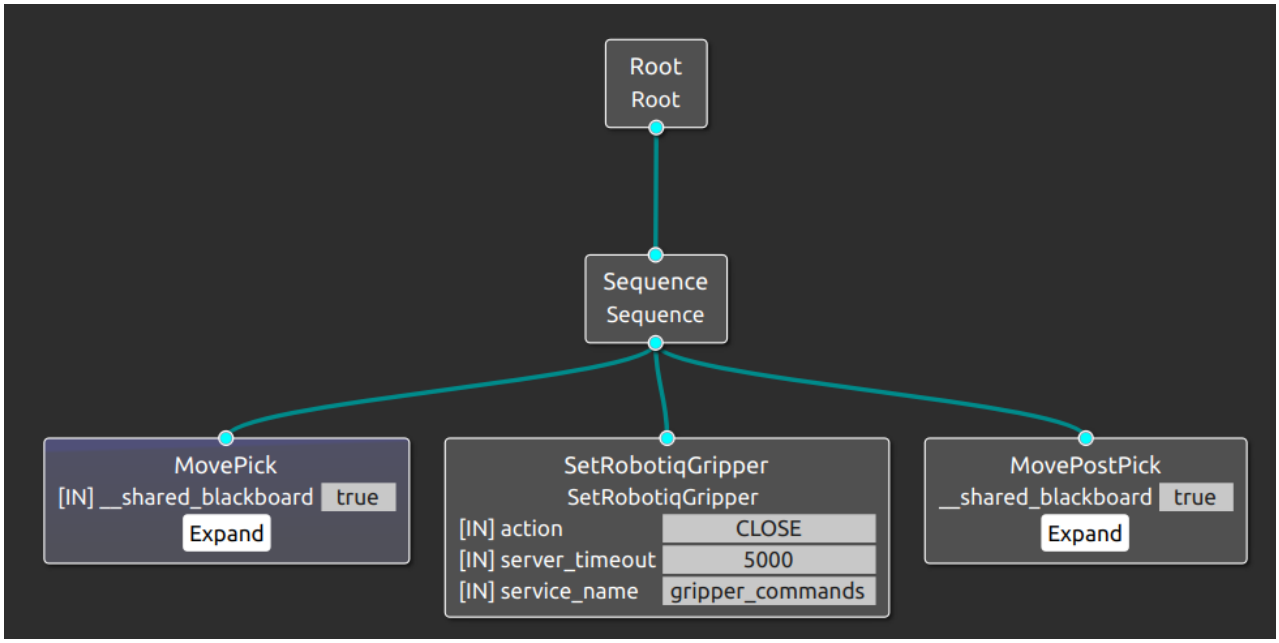



*Figure 35. Pick Subtree*

### *3.3.4.5 Place*

It implements the actions to move to a destination and open the gripper to release the part held by the robot.



