Semantic shared-Task recognition for Human-Robot Interaction

Adrien Vigné, Guillaume Sarthou, Aurélie Clodic

Semantic shared-Task recognition for Human-Robot Interaction

Adrien Vigné[1], Guillaume Sarthou[1] and Aurélie Clodic[1]

***Abstract*— When collaborating with humans during a shared task, a robot must be able to estimate the shared goal and monitor the tasks completed by its partners to adapt its behavior. Our contribution is a lightweight, hierarchical task recognition system that enables the robot to estimate shared goals and monitor human tasks. This recognition system is integrated into a robotic architecture to take advantage of the semantic knowledge flow available and builds upon our previous work on action recognition. We demonstrate the mechanisms of our recognition system and how we improved it to handle missing information using a kitchen scenario. This also enables us to showcase its usability from the perspective of other agents, using Theory of Mind.**

I. Introduction

For a robotic agent, the field of Human-Robot Interaction (HRI) brings the challenge of completing complex and joint tasks in collaboration with human partners. To achieve such a goal, robots have to correctly understand and interpret what humans around them are currently doing. Let's imagine a robot entering a kitchen where humans are preparing a meal. In order for the robot to help them and engage with them in a joint task, as defined by Tomasello et al. in [1], a shared goal has to be defined. For the robot to be proactive, it first has to estimate the goal of the humans around it. Here, in a kitchen context, the robot has to determine the recipe currently prepared and the advancement of this later, or at least have a shortlist of possible recipes in progress. From there, the robot will be able to engage efficiently in the joint task.

Once engaged in a joint task, as explained by Sebanz et al. in [2], involved agents have to coordinate their actions in space and time. To do so, they have to maintain an estimate of what each is doing in relation to the task on which they have previously agreed. They thus have to monitor others to adapt their own behavior accordingly. Nevertheless, as each agent has to carry out its own actions, such monitoring cannot realistically be continuous. Indeed, each agent will have to focus on its actions but thanks to a few insights like changes in the environment, it could still be possible to estimate what has happened, in order to update the estimate of the plan's progress.

While monitoring the others' actions is mandatory for collaboration, being able to estimate what the others know about the progress of the task is also important. Indeed, even if some actions have visual impacts on the state of the environment, others don't. For example, adding salt to a dish cannot be estimated if it has not been directly perceived. Consequently, to avoid such an action being performed twice, one has to inform the others that the action has been done if it estimates them to not be aware of so. Where some could communicate about each action performed, being able to estimate the others' beliefs allows pertinent and efficient communication acts.

Finally, as humans do not necessarily perform a single task at the time or at least are able to perform several sub-tasks of the same high-level task in parallel, a robot monitoring its partner should be robust to such situations. In addition, it should be considered that some actions are not part of the task, like for example drinking a glass of water.

In light of all these examples, we can see that task recognition in a realistic environment and scenario can be challenging. To tackle these challenges, this work is based on our prior work [3] about action recognition based on semantic facts. While, having been proven to be able to perform online recognition and having been integrated into a robotic architecture, this prior work was not able to recognize complete tasks. The main contribution of this paper is thus a **lightweight method for hierarchical task recognition based on a semantic knowledge flow**, even with partial observations of the scene. In addition, this method is able to emit hypotheses about higher tasks in a way to guide further recognition or to be used at a supervision level. Another contribution related to the HRI field is its direct ability to recognize tasks from the point of view of each agent. This work has been integrated into the Dacobot robotic architecture [4], which is in charge of providing the knowledge flow both from the robot's point of view and the estimated point of view of the human partners.

In Sec. II, we discuss related work and how task recognition is generally performed. The used domain and its underlined challenges are described in Sec. III. In Sec. IV, we present our basic recognition process. Next, in Sec. V we present how missing information is handled, then how hypotheses about the higher task can be generated, and finally how the process can be used with perspective taking. An analysis of the process on a simulated input is presented in Sec.VI before concluding this article and outlining future work.

II. Related work

As presented in [5] by Ingrand et al., providing a monitoring system that not only focuses on the robot but also on its environment is mandatory for a robot to make decisions. Considering Human-Robot Interaction, due to the human

This work has been supported by the Artificial Intelligence for Human-Robot Interaction (AI4HRI) project ANR-20-IADJ-0006 and DISCUTER project ANR-21-ASIA-0005.

[1]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France firstname.surname@laas.fr

ability to act on the environment, human monitoring has also to be considered mandatory. Indeed, more than modifying the state of the world, human acts aim to achieve a goal that a robot has to understand to collaborate. A taxonomy of the different ways to handle such recognition has been proposed by Saleem et al. in [6]. They define two main types of approaches: learning-based on one side and feature-based on the other.

