UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

MASTER'S DEGREE IN CONTROL SYSTEMS ENGINEERING

Hierarchical Task Planning for Human-Robot Collaborative Assembly

Supervisor
Prof. Carli Ruggero

Co-supervisor
Dott. Giacomuzzo Giulio

Candidate
Pegoraro Giulia

ACADEMIC YEAR 2023-2024

Graduation date 04/12/2024

Abstract

In this thesis, we address the challenges of task representation and task planning for assembly within a communication-free Human-Robot Collaboration (HRC) framework. Our primary goal is to present a novel task planning approach that does not rely on modeling the human as a predictable agent. Instead, we treat the human as an unpredictable agent and focus on optimizing the robot's actions. Building on the existing Hierarchical Task Network (HTN) planning framework, UHTP (User-aware Hierarchical Task Planning), we present an implementation based on personal interpretation, extending it to handle joint actions and incorporate a failure recovery process. The secondary goal is to develop a method for autonomously retrieving task representations from annotated videos, organizing tasks into a hierarchical structure. The proposed solutions are validated through simulation experiments and compared with others to demonstrate their robustness and effectiveness.

Sommario

In questa tesi affrontiamo le sfide legate alla rappresentazione e pianificazione del task per l'assemblaggio in un contesto di Human-Robot Collaboration (HRC) senza comunicazione. Il nostro obiettivo principale è presentare un nuovo approccio di pianificazione del task che non si basi sulla modellizzazione dell'essere umano come agente prevedibile. Al contrario, consideriamo l'essere umano come un agente imprevedibile e ci concentriamo sull'ottimizzazione delle azioni del robot. Basandoci sul framework esistente di task planning con Hierarchical Task Networks (HTN), UHTP (User-aware Hierarchical Task Planning), proponiamo un'implementazione derivata da un'interpretazione personale, estendendola per gestire azioni congiunte e integrare un processo di ripristino in caso di fallimento delle azioni. L'obiettivo secondario è sviluppare un metodo per ottenere in modo autonomo rappresentazioni del task da video trascritti, organizzando i compiti in una struttura gerarchica. Le soluzioni proposte sono validate attraverso esperimenti di simulazione e confrontate con altre metodologie per dimostrarne la robustezza ed efficacia.

Contents

Chapter 1

Introduction

In recent decades, technological innovation has profoundly transformed the industrial world, introducing advanced automation and artificial intelligence into manufacturing processes. One of the most promising developments in this context is represented by Human-Robot Collaboration (HRC). HRC is an innovative paradigm that aims to combine the unique capabilities of humans, such as manual dexterity and critical thinking, with the strength, precision, and repeatability of industrial robots. Thanks to advanced sensors and artificial intelligence algorithms that ensure safe and smooth interaction, collaborative robots, also known as cobots, leave the role of simple machine tools and become active partners for human workers. Different from traditional industrial robots, modern cobots are designed to work side by side with humans, sharing workspaces and tasks without the need for physical safety barriers. As a result, the novel HRC framework promises increased productivity and efficiency of workflows.

Assembly is one of the most relevant and complex tasks in the manufacturing industry. Assembly processes, indeed, are challenging as they typically involve a long sequence of subtasks. Each subtask may require particular strength to move heavy objects or dexterity to perform complex actions such as insertions or screwing. In this context, HRC results to be an effective solution both to improve efficiency and alleviate human workers from tedious and repetitive tasks.

Implementing an efficient HRC framework for assembly tasks, however, is also challenging. As said, assemblies require a long series of actions to be performed in a sequence, with specific constraints on their order. First, an efficient abstraction of the task, hereafter denoted as task representation or task description, is required, which describes the task goal and the constraints on action sequentiality. Besides efficient task representation, precise coordination between the human worker and the robot is fundamental: the robot should be continuously updated on the action performed by the human partner and, ideally, it should be aware of the human's intention, either through explicit communication or by movement interpretation. Finally, an effective task planning algorithm is needed to optimize the robot's action sequence according to the assembly state and the human's choices. All these aspects make collaborative assembly a challenging problem, subject to an intense research effort.

Several solutions have been proposed in the literature. Regarding task description, the large majority of presented works rely on hierarchical task representations, which group the entire set of actions to be performed in a hierarchy of subtasks of increasing (or decreasing) complexity, and organize them in a graph or tree structure. Hierarchical approaches are appealing for two main reasons: on the one hand, they explicitly model the inherent relationships of sequentiality or independence between subtasks or manipulated objects. On the other hand, grouping tasks into a hierarchical structure reduces the description complexity. Widespread hierarchical representations include, to name a few, And/Or graphs, Behavior Trees, Scheduling Algorithms, and Hierarchical Task Networks (HTNs). In particular, as it will be clearer in the following, HTNs are largely involved in the context of planning, thanks to their straightforward definition and interpretability.

Manually deriving task descriptions is a tedious and time-consuming task, which requires domain expertise and could become difficult to infeasible for very long assemblies with a large number of actions involved. For these reasons, deriving the task description automatically, for example from video demonstrations, is a widely discussed topic in the literature. Zhuo et al. [1] construct HTN trees starting from partially observed plan traces, the task is annotated with a focus on its preconditions and effects. The algorithm presented returns a series of decomposition trees, each referring to a higher-level task. Hayes et al. [2] introduce Clique/Chain HTNs, a new method based on Conjugate Task Graphs, graphical representations that aim to simplify and outline a hierarchical structure between tasks. Danfei Xu et al. [3] propose a learning algorithm based on neural networks and neural programming that takes as input generalized video demonstrations and decomposes their hierarchically complex structures into subtasks.

For what concerns planning algorithms, most of the solutions proposed rely on hierarchical representations. Prior works that adopt And/Or graphs are, for example, Darvish et al. [4] and Johannsmeier et al. [5]. Despite being effective, these methods reveal problems in online task replanning due to the increasing complexity of And/Or graphs for large-scale problems. Moreover, these approaches require the human agent to follow predefined plans, which can be restrictive. Differently, Fusaro et al. in [6] and [7] employ Behavior Trees due to their modularity and responsiveness, which allow the handling of different levels of human-robot interaction. However, also in this case the human agent is constrained to choose a certain path established a priori, based on cost optimization. Scheduling Algorithms such as Chaski [8] or APA [9] are also widely utilized. Chaski enables the robot to dynamically adapt to human behavior, though it explicitly requires communication between partners. APA synchronizes robot scheduling with human preferences, treating the robot as a follower, which may not fully leverage the robot's potential in collaboration. Milliez et al. [10] implement an adaptive task planning using HTNs, ensuring that human preferences and constraints are respected throughout the process. However, a limitation of this framework is that human behavior must be explicitly modeled. Roncone et al. [11] propose a planning approach based on HTNs aimed at reducing the human's cognitive load. Nevertheless, in their specific scenario, explicit communication between agents becomes essential.

Note that all the aforementioned works either suppose the human to follow a prescribed plan or require explicit communication between agents. These two aspects can reduce the naturalness of the interaction, leading to decreased efficiency. Similar to what happens among human teams, where one agent tries to understand the others just by observing their behavior, one would expect the robot to interpret the human and adapt its choice accordingly, without explicit communication if not needed. Moreover, in many situations, it is desirable that the robot proactively supports the human partner, neither prescribing actions for the human co-worker nor acting completely as a follower. For the aforementioned reasons, in this thesis, we consider a human-robot team where the robot adapts itself to the human, autonomously adopting actions that optimize the assembly success without explicit communication or human modeling.

In this context, Ramachandruni et al. [12] presented the UHTP (User-aware Hierarchical Task Planning) framework, based on HTNs, which is particularly suited for communication-free scenarios. It does not require explicit modeling of the human agent and avoids imposing predefined paths for either the human or the robot. Despite its efficiency, this approach does not address two important aspects of collaborative assembly: the possibility for the agents to perform joint actions and the handling of failures. A study addressing both joint actions and failure recovery is conducted by Giulio Giacomuzzo and Diego Romeres [13]. Their setup is the reference for this thesis. However, their approach, which employs Reinforcement Learning (RL) to solve the planning problem, is based on a complex event-triggered interaction model. Their reinforcement learning solution lacks interpretability. Moreover, while it has proved to converge to optimization-based solutions in small assembly problems, its generalization to more general and large-scale tasks is not guaranteed.

Optimization-based solutions like UHTP are more appealing as they are interpretable and are guaranteed to provide the optimal plan. For this reason, the primary goal of this thesis is the extension of the UHTP framework to make it handle joint actions and failures. Given that UHTP relies on HTNs, the secondary goal consists of the development of an automatic tool to derive HTNs from video demonstrations. In particular, the contribution is threefold:

1. We investigate and review methodologies in hierarchical task representation and planning for collaboration in assembly tasks.

2. We propose a novel algorithm to derive HTNs directly from annotated videos.
3. We present a novel task planning algorithm, which extends UHTP by adding handling of joint actions and failures.

The developed solutions are extensively evaluated on different simulated assembly scenarios, where they show improved performances compared to considered baselines.

This thesis is organized as follows. Chapter 2 contains a literature review, analyzing the current state of research on the subject, with a focus on the most commonly used methods for task representation and task planning. The core Chapters 3 and 4 discuss the developed strategy, detailing the techniques employed an the implementation details, and presenting relevant simulation experiments. Finally, a conclusion Chapter 5 highlights the key results obtained and discusses potential directions for future work.

Chapter 2

State of the Art and Background

This chapter reviews relevant literature in the field of HRC assembly. In particular, the chapter focuses on task representation solutions and planning algorithms. By examining fundamental studies and recent advancements, this review identifies key methodologies and limitations that support the development of the proposed framework by highlighting critical gaps in existing research, particularly in the areas of task flexibility, real-time adaptation and human operator constraints and communication-free environments.

2.1 Task Representation

Task representation has a fundamental role in HRC assembly as it provides a structured framework that facilitates coordination, communication, and priority execution between human and robot. In particular, hierarchical description enables the decomposition of complex assembly processes into manageable subtasks. Task representation can support dynamic replanning, allowing the system to adapt to changes or unexpected events without disrupting the workflow. It minimizes the need for explicit communication by providing a clear understanding of the task flow, reducing cognitive load for the human. Furthermore, task representation plays a critical role in optimizing task execution, as they provide the structure to define optimality conditions, according to which actions can be selected.

Task representation is crucial also to implement efficient task planning strategies as it improves decision-making processes, enabling the robot to plan actions accordingly to human behavior and tasks' status. It also provides the means to implement complex features such as joint actions and failure recovery ensuring a robust and adaptable planning

For all these reasons, the choice of a suitable task representation is crucial when addressing the collaborative assembly problem. The following methods are the most consistent and widely adopted.

2.1.1 Methods

AND/OR graphs

And/Or graphs are widely used in the field of HRC because of their suitability for solving complex tasks, breaking them down into more simple ones. As they are a state-based structure, their nodes represent states or sub-goals to be achieved, while arcs reflect the actions or transitions required to progress. There are two main types of arcs: AND and OR. AND arcs indicate that several actions must be completed simultaneously or sequentially in order to progress, while OR arcs denote the possibility of choosing between several alternatives to continue.