Funded by the European Union

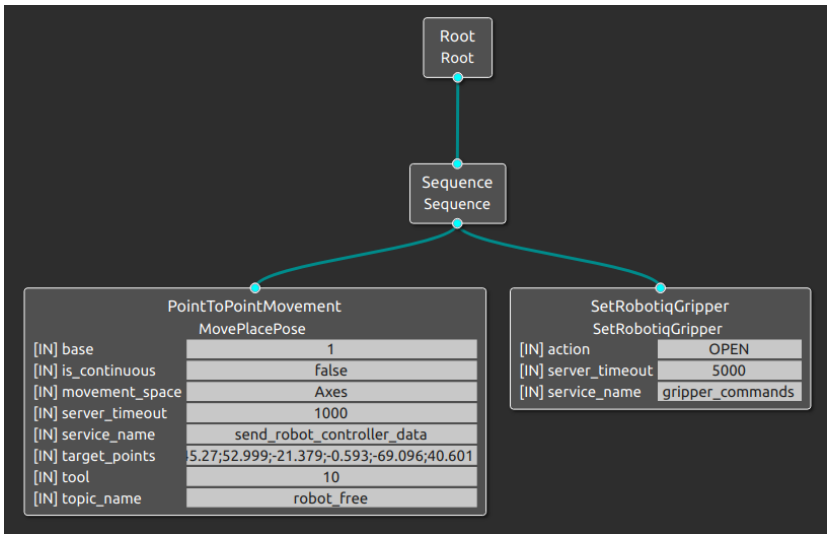



*Figure 36. Place Subtree*

# 4 Control of robotic applications: HARTU-APP-CONTROL

## 4.1 Behaviour trees: workflow overview

Visit the page Introduction to BTs for an explanation on how this workflow works.

## 4.2 HARTU-APP-CONTROL: Main Interface description

The main interface is shown in Figure 37:

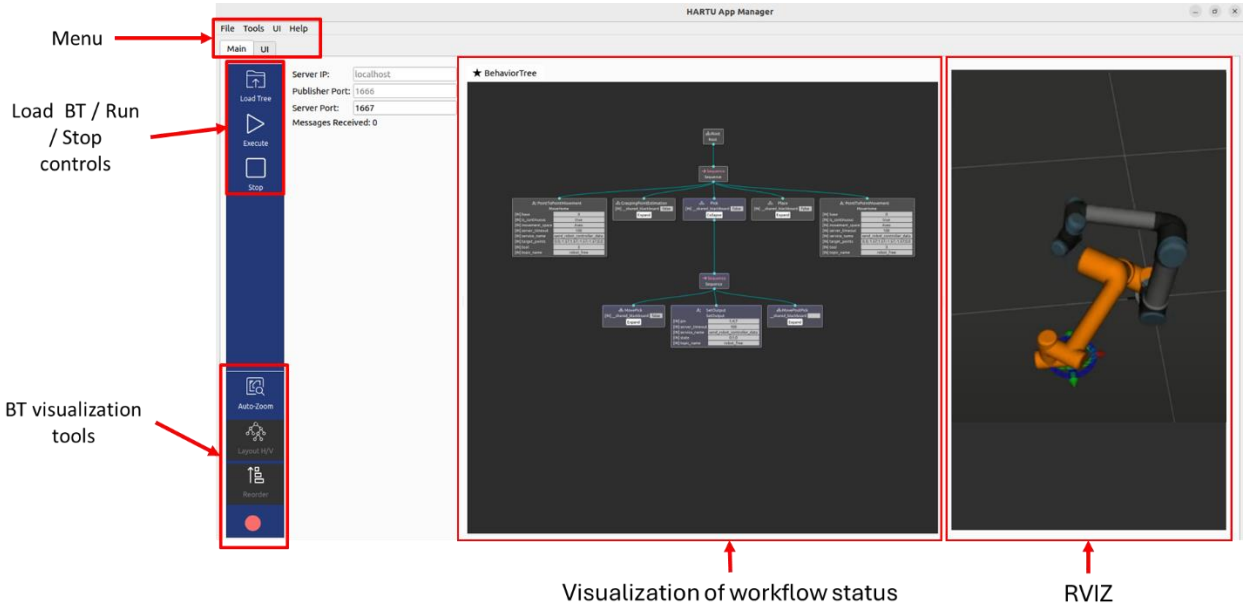



*Figure 37. Common interface to visualize the flow of actions (BT nodes)*

The GUI consists of 3 basic panels:



Funded by the European Union

- On the left side there is an icon bar to load trees, execute and stop then, as well as the BT visualization tools (see description in section 3.2.4).
- In the centre is displayed the BT: running nodes (orange), successfully executed nodes (green) and unsuccessfully execute ones (red)

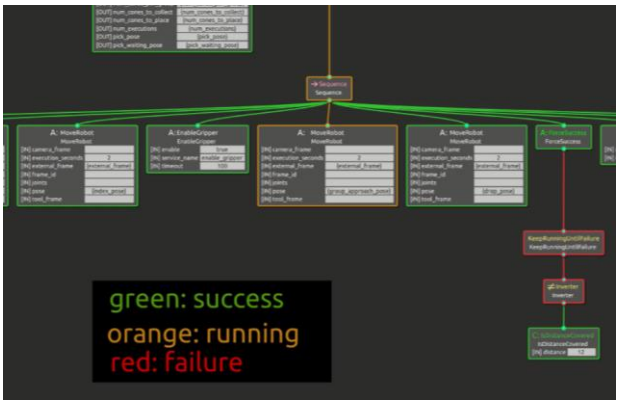



*Figure 38. Colours for the different states of execution of the nodes*

- On the right is displayed the RVIZ panel.