In HRC, And/Or graphs are useful for dividing complex tasks into manageable sub-tasks, allowing specific parts to be assigned to robots and others to humans. However, they have notable limitations: as the number of tasks increases, their size tends to grow exponentially, making them computationally expensive to process. Real-time replanning to adapt to human decisions and potentially anticipate human behavior is also challenging, as the graph typically has a fixed structure, therefore, adapting the graph to unexpected events or new conditions often requires a complete reconstruction, which limits flexibility in dynamic environments. [14] [15]

Behavior Trees

Behavior Trees (BTs) are state-based hierarchical data structures used for planning and executing complex behavior in autonomous systems, such as robots and intelligent agents. Technically, a BT is composed of nodes that can be divided into two main categories: control nodes and execution nodes. Control nodes, such as *Sequence*, *Fallback*, *Parallel*, and *Decorator*, manage the execution flow, determining the order in which child nodes are activated based on the conditions or results of other nodes. Execution nodes, on the other hand, represent specific actions or conditions that return a status of *Success*, *Failure* or *Running*, depending on the result of the operation performed.

One of the distinctive technical features of BTs is their ability to combine reactive and planned behavior. For example, a *Sequence* node will execute its children in order, moving on to the next only if the previous one succeeds, while a *Fallback* node will execute its children in sequence until one succeeds, making it ideal for handling fallback situations or multiple attempts. This modular and reusable design allows BTs to easily scale and handle both simple and complex behavior.

For this specific work BTs do not appear suitable to implement a real-time adapting framework as, like And/Or graphs, they tend to impose a predefined path on human operator. They present some limits in responding to unpredictable human input or environmental changes as they could be forced to handle extensive modifications. [16] [6]

Scheduling Algorithms

Scheduling algorithms are crucial in HRC as they enable efficient management of time and resources, ensuring that tasks are assigned and completed as efficiently as possible. These algorithms determine the sequence and priority with which tasks should be performed, taking into account constraints such as available time, processing capacity and shared resources between humans and robots. In HRC, scheduling is particularly complex as it must balance differences in skills, work rhythms, and dynamic interactions between agents.

There are different types of scheduling algorithms, each with specific characteristics. For example, static scheduling algorithms create a predetermined plan, where tasks are assigned to robots and humans in a fixed way, without changes during execution. This approach is useful in predictable environments with repetitive tasks. In contrast, dynamic scheduling algorithms allow assignments to be recalculated and readjusted during execution, in response to changes such as delays, failures, or priority changes. This approach is more suitable for flexible and complex work environments, such as those requiring frequent interactions between humans and robots.

In general, this approach considers more factors such as task deadlines, resource availability and task dependencies and is, therefore, less flexible in handling unforeseen events or dynamic task requirements. [17]

Hierarchical Task Networks

Hierarchical Task Networks are action-based tree structures used to organize the resolution of complex tasks in a hierarchy of sub-levels, each level represents a different abstraction or subdivision of the main problem. The root node corresponds to the general task to be completed, while subsequent nodes represent subtasks that become more and more specific as one moves toward the leaves of the tree. This organization facilitates workflow management in collaborative contexts, where robots and humans can be assigned to different levels of the hierarchy according to their skills and capabilities. For example, a robot could be responsible for the more detailed and repetitive subtasks at the leaf level, while humans can oversee decisions at a higher, more strategic level.

HTNs offer significant advantages in terms of modularity and adaptability, as they allow complex tasks to be divided into independent parts that can be assigned to different agents in parallel or in series. This structure also facilitates the management of dependencies and priorities between tasks, a set of requirements can be deducted for each task starting from an HTN. Consequently, HTN trees enable dynamic planning, allowing robots to update their actions according to changes in operating conditions or human progress, improving efficiency and cooperation within mixed teams.

An example of HTN tree follows. Yellow nodes represent the actions that are either executed always by the same agent or can be executed by one of them, depending on the agent's planning choices. Red nodes, conventionally called fully ordered nodes (FO-nodes), impose a specific sequential order on the actions appended beneath them. Blue nodes called partially ordered nodes (PO-nodes), represent parallelism, allowing the actions below to be executed in every order, even simultaneously.

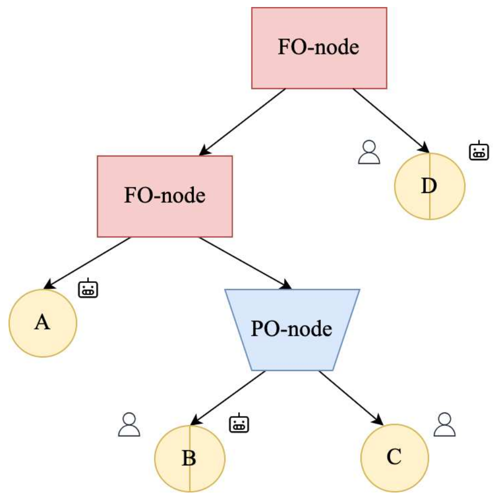

Figure 2.1: Example of HTN tree structure.

They appear to be the most suitable approach for the purposes of this thesis, as their flexibility allows them to avoid imposing constraints on the human agent. [18]

In Appendix A, the basic concepts of general tree data structures are provided to help understand their implementation.

2.1.2 Literature Review on HTN Construction from Sequences of Actions

To further support our research goals, this section explores existing methodologies for HTN construction derived from annotated video data. By examining key studies in this area, we aim to identify techniques and insights that align with our objectives, offering a foundation for applying video-based task modeling in our task planning framework.

Learning Hierarchical Task Network Domains from Partially Observed Plan Traces

Zhuo et al. [1] introduce the learning algorithm HTNLearn, which aims to autonomously create HTN planning models. The algorithm helps construct HTN methods and action models simultaneously from partially observed plan traces, significantly reducing the manual effort required to encode these models.

In a planning context using HTNs, an annotated task is defined as a triple consisting of the task itself, preconditions, and effects. The preconditions represent the conditions that must be true before the task can be performed, while the effects describe the observable changes in the state of the system once the task is completed. To decompose a complex task into simpler tasks, methods are defined. A method is characterized by a unique name, the main task it decomposes, a list of preconditions necessary for applying the method, and a list of subtasks into which the main task is divided. The preconditions must be met for the method to be used to decompose a non-primitive task into subtasks. Subtasks can be either primitive, representing executable actions, or non-primitive, requiring further decomposition.

The solution to an HTN problem results in a series of decomposition trees, each referring to a higher-level task. The root of each tree represents the main task, the inner nodes contain non-primitive tasks decomposed through methods, and the leaf nodes represent fully instantiated actions that can be executed sequentially to reach the final goal from the initial state.

Autonomously Constructing Hierarchical Task Networks for Planning and Human-Robot Collaboration

Bradley Hayes and Brian Scassellati [2] propose a method for autonomously constructing HTNs, designed to improve collaboration between humans and robots. The approach introduces "Clique/Chain HTNs" (CC-HTNs), a type of HTN based on Conjugate Task Graphs, a graphical representation that facilitates the identification of ramifications and hierarchical structures within tasks. The ultimate goal of this approach is to enable task planning in collaborative contexts.

The main implementations explored in this article are the following:

- **Clique/Chain HTN (CC-HTN):** This new type of HTN represents relationships between actions using "clique" (commutative actions) groups, where actions can be performed in varying order, and "chain" (actions with obligatory order) graphs, which represent sequences of actions that must follow a specific order.

- **Algorithm for CC-HTN Construction:** The algorithm analyzes task graphs to identify and abstract commutative and sequential subsequences, reducing the complexity of the scheduling problem and making the task representation more compact and readable.

- **Evaluation and Applications:** The paper demonstrates how CC-HTNs can be applied to state estimation and transfer learning in reinforcement learning domains, improving computational efficiency. For example, in the context of IKEA furniture assembly, the use of CC-HTNs has been shown to significantly reduce the computational time required for task planning and goal identification during collaboration.

Neural Task Programming: Learning to Generalize Across Hierarchical Tasks

Danfei Xu et al. [3] present a learning framework for robots, called Neural Task Programming (NTP), which enables generalization of complex hierarchically structured tasks. The goal is to address two fundamental challenges in advanced robotics: learning policies that can adapt to new task goals and hierarchically decomposing tasks into subtasks.

NTP is a learning algorithm based on neural networks and neural programming. It takes as input a task specification, such as a generalized video demonstration, and recursively decomposes it into subtask specifications (a time sequence that describes the process and the final goal). These subtasks are handled through a hierarchical policy, where lower-level programs represent executable subroutines that interact with the environment.

During training, NTP generates conditional policies using Few-Shot Learning from Demonstrations, enabling it to generalize to unseen tasks by adapting to variations in sequence length, task topology, and goals.

More specifically, policies are represented as neural programs that take task specifications as input, using a task-agnostic core network to decide the next sub-program and adaptively feed specific task information to each level. This allows NTP to recursively decompose tasks through a divide-and-conquer strategy.

2.2 Task Planning

Task planning is a crucial element in the field of robotics, as it allows us to model the environment in which agents, whether robots or humans, operate and to define how they can interact within that environment. In other words, task planning provides a structured framework that allows agents to identify the actions necessary to achieve certain goals and to understand how these actions affect the overall state of the system.

Task planning is structured in terms of actions and states: an agent performs an action with the objective of changing the current state of the system, progressively moving towards achieving the final goal. Actions, which may be simple or complex, are planned in an optimal sequence, taking into account initial conditions, available resources and possible interactions with other agents or the environment.

In a robotic context, task planning is not only concerned with determining which actions are to be performed, but also with predicting and managing possible uncertainties or variations in the environment, such as unexpected obstacles or changes in operating conditions. This process requires an integration of different skills, such as situation recognition, real-time decision-making, resource planning and adaptability.

A thorough study of existing literature has been conducted to deeply understand the needs of our research and choose the most suitable method to implement our task planning. Several notable case studies are highlighted below to point out the affinity with our thesis goals. These studies are divided into four subsections, each identifying the task representation method adopted.

2.2.1 And/Or Graphs

Flexible Human–Robot Cooperation Models for Assisted Shop-Floor Tasks

Darvish et al. [4] adopt And/Or graphs to implement a flexible architecture (FlexHRC) for HRC in assisted cooperation. It challenges collaboration by using wearable sensors for human action recognition and a task priority framework to separate action planning from robot motion planning and control.

FlexHRC implements action planning through a task representation module that uses AND/OR graphs to decide which action should be performed by the operator or the robot. These graphs make it possible to represent different patterns of cooperation and establish optimal action sequences based on the structure of the graph itself. The Planning module uses these graphs to suggest actions to operators, leaving them free to decide whether to follow them or not, while imposing actions on the robot.

The FlexHRC architecture places special emphasis on the flexibility and freedom of human operators, trying to minimize the constraints imposed during the cooperation process. Specifically:

- **Flexibility:** Operators are not forced to follow a predefined sequence of operations, but can decide which actions to perform on the fly, as long as they adhere to the overall goals of cooperation. Robots must balance between providing optimal suggestions and reacting appropriately when operators do not follow those suggestions.

- **Intelligibility:** Operators must be able to intuitively understand the robots' actions and intentions, achieving symbolic and linguistic communication. Collaborative robots must be able to separate action planning from motion planning and control, hiding low-level complexities.

- **Adaptability:** The system must adapt to operators without requiring an operator-specific calibration process, using wearable sensors and statistical techniques for action recognition.

- **Transparency:** Operators should not restrict their freedom of movement, such as staying in front of the robot at all times, to allow the system to monitor their actions during cooperation.

While this approach supports task planning for both the robot and human, it limits the number of cooperation paths and lacks integrated safety strategies, such as detecting sudden or unwanted contacts.

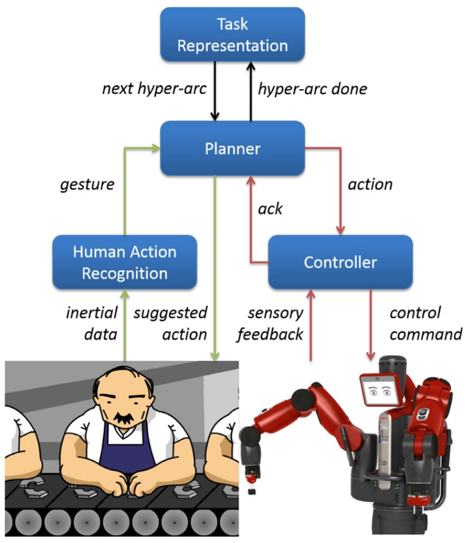

Figure 2.2: FlexHRC system's architecture: modules and data flow.

A Hierarchical Human-Robot Interaction-Planning Framework for Task Allocation in Collaborative Industrial Assembly Processes

Johannsmeier et al. [5] implement a framework using AND/OR graphs for assembly planning and processes as they implicitly model parallelism.

The described approach for task planning has three integrated hierarchical levels: team level, agent level, and real-time level.

- **Team-level:** The system plans the entire assembly process from the perspective of the team leader, using an abstract world model. This planning, performed offline, employs AND/OR graphs and A* graph-search to determine the optimal task sequences for each agent in the team, consisting of human and robotic operators. There is no distinction between the agents since this specific level conveys only abstract descriptions of tasks without the need to consider specific implementations of the necessary skills. The allocation of the optimal task planning is derived taking into account the computed cost of each agent.

- **Agent-level:** It translates the team level planning into tasks executable by individual agents. Agents, both human and robot, use modular and parametric skills, managed through hierarchical and concurrent state machines, to perform assigned tasks. This layer also deals with handling unexpected events such as collisions or interruptions by human operators, in a way that should be consistent with the entire team-level planning and when this is not possible request a rescheduling. The basic actions implemented are: pick-up, assemble, hand-over, collision handling.

- **Real-time level:** It focuses on the execution of planned trajectories, control, and immediate reactions to unexpected events. It ensures rapid and safe response to unplanned disturbances, ensuring the continuity and safety of operations.

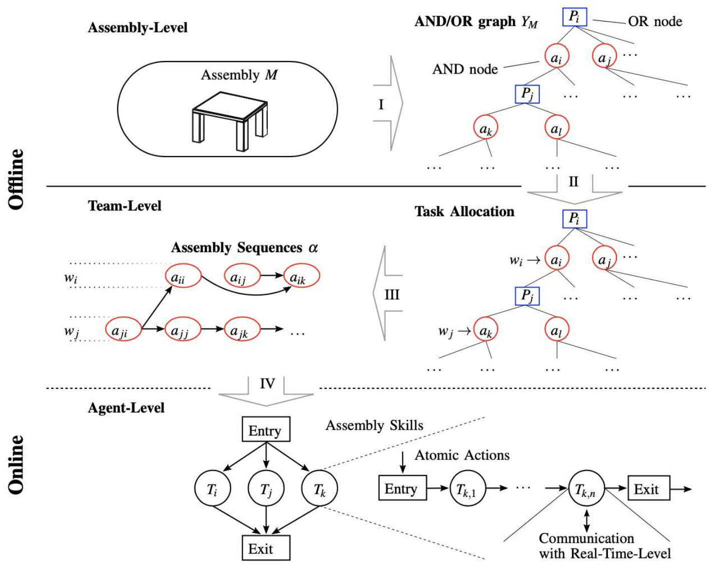

Figure 2.3: Planner workframe.

If an agent is unable to perform a specific action it is considered with infinite cost, thus affecting the optimal task allocation. Interruptions by human operators must be handled in a manner consistent with the overall plan or require offline replanning at the Team-level.

2.2.2 Behavior Trees

A Human-Aware Method to Plan Complex Cooperative and Autonomous Tasks using Behavior Trees

The paper by Fusaro et al. [6] proposes an innovative method for task planning in industrial environments involving collaboration between robots and humans. This method uses BTs modified to account for costs associated with actions and specific metrics for cooperation with human workers. Modifications include the implementation of cost functions that consider: human availability, decisions made by human workers, ergonomics, or the comfort and safety of the worker during interaction with the robot.

The BT has a directed tree structure, with a root node periodically sending signals, called “ticks,” to trigger the execution of child nodes. Each node of the tree, apart from the root, has a parent node and one or more child nodes. When a node receives a tick, it performs its task and returns a status to the parent: SUCCESS if the action is successfully completed, FAILURE if it fails, and RUNNING if it is still in progress.

This approach allows the robot to adapt its plan in real-time based on the behavior and intentions of the human operator. Specifically, the robot selects tasks by minimizing execution costs (calculated using weights defined for a particular realistic scenario through simulations). The methodology was validated with a practical experiment involving the packing of four different objects, demonstrating that cost-based BTs enable the robot to react and plan new tasks in response to dynamic changes in the environment.

In the proposed approach, the main constraints for the human operator include:

- **Availability:** The system must consider whether the operator is available or engaged in other activities.
- **Decisions:** Decisions made by the operator directly affect the robot's planning. The robot must be able to recognize and adapt to these decisions.
- **Ergonomics:** The system takes into account ergonomic conditions, such as the weight of objects to be handled and their location, to ensure that assigned tasks are safe and comfortable for the human operator.

BTs allow different levels of human-robot interaction to be handled, from simple coexistence to full cooperation and autonomous task execution by the robot. The performance of the system was evaluated in both simulations and real-world scenarios, demonstrating the robot's ability to dynamically adapt and optimize planning based on human actions.

An Integrated Dynamic Method for Allocating Roles and Planning Tasks for Mixed Human-Robot Teams

The paper [7] describes an innovative methodology for planning and allocating tasks in mixed teams of robots and humans, particularly suitable for production environments. The task planning process is based on BTs, which allow complex work to be modeled as a series of distinct tasks with time and logic constraints. The key feature of BTs is their modularity and responsiveness, which allows dynamic situations to be managed by adapting plans in real time. In the proposed context, BTs do not control the behavior of a single agent, as is traditionally the case, but rather manage the entire task, delegating tasks to the various agents (humans and robots) available. This approach allows tasks to be dynamically allocated to team members based on the costs associated with the actions and the availability of the agents.

One of the innovative aspects of the proposed methodology is the integration of task planning and role allocation. This integration is achieved through the definition of customized nodes within the BTs:

1. **Role allocation node:** This is responsible for assigning tasks to agents based on the associated costs. The costs consider various factors such as ergonomics and agent experience. This node calculates the optimal allocation by solving a linear mixed integer optimisation problem (MILP).

2. **Agent management node:** Determines which agent (human or robot) should perform the assigned task. If the task is intended for a human, the node will communicate the action to the operator via an interface. If, on the other hand, it is intended for a robot, it will activate the robot planner directly.

3. **Robot action node:** Controls the execution of the robot's movements, such as grasping or moving, according to the assigned tasks.

4. **Human communication node:** Communicates to the human operator the tasks to be performed, using devices such as monitors (smartwatches or smartglasses, depending on the working environment).

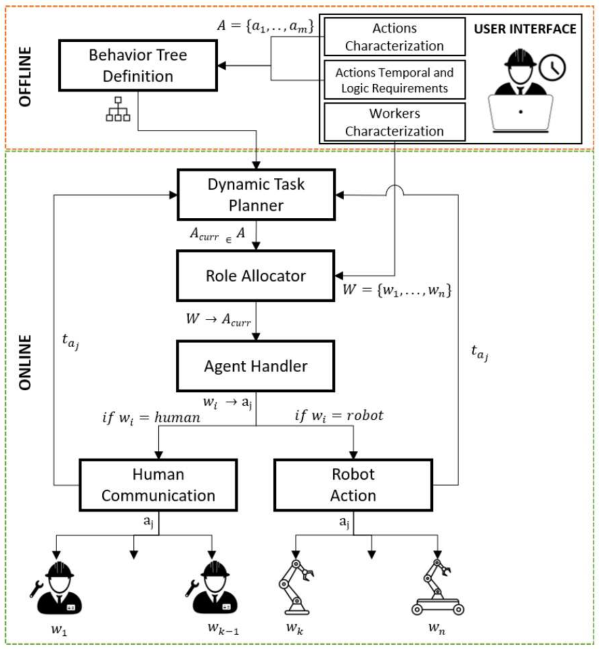

Figure 2.4: Scheme of the allocation and planning method.

The experiments conducted showed that the proposed methodology can effectively adapt to different production scenarios, optimizing task allocation and reducing waiting times. However, further studies are needed to evaluate the effectiveness of the method in real environments and to improve the user interface for task generation.

2.2.3 Scheduling Algorithms

Improved Human-Robot Team Performance Using Chaski, A Human-Inspired Plan Execution System

The Chaski algorithm [8] is designed to improve the performance of human-robot teams by emulating the effective behaviors of human teams. This algorithm allows robots to dynamically adapt to human partners, reducing human downtime and making interactions more natural and fluid.

The main technical features of Chaski algorithm are here listed:

1. **Task level executive:** Chaski acts as an executive that plans and schedules the robot's actions based on a shared plan among team members. It uses the task-based hierarchical planning model, HTN, to manage tasks.
2. **Dynamic decision-making:** The algorithm enables robots to make decisions in real time as it supports the interruption and resumption of tasks, adapting to changes in context and the actions of human partners.
3. **Minimization of downtime:** Chaski predicts and minimizes human downtime by optimizing the sequence of tasks to ensure smooth and continuous collaboration. It implements continuous monitoring mechanisms to detect when a team member is inactive and to redistribute tasks accordingly.

Human constraints imposed by the algorithm:

- **Equal partners:** In this model, each team member, both human and robotic, has equal authority in making decisions regarding the execution of the plan. Humans must be prepared to share control of activities and coordinate closely with robots.
- **Choice of activities:** Humans have the freedom to choose their actions within the constraints of the shared plan. They must manage the timing of their activities so as not to interfere with the robots' operations, maintaining a balance between autonomy and coordination.
- **Information sharing:** Operators have a shared mental model and the human is expected to regularly communicate the status of activities, sending immediate feedback on task performance and reporting any problems or changes in the operational environment. This ensures that robots can adapt quickly and that interaction remains smooth and efficient.

The framework presented requires communication between agents and the human is supposed to adapt to the robot's choices.

Optimization of Temporal Dynamics for Adaptive Human-Robot Interaction in Assembly Manufacturing

Wilcox et al. [9] address the optimization of human-robot collaboration in assembly manufacturing, focusing on two main challenges: adapting to varying human preferences and meeting rigid temporal constraints.

The Adaptive Preferences Algorithm (APA) is presented as an algorithm for optimizing time scheduling in a flexible and robust manner. It allows robot scheduling to be adapted to the evolving preferences of human operator, while maintaining synchronization and adherence to task schedules.

To achieve the prefixed goal, the algorithm requires some constraints:

1. **Variable preference:** Human workers can change the order and time of task completion. The APA algorithm must therefore be highly adaptable to respond to these changes.

2. **Strict scheduling constraints:** Despite adaptability, the system must ensure that tasks are completed within predetermined timescales, such as flow rates between assembly stations and production deadlines.

3. **Robustness and flexibility:** The algorithm must be able to adapt to time disturbances without requiring a complete re-computation of the schedule, thus providing a robust solution to variations.

APA represents a significant step forward in human-robot collaboration, providing adaptable temporal scheduling that responds quickly to human preferences and operational disturbances, while respecting the essential temporal constraints of the production process.