## 4.3 HARTU-APP-CONTROL: Use Case dependent Interface description

In addition to the main interface described above, each application may require a specific UI to define some parameters, to display application parameters and messages and to manually command some actions (e.g. stop, resume).

This UI is accessible through the UI window and gives access to the user-defined display, an example of such a UI is presented in Figure 39.

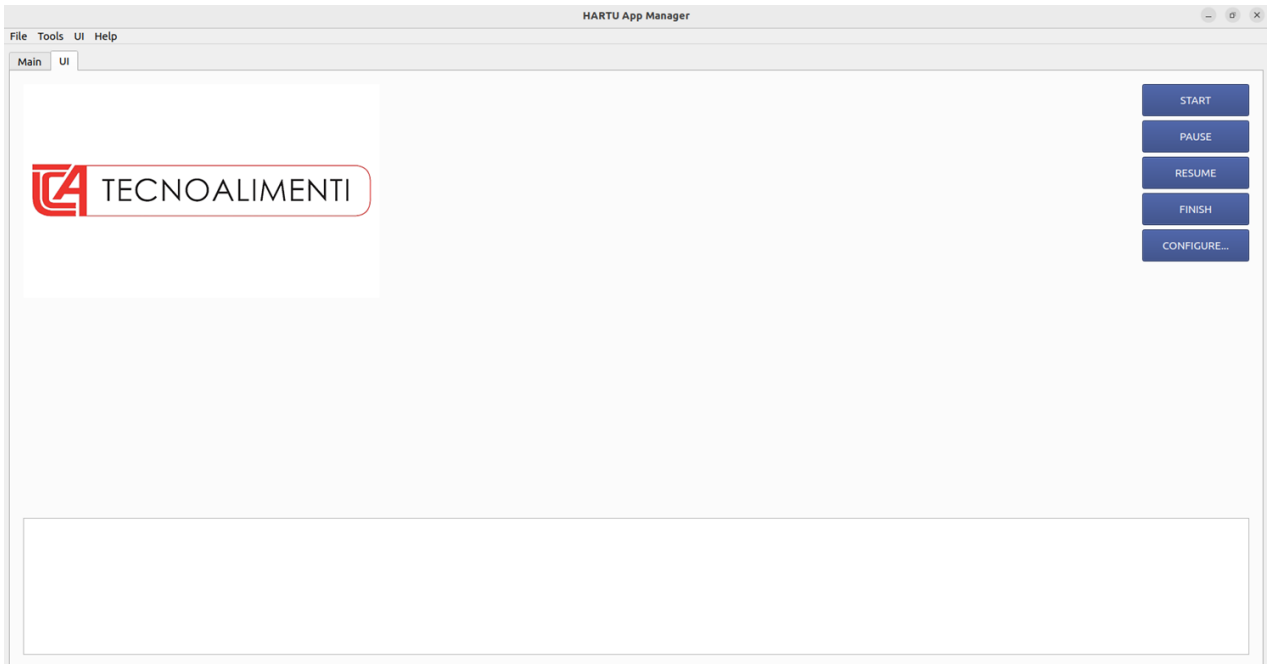



*Figure 39. Interface to be customize for each Use Case*

The development of this interface is done using QT.



Funded by the European Union

# 5 HARTU-APP-MANAGER Deployment

The readme.txt available in the https://gitlabpa.eng.it repository explains how to deploy the application.



Funded by the European Union