Here are some application examples:

- **Robotic Assistant for Assembly Mechanic:** A robot such as FRIDA can assist a mechanic in preparing and handling components, allowing the mechanic to focus on more critical tasks such as aligning and securing parts.

- **Orchestration of Robot Teams:** A single operator can lead a team of robots, ensuring that scheduling deadlines are met and work is redistributed quickly in the event of interruptions.

2.2.4 Hierarchical Task Networks

Using Human Knowledge Awareness to Adapt Collaborative Plan Generation, Explanation and Monitoring

Milliez et al. [10] implement task planning using HTNs for its domain representation and efficient planning. This structure facilitates the monitoring of agents' task execution and subsequently collaboration between them.

The main aspects of the adaptive planner for the robot are:

1. **Agent based:** The system computes multi-agent plans with actions of both humans and robots.
2. **Cost-driven:** The best or "good enough" plan is found faster by pruning plans.
3. **Social rules:** Plans are refined according to a set of rules to promote socially acceptable plans, such as balancing efforts according to human preferences and context.

Each action in the domain provides a function that estimates the cost when added to the plan. During plan construction, the cost of the partial plan is continuously calculated and compared with the current best score, discarding plans that exceed that cost. In addition, after plan generation, filtering rules are applied to ensure that plans meet certain social behaviors. Once the best plan is retrieved, it is sent to the supervisor in the form of HTN tree decomposition and action streams are processed for each agent (human or robot).

Human operator is modeled through different levels of task knowledge, which influence the robot's behavior during the generation and execution of the collaborative plan. The defined knowledge levels are:

- **NEW:** Used for tasks never performed by the user. If the user observes the task with explanation or performs it, the level changes to BEGINNER.
- **BEGINNER:** For users who have already performed the task but may need explanation. If the user successfully performs the task without asking for explanation, the level changes to INTERMEDIATE.
- **INTERMEDIATE:** For users who can perform the task without guidance. If the user successfully performs again, the level changes to EXPERT.
- **EXPERT:** For users who can perform the task without guidance and explain to others. If it fails, it is downgraded to INTERMEDIATE.

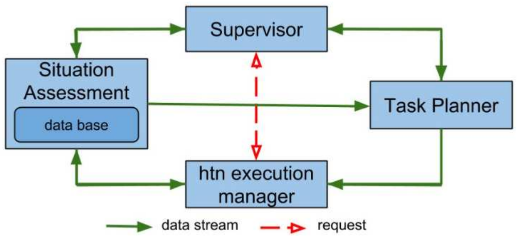

Figure 2.5: System architecture.

These levels allow adaptive collaborative plan generation, explanation and monitoring.

Adaptive plan execution involves a plan management algorithm that considers the user's knowledge level and interaction policy (teaching or efficiency). The robot not only performs and monitors actions, but also teaches the user when necessary. Examples of specific constraints include: if the agent is only the robot and the user is at NEW level, the robot will execute and explain actions; if the user is BEGINNER, the robot will propose explanations and monitor execution, for INTERMEDIATE and EXPERT levels, the robot mainly monitors, adapting assistance as needed.

In summary, the system implements adaptive planning that considers the user's knowledge and interacts dynamically to optimize task execution, ensuring that human preferences and constraints are respected throughout the process. Plans are dynamically adapted to respond to changes in context and to handle any failures of actions.

Transparent Role Assignment and Task Allocation in Human Robot Collaboration

The paper by Roncone et al. [11] discusses how robots can be qualified collaborators able to reduce cognitive load for the human partner. This is achieved with the employment of HTNs that derive optimal robotic planners, predicting also possible human actions. Markov Decision Processes are extended to Partially Observable Markov Decision Processes (POMDPs) for the purpose of this research, they address interaction uncertainties as non-observable, covering situations such as misalignment of task progress and environmental changes that are non-observable by the robot. In addition to the sequential and parallel operators typical of the HTN representation, we introduce the alternative operator. This operator allows subtasks to be disjointed, greatly improving the expressiveness of the network.

The proposed framework implements a model based on human-robot communication and focuses on two main aspects:

1. **Role Assignment:** The system is able to determine the roles of each participant, based on specific capabilities and task requirements. Moreover it uses shared mental models to ensure that both participants (human and robot) understand each other's roles and responsibilities.

2. **Task Allocation:** Tasks are divided into subtasks that can be executed in parallel, reducing cognitive overhead for the human operator. The system adopts an intuitive interface and high-level commands, making it accessible even to inexperienced users.

The system is transparent and designed to reduce the cognitive load on the human operator, allowing him to focus on tasks that require dexterity and superior skills, in fact the goal is to minimize the need for continuous communication, which can be expensive and ineffective in noisy environments. The robot often cannot fully observe and understand the state of the world, especially in unexpected or unplanned situations, therefore in some specif cases communication is used to reduce uncertainty.

Human-Robot Collaboration for Assembling IKEA Furniture through Reinforcement Learning-Based Planning

The task planning designed in the paper by Giulio Giacomuzzo and Diego Romeres manages complexities such as joint actions, failure recovery and change of mind. [13] The main approach used for the design of the task planning focuses on the concept of Discrete Event Markov Decision Process (DE-MDP). This approach integrates hierarchical task representation and advanced solution methods to facilitate efficient HRC in assembly tasks.

The process relies on avoiding human behavior modeling, the human agent is considered an uncontrollable agent. In this context, the robot does not directly control the human actions but must observe and respond to them. Human actions are detected by the robot after a certain latency period, which introduces an element of uncertainty and necessitates real-time adaptability in the robot's planning. This setup essentially transforms the collaborative task planning problem into a single-agent problem from the robot's perspective, focusing on how the robot reacts and adapts to human actions.

The assembly task uses HTNs for representation and is then modeled as a Discrete Event Markov Decision Process (DE-MDP), in which state transitions are determined by significant events such as completion of an action, detection of human actions, or changes in human intent. This model takes into account the asynchronous nature and variable duration of tasks, providing a flexible and dynamic framework.

To solve DE-MDP, the approach considers two methods: a deterministic decision graph and Reinforcement Learning (RL). RL is particularly valued for its ability to find optimal policies in both deterministic and stochastic contexts, making it suitable for handling variability in human actions and work environments.

Several challenges are addressed within this framework:

- **Synchronization:** Ensuring that both human and robot agents can perform joint actions despite differing action durations.

- **Human Detection:** Adapting robot actions based on observed human actions, which introduces scheduling delays.
- **Human Change of Mind:** The robot must be capable of rescheduling its plans if the human changes tasks mid-way.

By integrating these elements, the approach creates a robust framework for collaborative assembly, emphasizing the robot's ability to dynamically adjust to human behavior and environmental changes. This dynamic adaptation is crucial for effective collaboration, optimizing task execution, and enhancing overall system performance.

UHTP: A User-Aware Hierarchical Task Planning Framework for Communication-Free, Mutually-Adaptive Human-Robot Collaboration

Ramachandruni et al. [12] in this paper present a framework called UHTP (User-aware Hierarchical Task Planning), designed for adaptive collaboration between humans and robots without the need for explicit communication. This approach overcomes the traditional 'leader-follower' paradigm, in which one agent (human or robot) follows the actions established by the other. UHTP allows the robot to observe human actions, perform tasks that support human decisions and select actions that maximize the overall efficiency of the collaboration.

UHTP is based on HTNs that represent both human and robot actions. The process is structured as follows:

1. **Extension of the HTN:** The traditional representation of the HTN is extended to include the assignment of actions to agents (human or robot) and their action costs.
2. **Action Assignment:** Actions are assigned to agents based on their capabilities. If an action can be performed by both, it is converted into a decision node with two children, one for each agent.
3. **Cost Calculation:** The total cost of executing actions is calculated, considering both agent-specific and cumulative costs, to identify the optimal execution path.
4. **Online Planning:** During task execution, UHTP continuously monitors the current state of the task through real-time human action recognition and selects robot actions that minimize the total expected cost.

One of the main advantages of the UHTP framework is that it does not constrain the human operator to follow a predefined plan or to communicate explicitly with the robot. The human retains the freedom to choose his actions based on observation of the task and the robot, without being constrained by a planner or the robot's actions. This approach reduces the cognitive load on the human and improves the overall efficiency of the collaborative task.

Chapter 3

HTN Construction From Annotated Videos

Through a deep study and analysis of the State of the Art reported in Chapter 2, including the most common approaches for achieving goals similar to those of this thesis, HTNs emerged as the most suitable approach for task representation. This methodology offers intuitive representation, low computational costs for online replanning, and an action-based framework that facilitates robot policies based on cost accumulation.

Among the reviewed approaches, the framework *UHTP: A User-Aware Hierarchical Task Planning Framework for Communication-Free, Mutually-Adaptive Human-Robot Collaboration* [12] stands out as the most aligned with the established requirements for task planning implementation using HTNs. UHTP imposes no constraints on the human agent, supports seamless online replanning, and will be reinterpreted and extended with additional functionalities in the next chapter.

The applicability of various algorithms to HTNs reveals that a significant challenge lies in constructing the HTN tree. Therefore, this thesis extends its focus to the exploration of an algorithm designed to automatically generate an HTN tree. Manually generating an HTN from a series of actions, each with its own requirements, can be time-consuming and not always intuitive. Human error must also be taken into account. The complexity of this framework is further increased by the flexibility of the collaborative assembly setups considered in this work. There are no single, fixed sequences of tasks as agents can exchange actions, and multiple paths can be taken to achieve the same goals.

One approach to deal with this complexity could be Learning from Demonstration (LfD), a widely used method for robot training and policy learning. Rather than simply teaching robots specific instructions, many researchers aim to use LfD to impart more generalized knowledge that increases the robot's adaptability. This approach also reduces the amount of data required for training, enabling more efficient learning processes and extending the robot's ability to perform flexible task sequences within the HTN framework. [19]

The study and development of this approach are left to other works; the foundation of this Chapter will be on the assumption that a dataset is already available and the goal is to create the HTN by analyzing the relationships between actions and understanding how agents collaborate within these tasks.

3.1 HTN Construction: Algorithm Implementation

The approach used to implement the algorithm involves two main methods: one that extracts action requirements from a dataset and another that constructs the HTN tree structure for the UHTP framework.

The dataset consists of annotated videos structured as a list of tuples, each tuple contains the human action and the robot action being executed simultaneously. Each action is given as a list that contains: the name of the action, its duration and the agent performing it. If either agent is not executing an action, it is labeled as *idle*. The succession of tuples is not determined by a time variable but instead by the transitions between actions, a new tuple is recorded whenever the human or robot changes their action, including transitions to or from the *idle* action. If the tuple consists of the same action for both the human and the robot, it indicates that they are performing a joint action.

An example of tuple is reported below, multiple tuples expressed in this form represent an annotated video.

```
(["action 1",[1],["human"]] , ["action 2",[1],["robot"]])
```

The requirements of an action are the set of actions that must be completed in advance to achieve a specific task goal. Task descriptions based on action requirements offer an alternative approach to HTNs. While this approach might be useful for certain implementation strategies, its main limitation is the absence of hierarchical structure typical of HTNs, which is essential for task planning. Therefore, a method that converts one representation into the other is necessary.

The first method, that extracts action requirements, processes all the annotated videos one by one. For each video, a list of executed actions is maintained to track the actions that have been completed. When a new action occurs, whether performed by the human or the robot, its list of requirements is updated to include all the actions in the executed actions list. Since there are multiple ways to achieve the same goal and each annotated video reports a different sequence, the requirements for an action must always be a subset of the completed actions as the videos progress. If any requirement does not meet this condition, it is removed from the list of requirements.

Algorithm 1 Extract actions' requirements from annotated videos

```
1: function extract_htn_requirements(annotated_videos)
2:     action_list ← []
3:     for each action_sequence in annotated_videos do
4:         executed_actions ← []
5:         for each (human_action, robot_action) in action_sequence do
6:             append executed actions to current_human_action and current_robot_action
7:             if human_action equals robot_action then
8:                 current_joint_action ← human_action
9:             else
10:                current_joint_action ← ”idle”
11:            end if
12:        end for
13:        if current_action is not ”idle” then
14:            for each action in action_list do
15:                if action equals current_action then
16:                    update action[3] based on executed_actions
17:                end if
18:            end for
19:            if action not found then
20:                append current_action to action_list
21:            end if
22:        end if
23:    end for
24:    return action_list
25: end function
```

The second method, that creates an HTN tree given a set of requirements, is found on the structure presented by UHTP framework.

The HTN is constructed starting from the leaves. It iterates through the list of actions and progressively appends the actions whose requirements are contained within the past actions list. The past actions list maintains a record of all actions that have already been appended to a node in the tree. If an action can be performed by only one agent, a single action node is created. Otherwise, if the action can be performed by multiple agents, an action node is created for each agent and these nodes are appended to a decision node (this type of node belongs to UHTP implementation and will be better introduced in the next Chapter 4). Partially ordered nodes are added to the tree to introduce parallelism between actions, while fully ordered nodes establish sequential relationships. When multiple actions share the same set of requirements, they can be added as children of a partially ordered node. Fully ordered nodes are used when there are multiple actions that need to be performed in a specific order. When multiple actions share the same set of requirements, they can be appended as children of a partially ordered node, allowing for parallel execution. Fully ordered nodes, on the other hand, are used when multiple actions must be performed in a specific sequence. Additionally, each node appended to a fully ordered node must have all its action requirements already appended to the left of the same parent node, ensuring that dependencies are respected.

Once all action requirements are identified, an HTN can be constructed and this concept leads to the following theorem:

Theorem 1 *Given a dataset of actions, each with its own set of requirements, one or multiple Hierarchical Task Networks (HTNs) can be constructed. Vice versa, from a given HTN tree structure, a unique set of action requirements can be extracted.*

Proof: The proof is algorithmic.

For any given set of actions $A = \{a_1, a_2, \ldots, a_n\}$, each associated with a set of requirements $R(a_i)$, it is always possible to derive a correspondent HTN structure. This is because the requirements of an action a_i that depend on another action a_j implicitly include all of a_j's requirements, even if not explicitly stated. This dependency induces a hierarchical structure between actions.

While the requirements of actions uniquely identify the relationships between them, the hierarchical structure that defines the HTN is not necessarily unique. Multiple HTN trees can be derived depending on how task-subtask relationships are conventionally defined.

For instance:

- Fully ordered (FO) nodes can represent the hierarchy imposing sets of requirements on the actions
- Partially ordered (PO) nodes can organize the hierarchy into subsets of tasks
- Primitive nodes represent the tasks as leaves

Conversely, given an HTN tree structure $\mathcal{T}$, the unique set of requirements for each action can be derived traversing the tree. Any node whose parent is a fully ordered node inherits as requirements all the actions contained in the left subtree(s) of the same FO node, if such a subtree exists.

Algorithm 2 Generate HTN from action list

```
function generate_htn(action_list)
    action_sequence ← action_list
    past_actions ← []
    next_node_children ← []
    function add_action(parent_node, child)
        if len(child[2]) == 1 then
            create primitive action node
            add node to parent_node
        else
            create decision node
            create and add action nodes for 'human' and 'robot'
        end if
    end function
    while action_sequence is not empty do
        children ← []
        for each action in action_sequence do
            if all requirements of action are in past_actions then
                append action to children
                remove action from action_sequence
            end if
        end for
        if len(children) > 1 then
            create PO node
            for each child in children do
                add action to PO node using add_action function
            end for
            append PO node to next_node_children
        else
            append children to next_node_children
        end if
        if len(next_node_children) > 1 then
            create FO node
            for each child in next_node_children do
                add child to FO node
            end for
            clear next_node_children
            append FO node to next_node_children
        end if
    end while
    create HTN tree with root as next_node_children[0]
    return htn_tree
end function
```

3.2 Experiments

Example 1

Based on the chair assembly example from the UHTP paper [12], a sequence of annotated videos is used to determine the action requirements and construct the HTN. For the purposes of this example, all of the possible combinations of action sequences are given, implying the full knowledge of the actual requirements.

An annotated video in this example is reported below.

```
(["attach left leg",    [3],["human"]], ["attach right leg",[2],["robot"]]),
(["attach left leg",    [3],["human"]], ["attach back",     [5],["robot"]]),
(["flip seat",          [2],["human"]], ["attach back",     [5],["robot"]]),
(["attach back to seat",[5],["human"]], ["idle",            [5],["robot"]])
```

The method `get_htn_requirements` returns the various actions, each with its own requirements reported as a list included at the end of each action's description.

```
["attach left leg",    [3,2], ["human","robot"], []],
["attach right leg",   [3,2], ["human","robot"], []],
["attach back",        [5,5], ["human","robot"], []],
["flip seat",          [2],   ["human"],         ["attach right leg",
                                                 "attach left leg"]],
["attach back to seat",[5,7], ["human","robot"], ["attach right leg",
                                                  "attach left leg",
                                                  "attach back",
                                                  "flip seat"]]
```

Given this set of requirements, the following HTN structure is obtained through `generate_htn` algorithm. Traversing the tree in postorder from each leaf confirms that all requirements are satisfied. For instance, `attach left leg`, `attach right leg`, and `attach back` have no requirements, as they are children of a PO node. Conversely, `flip seat` is appended to a FO node and inherits as requirements all actions located in the left subtree of the same parent node, specifically `attach left leg` and `attach right leg`. Similarly, `attach back to seat` is the rightmost child appended to the FO node, which serves as the root of the tree. As a result, it inherits as requirements all the actions appended to the left of this node.

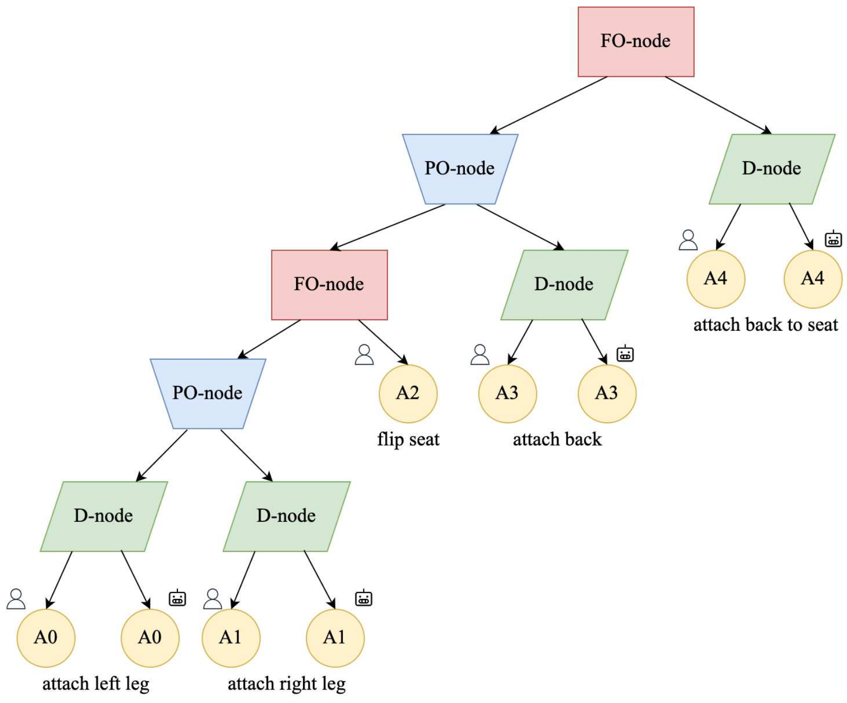

Figure 3.1: HTN chair assembly exemple.

Example 2

Below is reported another example with a larger set of actions to validate the illustrated methods for HTN construction. Firstly all actions are reported with their requirements (obtained through the execution of `get_htn_requirements`).

```
["action 0", [8,16], ["human", "robot"], []],
["action 1", [16,8], ["human", "robot"], []],
["action 2", [16,8], ["human", "robot"], [0,1]],
["action 3", [16,8], ["human", "robot"], [0,1,2]],
["action 4", [16,8], ["human", "robot"], [0,1,2,3]],
["action 5", [16,8], ["human", "robot"], [0,1,2,3]],
["action 6", [16],   ["joint"],          [0,1,2,3]],
["action 7", [16],   ["joint"],          [0,1,2,3]],
```

The resulted HTN after the execution of `generate_htn` has the following structure. Applying the same reasoning as in the previous example confirms that all requirements are satisfied within the generated HTN.

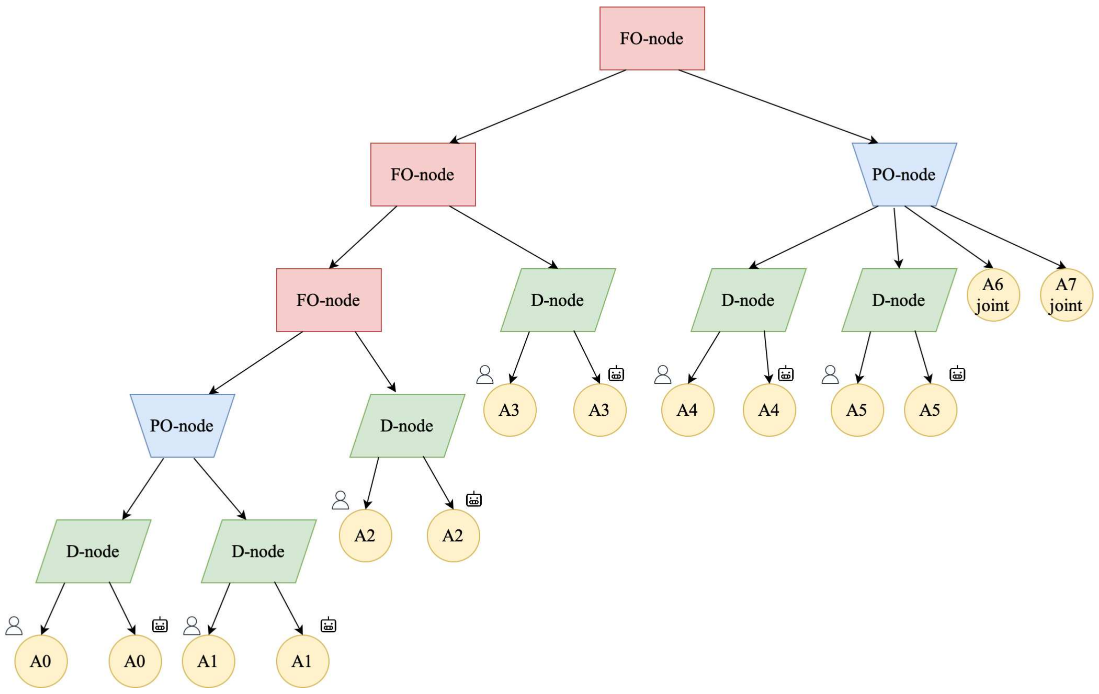

Figure 3.2: HTN random example.

After testing the two implemented methods with various setups, it can be concluded that the constructed HTNs satisfy the requirements of each action and provide a viable solution. The `generate_htn` method could be further modified to produce alternative tree structures. For example, in the second example, the three concatenated FO nodes could be merged into a single FO node, with their children appended in hierarchical order to preserve their requirements. This reasoning strongly aligns with and supports the theorem stated earlier (1).

Chapter 4

UHTP Implementation

The implementation of task planning is based on the UHTP framework presented in [12], as it meets the key requirements of this thesis. These include the absence of communication between agents, support for online replanning, no constraints imposed on the human agent, and most importantly the adoption of HTNs. Additionally, the framework demonstrates efficient performances.

Below, a reinterpretation of the algorithm is provided, extended to incorporate features not yet implemented such as management of joint actions, actions performed simultaneously by both agents and, recovery plan for failed actions. Building on the pseudocode from the paper, which outlines the program flow, the algorithm is fully deduced and implemented.

4.1 Framework Implementation

A notable aspect considered is the classification of the node types that populate the HTN: to adhere to the structure of the original algorithm, the approach is to implement a `Node` class with several subclasses, each representing a specific type of node. Each subclass defines a unique method for computing the cost of the subtree starting from that particular node. In particular, UHTP introduces another type of node, called decision node, which extends the standard HTN classification discussed earlier.

Here follows the definition of the different types of nodes:

- **Primitive nodes:** which represent the actions performed by either the human or robot agent. As the leaves of the tree, they represent the simplest subtasks, with their cost being intrinsic to the action itself.

- **Decision nodes:** introduced when an action can be chosen by multiple agents as it is followed by as many primitive nodes as there are agents capable of executing that action. Its cost is computed as the sum of the children's costs, weighted by their respective probabilities.

- **Fully ordered nodes:** indicating actions that must be executed sequentially, in a specific order. The total cost is the sum of the costs of the child nodes.
- **Partially ordered nodes:** which allow actions to be performed in parallel, meaning several tasks can be performed simultaneously. The cost is calculated by comparing the upper and lower bounds: the upper bound is the sum of the children's costs, while the lower bound is the maximum of these costs.

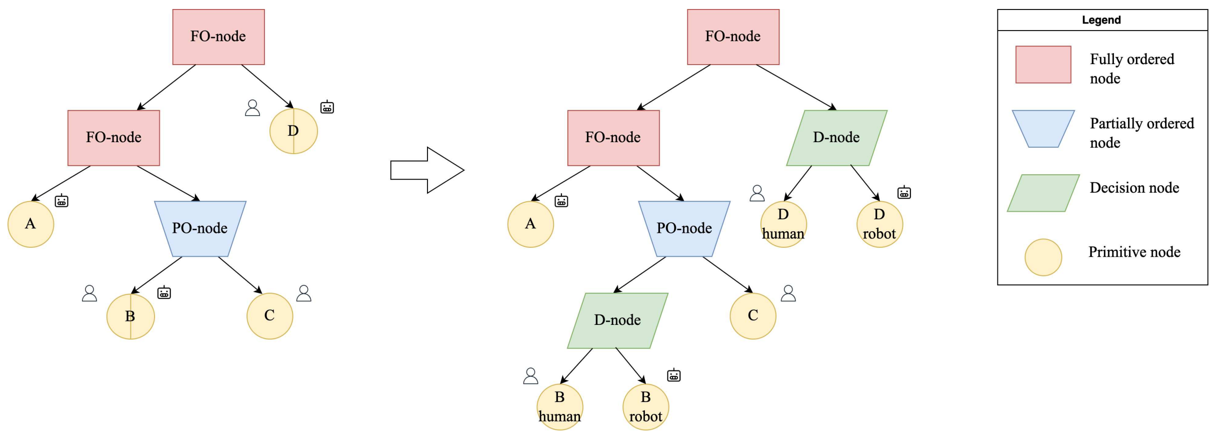

Figure 4.1: UHTP tree example.

Each node object contains a `children` attribute, which is a list of its child nodes. A notable feature is that primitive nodes, being leaf nodes, cannot have any children.

All subclasses of the `Node` class share three common parameters: `name`, `type`, and `parent` (which refers to the preceding node in the structure). Additionally, the `Decision` node class includes a `probability` parameter, representing the likelihood of each agent executing the subsequent action. This parameter is useful for calculating the aggregate cost associated with the `Decision` node.

The `Primitive` node class includes additional parameters: `index`, `cost`, `duration`, and `agent`, which further define the specifics of the task it represents.

The `HTN` class is also defined, where its objects are initialized using the root node of the tree as a parameter. Since each node maintains its own list of children, passing only the root node implicitly passes the entire tree structure and its nodes. The attributes of this class are as follows:

- **`current_human_action`**, **`current_robot_action`**: Initialized as `None`, representing the current actions being executed by the human and the robot;

Algorithm 3 Pseudocode of UHTP program flow

```
function UHTP($\mathcal{T}, A_h, A_r$)
    $\mathcal{T}^{UHTP} \leftarrow$ AssignActions($\mathcal{T}, A_h, A_r$)
    AggregateCosts($\mathcal{T}^{UHTP}$)
    $a_h \leftarrow \emptyset$
    $a_r \leftarrow \emptyset$
    while $\mathcal{T}^{UHTP}$ is not empty do
        $a'_h \leftarrow$ ActivityRecognition$()$
        if $a'_h \neq a_h$ then
            $\mathcal{T}^{UHTP}$.Remove$(a_h)$
            PruneBranches(Root($\mathcal{T}^{UHTP}$), $a'_h$, human)
            $a_h \leftarrow a'_h$
        end if
        if robot is idle then
            $\mathcal{T}^{UHTP}$.Remove$(a_r)$
            $a_r \leftarrow$ SelectMinAction($\mathcal{T}^{UHTP}$, robot)
            PruneBranches(Root($\mathcal{T}^{UHTP}$), $a_r$, robot)
            Execute($a_r$)
        end if
    end while
end function
```

- `Ah`, `Ar`: Empty lists, which will contain the actions assigned to the human and the robot, respectively;

- `executed_Ah`, `executed_Ar`: Empty lists that will store the completed actions of the human and robot;

- `robot_idle`: Boolean attribute indicating whether the robot is *idle* or currently performing an action.

The following methods are all implemented inside `HTN` class:

- `AssignActions:` Initializes the lists `Ah`, `Ar`.

- `AggregateCosts:` Performs a postorder traversal of the tree to compute and aggregate costs. It ensures that the costs of child nodes are calculated and aggregated before computing the cost of their parent node.

- `ActivityRecognition:` Simulates the activity recognition process. While in a real-life context this would involve sensors or vision-based systems, for the purposes of this thesis it is simplified to a human selecting the first available action without considering costs.

- **`PruneBranches`:** Removes irrelevant branches from the tree based on the given action and agent to maintain consistency. When it encounters a decision node, it preserves branches where the action is valid and discards others. For fully ordered nodes, it recursively calls `PruneBranches` on only the first child (since actions are sequential). For partially ordered nodes, it recursively calls `PruneBranches` on all child nodes (since actions can be parallel).

- **`SelectMinAction`:** Identifies the most cost-efficient action for the robot agent by evaluating all possible actions at a given node. It first checks if the action's requirements are met (i.e. if the required tasks are already completed). For each valid action, it creates a copy of the tree, prunes irrelevant branches, and calculates the total cost of the remaining subtree. It then selects the action with the minimum cost. If no valid action is found, the method returns `None`, indicating the robot is *idle*.

- **`Execute`:** In simulation context this method simply removes the executed actions form the lists `Ah`, `Ar` and appends them to `executed_Ah`, `executed_Ar`, changing the state of the robot to *idle*.

Ultimately, the method **`UHTP_main`** emulates the workflow presented in pseudocode with the insertion of time tracking and some rearrangements in the logical transitions. One of the challenges is testing the code through simulation. To address this, the human agent is also simulated, and his decisions are modeled using the `ActivityRecognition` method. The agent's choices are guided by the requirements of each action, these requirements consist of other actions that must be completed in a specific order to accomplish assembly tasks. Once all prerequisite actions for a particular task have been fulfilled, the agent can select that task for execution. The method `get_requirements` is implemented to follow the logic explained above.

New methods have been introduced in the `HTN` class to enhance the logic of `UHTP_main` and improve the overall flow of the method.

The `UHTP_main` process is organized around two primary methods: `execute_action` and `recognize_and_prune`. The `execute_action` method is triggered when an action's duration reaches zero, indicating its completion. This method calls the `HTN` method `execute` to finalize the action, removes the action from the HTN tree, and then calls `removeNodes` to prune any nodes that lack children with primitive actions.

The second method, `recognize_and_prune`, handles new action assignments. It calls either `ActivityRecognition` if the human has completed the last action, or `SelectMinAction` if the robot has. When a new action is assigned, the method `PruneBranches` is called to refine the task tree, and the `duration` variable is updated to reflect the time needed for the new action.

Algorithm 4 Extract Requirements from Node to Root

```
function get_requirements(node)
    requirements ← []
    curr_node ← node
    while curr_node is not root do
        curr_parent ← curr_node.parent
        if curr_parent.type equals ”Fully ordered” then
            node_idx ← 0
            for each (idx, child) in enumerate(curr_parent.get_children()) do
                if child equals curr_node then
                    node_idx ← idx
                end if
            end for
            nodes_tbc_list ← [curr_parent.get_children()[i] for i in range(0, node_idx)]
            for each check_node in nodes_tbc_list do
                actions ← getNodePrimitiveActions(check_node)
                for each action in actions do
                    append action to requirements
                end for
            end for
        end if
        curr_node ← curr_parent
    end while
    return requirements
end function
```

To properly simulate the sequence of action execution, tracking time is essential. Each action, whether assigned to the robot or the human, has an associated `duration` variable indicating its required completion time. When an action's duration reaches zero, it signals that the action has been fully executed, enabling the agent to select or receive a new action.

Upon assignment of a new action, the `duration` variable is reset. During each iteration of the while loop, a time counter increments by the minimum of the remaining action durations. Each action's duration is then reduced by this minimum value, effectively progressing the time. At the end of each loop iteration, the maximum duration of completed actions is added to the cumulative time variable, ensuring an accurate track of the execution sequence.

4.1.1 Failure and Recovery of Actions

An essential feature implemented in the UHTP algorithm is the management of failed actions. Failures can arise in a variety of scenarios, each of which may disrupt the task sequence. For instance, a failure could occur if an agent drops a part during assembly, incorrectly aligns or assembles a component, or encounters unexpected resistance in a task step. In these cases, the agent's progress is temporarily halted, necessitating corrective actions to restore the proper sequence of tasks.

To address such failures within this framework, the approach taken in this thesis is to introduce a recovery mechanism in the HTN. Specifically, when an action fails, a designated recovery node is added to the HTN. This recovery action does not need to be executed immediately after the failure; agents may perform other actions first, depending on the requirements and priorities of the task under consideration. Once the recovery is successfully completed, the recovery action is removed from the HTN, and the initially failed action(s) is reinserted, allowing the agent to attempt it again in the correct sequence.

A new class of recovery actions, inheriting from the primary `Node` class, has been developed to facilitate this functionality. This recovery action class includes the following parameters: `index, name, type, parent, cost, duration, agent`

Additionally, the recovery action class includes an attribute list that stores references to the failed action(s), ensuring that all necessary corrections are properly tracked and can be reattempted in the correct sequence after recovery.

This design assumes that recovery actions can be performed by the same agents responsible for the failed actions, facilitating a straightforward corrective workflow. Through this mechanism, the HTN framework not only manages task successions but also handles unexpected disruptions, making the algorithm more robust and adaptable to real-world execution variability.

A method of the class `HTN` manages the whole recovery process explained above.

The recovery procedure is integrated into `UHTP_main` through a method called `handle_failure`. This method triggers `recovery` and resets the duration of the agent whose action encountered the failure, ensuring a smooth continuation of the task.

4.1.2 Joint Actions

Another notable improvement introduced to the UHTP algorithm is the implementation of joint actions, which require coordinated efforts from both human and robotic agents to accomplish a given task. In this framework, joint actions are designed to facilitate collaboration by taking advantage of the unique capabilities of each agent. The underlying logic ensures that the human agent initiates a joint action by selecting it, taking into account the specific requirements associated with that action. If the robotic agent is currently idle, it immediately joins the human to execute the joint task. Otherwise, if the robot is already engaged in another task, it is programmed to prioritize completion of its current action before joining the human in the joint action.

To incorporate joint actions into UHTP, they were added only to the human's set of actions

Algorithm 5 Recovery Process for Failed Action Nodes

```
1: function recovery(node)
2:     failed_action ← node
3:     parent ← failed_action.get_parent()
4:     recovery_action_list ← failed_action.recover
5:     agent_dict ← {"human": (self.Ah, self.executed_Ar), "robot": (self.Ar, self.executed_Ah)}
6:     for each recovery_action in recovery_action_list do
7:         append failed_action to recovery_action.post_recover
8:         add recovery_action as child of parent
9:     end for
10:    action_list, _ ← agent_dict[recovery_action.agent]
11:    append recovery_action to action_list
12:    failed_agent_list, executed_list ← agent_dict[failed_action.agent]
13:    remove failed_action from failed_agent_list
14:    for each action in executed_list do
15:        if action.name[5:] equals failed_action.name[5:] then
16:            remove action from executed_list
17:            append action to recovery_action.post_recover
18:        end if
19:    end for
20:    self.find_and_remove(failed_action)
21: end function
```

(`Ah`), as only the human agent can initiate a joint task. Two new attributes were introduced to the `HTN` class: `joint`, a Boolean indicating whether a joint action has been selected, and `current_joint_action`, which stores the joint action identified by the human. When the robot agent is idle and able to assist, both the human's and robot's current actions are updated to the joint action, ensuring synchronized durations for both agents.

Also joint actions can fail and when it happens the same procedure outlined above is applied, in these situations both the human or the robot are required to execute the recovery action.

4.2 Experiments

4.2.1 Failure Recovery Experiments

Failure recovery process is tested using examples to evaluate its robustness.

For simulation experiments a function `action_has_failed` is implemented. It takes as input an action node and generates a random floating-point number between 0 and 1 using `random.random()` which is confronted with a decimal number that indicates the failure probability. The function returns a boolean value `True` if the action has failed. To validate that the whole task planning can handle failures, few examples were tested with an increasing probability of failure for each action.

Example 1

Firstly the simple chair assembly example illustrated in UHTP paper is tested. Its HTN structure and task actions are already reported in the first example of the previous Chapter 3.1.

The plot shows the distribution of task completion times (in seconds). The experiment starts with a failure probability of 10% per action and increases in 10% increments up to 50%. For each failure probability, the task was executed 100 times, and the completion times were recorded. The plot illustrates how increasing failure probabilities impact task completion time.

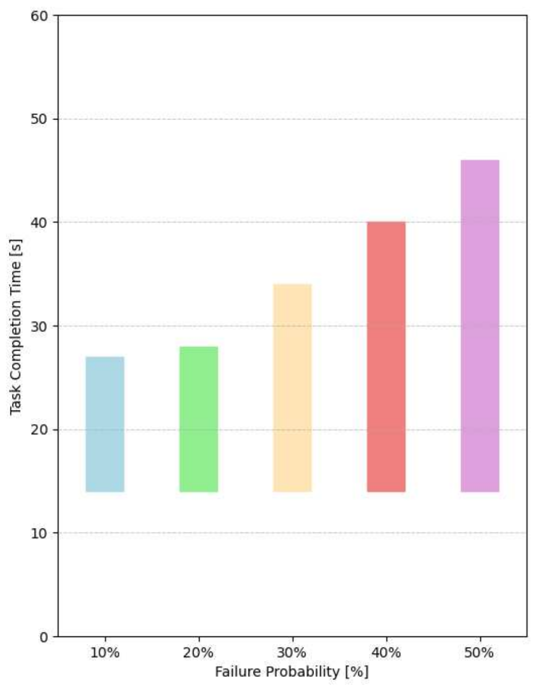

Figure 4.2: Task completion time across failure probabilities for example 1.

Example 2

A second, more complex example is presented, featuring the HTN structure shown below. This example incorporates joint actions, adding complexity compared to the previous scenario. The tests were conducted following the same methodology as the previous example.

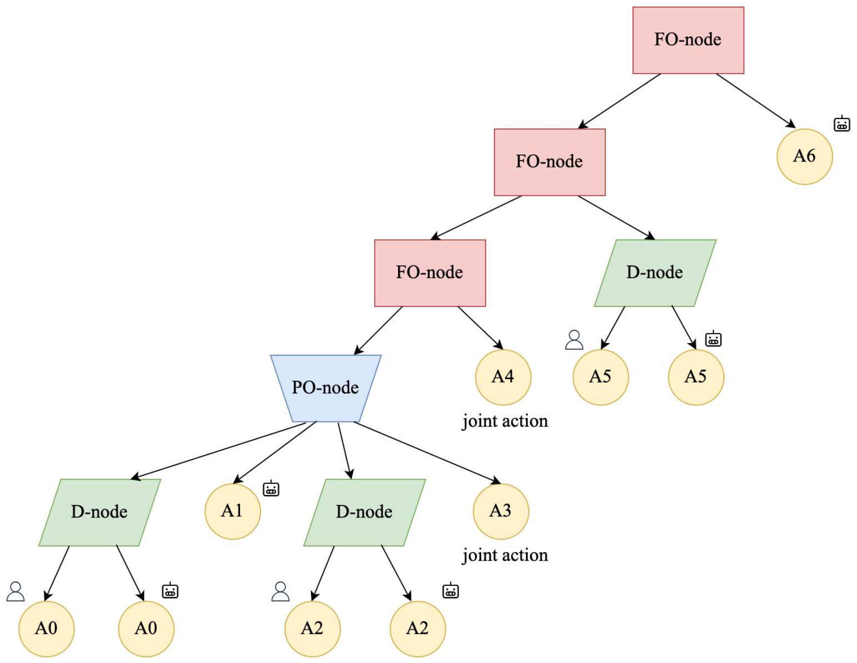

Figure 4.3: HTN task representation for example 2.

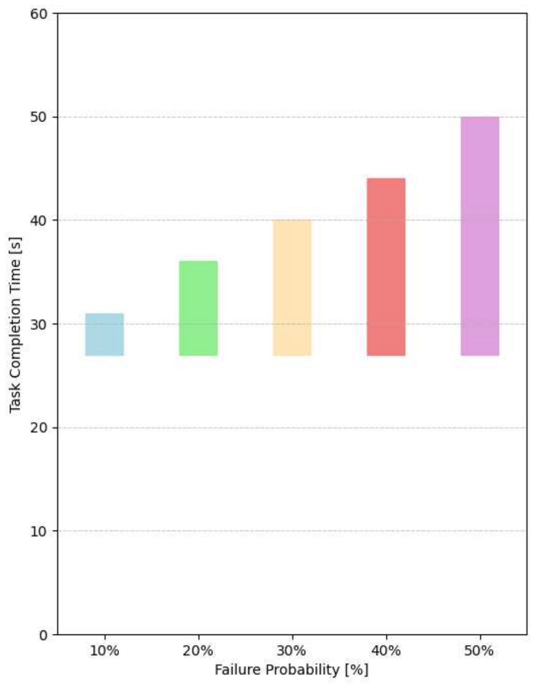

Figure 4.4: Task completion time across failure probabilities for example 2.

The results of these tests clearly indicate that failing actions lead to increased task execution times. However, the framework demonstrates robustness by successfully completing the task in all cases, regardless of the failure probability.

4.2.2 Policy Comparison Experiments

This thesis aims to directly compare the efficiency between different policies. To this aim, a simulated environment has been created based on the *open-ai gym* library [20], implementing the Markov Decision Process (MDP) proposed in [13].

The primary mechanism of the aforementioned MDP relies on event-driven transitions: based on the event that occurs, the robot agent can either be set to *idle* or assigned a new action by its policy, if a feasible action is available. To monitor the progress of task completion, a `state` object is continuously updated, containing the following parameters:

- **`sa`**: A list of binary values that represents the current state of the assembly process, with each element representing an action; set to zero if the action is pending, or one if it has been completed.
- **`ah`**: The current action being performed by the human; this can also be set to *idle*.
- **`hdt`**: The remaining time for the human to complete their current action.
- **`rdt`**: The remaining time for the robot to complete its current action.
- **`j`**: A flag that is set to `True` if a joint action is detected.
- **`aj`**: The current joint action, if one is in progress.
- **`ar`**: The current action assigned to the robot, which can also be set to *idle*.
- **`d`**: A flag that is set to `True` if a human action has been detected.

The key aspect to investigate is the robot's action-selection and interfacing other policies with UHTP policy. To achieve this, a method called `state_to_action` is introduced in UHTP. This method is invoked in the main gym function when the robot is allowed to choose an action. It takes as input the current state, the environment setup, `Ah`, and `Ar` (the lists of actions defined in UHTP) and returns an object of the gym action class.

The environment setup, `Ah` and and `Ar` are autonomously generated by the function `generate_htn` (algorithm 2) given the complete set of actions with their respective possible agents, execution durations, and requirements.

Before applying the policy for the robot's action choice, the method verifies which actions have already been completed. A new attribute, `gym_id`, is introduced in the `Primitive` node class to establish a correspondence between actions in UHTP and those in the gym environment.

The integer variable `gym_id` stores the index of each action in the gym environment, addressing the structural difference between the two systems: in gym, there is one object per action with a list of agents able to execute it, while in UHTP, there is a `Primitive` node object for each agent capable of executing the action.

The method processes the list `sa` to update the lists `Ah` and `Ar`, removing completed actions from the HTN as well. It calls the `remove_nodes` method to prune any nodes that no longer have children. Before assigning an action to the robot, the method verifies if the human currently has an action and calls `pruneBranches` to prevent the robot from selecting the same action. Finally, `selectMinAction` is triggered. If an action is found, it is returned (selected with the gym index) or on the contrary the robot is set to *idle*.

Algorithm 6 State to Action Mapping

```
 1: function state_to_action(curr_state, env, Ah, Ar)
 2:     current_human_action ← curr_state.ah
 3:     durations ← {"human": curr_state.hdt, "robot": curr_state.rdt}
 4:     for each action_state in curr_state do
 5:         if action_state equals 1 then
 6:             remove action_state from Ah or Ar
 7:             find_and_remove(action_state)
 8:         end if
 9:     end for
10:     remove_nodes(root)
11:     if current_human_action exists then
12:         prune_branches()
13:     end if
14:     if durations["robot"] equals 0 then
15:         action ← selectMinAction()
16:         if action exists then
17:             return action
18:         else
19:             return "idle"
20:         end if
21:     end if
22: end function
```

Several examples were tested to evaluate the performances of the proposed approach. The testing began with the chair assembly example presented in the UHTP paper (already reported in previous examples 3.1), followed by four additional examples generated with random requirements. These examples varied in complexity, featuring different numbers of actions: 8, 16, 24, and 32 actions, respectively.

The following plots illustrate the mean and standard deviation results obtained by running 1000 simulations for each policy on each example. These metrics provide insight into the robustness and consistency of the policies across varying task complexities. The other policies adopted to test the efficiency were the Greedy Policy, which selects actions that minimize the single-action cost and, the Random Policy, which randomly assigns an action to the robot by selecting it from the set of available ones.

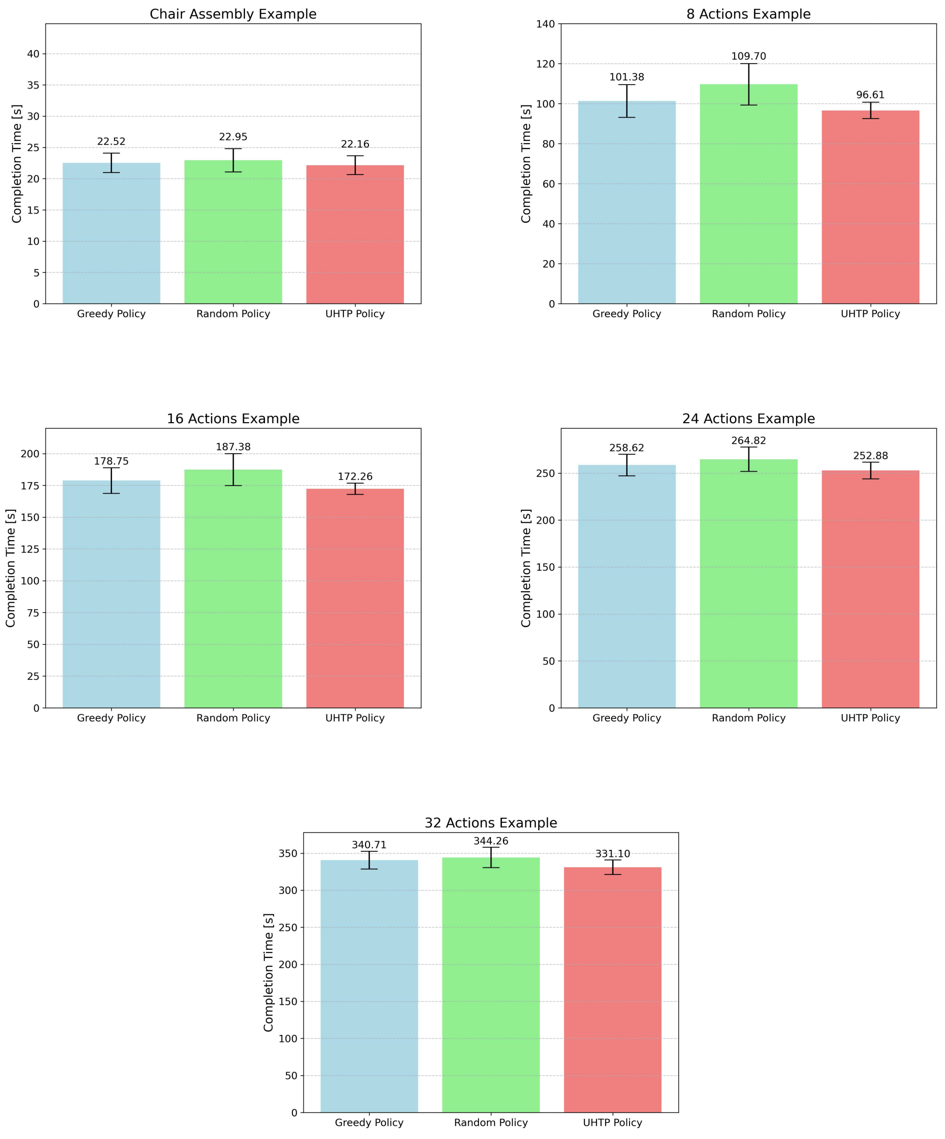

Figure 4.5: Task completion time across different policies.

Looking at the different examples tested, it can be concluded that UHTP Policy generally minimizes task completion time and demonstrates the highest efficiency among the policies evaluated.

Chapter 5

Conclusion and Further Work

In this thesis, we introduced a novel algorithm for retrieving the task representation for assembly tasks, specifically an HTN, from annotated videos. The first achievement was a method that computes actions' requirements given a set of annotated videos as these requirements were crucial for highlighting the hierarchical structure inherent in the sequence of performed actions. The primary challenge lay in identifying a unique hierarchy, given that the actions in the various videos were not performed in a fixed way.

Another method was implemented taking advantage of the hierarchical structure obtained from the requirements. This method constructed consistent HTN tree structures for various tasks and was validated using different examples to ensure reliability and robustness.

In future work, it would be valuable to test the proposed approach on large-scale tasks to further evaluate its effectiveness in more complex scenarios.

As a second contribution, we presented a novel framework for task planning, building upon the existing UHTP framework, which was implemented with a self-derived interpretation. The framework was based on a communication-free environment with no constraints on the human agent and was further extended to include additional features, such as handling joint actions and employing recovery action processes. However, several challenges were encountered during this process. One of the most significant was understanding and interpreting how the original framework was implemented, as the details had to be deducted from the paper [12]. For this reason, the program flow was modified and adjusted to achieve viable results.

Both of the added extensions proved to be robust, enabling the task planning framework to effectively handle complex tasks involving numerous actions and multiple failures. These results were confirmed through extensive validations. Additionally, the efficiency of the UHTP policy for the robot's action selection was compared against other policies, namely Greedy and Random, and emerged to be the most effective overall. Furthermore, the framework demonstrated the ability to efficiently coordinate human and robot planning while treating the human as an unpredictable agent and allowing the robot to dynamically adapt to human decisions.

An interesting study for future research would be to implement mechanisms for handling human changes of mind, further enhancing adaptability and responsiveness of the framework.

Appendix A

Tree Data Structure

Trees are hierarchical structures characterized by a collection of nodes that branch out from a root node through parent-child relationships. These relationships form a minimal set of connections that link all nodes within the structure. A list represents an edge case of a tree, exemplifying a linear hierarchical structure.

A **rooted tree** T is a collection of nodes that, if not empty, satisfies the following properties:

- $\exists$ a special node $r \in T$, called the **root** of the tree.
- $\forall\ v \in T$, $v \neq r$, $\exists$ a unique node $u \in T$ such that u is the **parent** of v (and v is a **child** of u).
- $\forall\ v \in T$, $v \neq r$, tracing back through the parent nodes eventually leads to r (i.e., every node is a descendant of the root).

A node x is an **ancestor** of a node y if $x = y$ or x is an ancestor of y's parent, and x is a **descendant** of y if y is an ancestor of x; **internal nodes** are those with one or more children, while **leaves** are nodes without any children.

A **subtree** rooted at v, denoted as T_v, is the tree formed by all the descendants of v.

An **ordered tree** T is a tree where, for every internal node $v \in T$, a linear order is defined among its children $u_1, u_2, \ldots, u_k$.

The **depth** of a node v in a tree T, denoted as $\text{depth}_T(v)$, can be defined in two alternative ways:

- $\text{depth}_T(v) = |\text{ancestors}(v)| - 1$.
- If $v = r$ (the root) then $\text{depth}_T(v) = 0$, otherwise $\text{depth}_T(v) = 1 + \text{depth}_T(\text{parent}(v))$.

The level i represents all the nodes at depth i $(i \geq 0)$

The **height** of a node v in a tree T, denoted as $\text{height}_T(v)$, is defined as follows:

- If v is a leaf then $\text{height}_T(v) = 0$.
- Otherwise $\text{height}_T(v) = 1 + \max_{w:w \text{ is a child of } v}(\text{height}_T(w))$.

Tree Traversal is a systematic scan of all the nodes in a tree, allowing a specific operation to be performed on each node.

The most common techniques for traversal are the following:

- *Preorder*: First visit the parent, then recursively traverse the subtrees rooted at its children.
- *Postorder*: First recursively traverse the subtrees rooted at the children, then visit the parent.

Algorithm 7 Preorder Traversal

```
Require: A node $v \in T$
Ensure: Visit all nodes of the subtree $T_v$
1: procedure Preorder($T, v$)
2:     Visit $v$
3:     for all $w \in T$.children($v$) do
4:         Preorder($T, w$)
5:     end for
6: end procedure
```

Algorithm 8 Postorder Traversal

```
Require: A node $v \in T$
Ensure: Visit all nodes of the subtree $T_v$
1: procedure Postorder($T, v$)
2:     for all $w \in T$.children($v$) do
3:         Postorder($T, w$)
4:     end for
5:     Visit $v$
6: end procedure
```

[21]

Bibliography

[1] H. H. Zhuo, H. Munoz-Avila, and Q. Yang, "Learning hierarchical task network domains from partially observed plan traces," *Artificial intelligence*, vol. 212, pp. 134–157, 2014.

[2] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 5469–5476.

[3] D. Xu, S. Nair, Y. Zhu, *et al.*, "Neural task programming: Learning to generalize across hierarchical tasks," in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 3795–3802.

[4] K. Darvish, F. Wanderlingh, B. Bruno, E. Simetti, F. Mastrogiovanni, and G. Casalino, "Flexible human–robot cooperation models for assisted shop-floor tasks," *Mechatronics*, vol. 51, pp. 97–114, 2018.

[5] L. Johannsmeier and S. Haddadin, "A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 41–48, 2016.

[6] F. Fusaro, E. Lamon, E. De Momi, and A. Ajoudani, "A human-aware method to plan complex cooperative and autonomous tasks using behavior trees," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2021, pp. 522–529.

[7] F. Fusaro, E. Lamon, E. De Momi, and A. Ajoudani, "An integrated dynamic method for allocating roles and planning tasks for mixed human-robot teams," in *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, IEEE, 2021, pp. 534–539.

[8] J. Shah, J. Wiken, B. Williams, and C. Breazeal, "Improved human-robot team performance using chaski, a human-inspired plan execution system," in *Proceedings of the 6th international conference on Human-robot interaction*, 2011, pp. 29–36.

[9] R. Wilcox, S. Nikolaidis, and J. Shah, "Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing," 2013.

[10] G. Milliez, R. Lallement, M. Fiore, and R. Alami, "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, IEEE, 2016, pp. 43–50.

[11] A. Roncone, O. Mangin, and B. Scassellati, "Transparent role assignment and task allocation in human robot collaboration," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 1014–1021.

[12] K. Ramachandruni, C. Kent, and S. Chernova, "Uhtp: A user-aware hierarchical task planning framework for communication-free, mutually-adaptive human-robot collaboration," *ACM Transactions on Human-Robot Interaction*, 2023.

[13] G. Giacomuzzo and D. Romeres, "Human-robot collaboration for assembling ikea furniture through reinforcement learning-based planning,"

[14] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.

[15] S.-C. Zhu, D. Mumford, *et al.*, "A stochastic grammar of images," *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 4, pp. 259–362, 2007.

[16] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104 096, 2022.

[17] M. Gombolay, R. Wilcox, and J. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," 2013.

[18] K. Erol, J. A. Hendler, and D. S. Nau, "Umcp: A sound and complete procedure for hierarchical task-network planning.," in *Aips*, vol. 94, 1994, pp. 249–254.

[19] T. Cui, G. Chen, T. Zhou, *et al.*, "Human demonstrations are generalizable knowledge for robots," *arXiv preprint arXiv:2312.02419*, 2023.

[20] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *Openai gym*, 2016. [Online]. Available: `https://www.gymlibrary.dev/`.

[21] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data structures and algorithms in Java*. John wiley & sons, 2014.