Nowadays, learning-based approaches have gained in popularity. Among the more recent advancements in the field, we can cite [7] in which they combine visual activity understanding with a Large Language Model to improve the recognition explainability, or [8] using a two-stage modality fusion to recognize human actions. While providing impressive results, such approaches still face issues. First, they require huge among of data to be trained. Even if numbers of datasets already exist [6], [9], new ones have to be created to recognize new activities and additional training is required to make such a system evolve. This latter issue can be acceptable but the one that mainly interests us is the requirement to always perceive the human to handle the recognition. For example, in [10], to avoid any occlusions and always keep track of the human, they use multiple cameras to provide a holistic view of the environment to the robot. To be more resilient to small environmental occlusions from a single perspective, Ikizler-Cinbis and Sclaroff proposed in [11] to multiply analysis on the image to focus on regions of interest, not only considering the body motion but also the surrounding objects and humans. Finally, [12] uses a radio frequency sensor to ensure recognition even through walls. However, only focusing on the skeleton they cannot detect interactions with objects. However, as a robot engaged in everyday activities cannot constantly monitor the human partners, as everyday environments cannot be equipped for the robot, and as everyday environments are full of occlusions, we think that such approaches are limited for Human-Robot Interaction. Despite these limitations, learning-based approaches can provide online recognition after being trained in contrast to other feature-based approaches.

Regarding feature-based approaches, we can complete Saleem's taxonomy with the one provided by Ramirez-Amaro in [9] and refine it into syntactic, knowledge & logic, and graph-based approaches. Syntactic approaches aim at converting visual information to textual information and compute the recognition process thanks to grammar systems like the Stochastic grammar system presented in [13]. In this work, Minnen et al. extract symbols representing low-level actions from images. From there, thanks to a complete description of a task, they can recognize if the given task has been performed or not. Thanks to their stochastic aspect, this approach has been proven robust to noise in the perception and partial occlusion of the scene. Nevertheless, it can only recognize one given task and still has to perceive the entire course of action. In [14] they present a Stochastic Context-Free Grammar that allows the recognition of a task made by multiple agents independently or multiple concurrent tasks made by a single agent.

Regarding knowledge and logic, a common approach is the use of planning domains to drive the recognition process. For example, [15] presents a human-aware plan recognition system based on a typical planning domain representing what can be done by humans. Providing a vector of observations, the approach aims at identifying the most likely plan even if part of the observations is missing. With the extension presented in [16], the authors consider tasks performed by multiple agents and extend their system to manage missing or unknown information. To this end, they define three sources of information, each with partial information: the plan library, where some actions are unknown; the action model, where some effects or preconditions are missing; and the observation, where some actions are unknown. Unknown actions are marked with a placeholder to help the recognition system. Although this approach is able to handle missing or unknown information, it cannot run online as it requires a whole sequence of observations. Thus, recognition can only be performed at the end of the task execution. Furthermore, while such a system is more robust to unknown information than to missing information, it is also highly sensitive to false or extra information.

To be more resilient to missing information, false detection, or extra actions, several approaches aim at using systems based on statistics and graphs. In [17], they compare several Hidden Markov Model (HMM) structures and conclude that the aggregation of information allows a reduction of the number of parameters. In addition, they prove that the use of a hierarchical representation of objects provides a more robust recognition thanks to the possibility of learning abstractions. In this work, they provide a way to recognize hierarchical activities or interrelated activities but not both at the same time. Interrelated actions refer to actions that share at least one variable. Thanks to this link the knowledge of one action can help recognize the other. In the case of joint action with hierarchical models, both hierarchical and interrelated activities are necessary for understanding and to be generic. Another approach using extended Petri nets is proposed in [18]. As a source of information, they choose to use an ontology providing a rich and consistent abstraction of the current observation, as well as a taxonomy to describe to actions to be recognized. However, this work cannot handle partial observation of the actions composing a task. This issue has later been tackled with the use of fuzzy ontology and fuzzy semantic Petri nets [19]. Thanks to the addition of probability in the abstraction layout and in the recognition process they successfully handle missing observations. Another approach to solve this problem is presented in [20] where they propagate constraints in the Petri net to use the link between actions that were perceived during the recognition process. The remaining issue here is the ability to recognize tasks on the fly and to recognize multiple tasks for the same agent at the same time because humans made rarely only one thing at the same time.

III. Domain and challenges

To demonstrate this work, we will use a hypothetical scenario of a robot assisting a human in a kitchen. Recipes are essentially plans that involve processing ingredients to create a final product. As a result, they can be transformed into planning domains for robotic applications. This domain is hierarchical in nature, providing an abstracted representation of the high-level tasks to be performed and enabling the reuse of the same sub-tasks in multiple high-level tasks. Furthermore, the kitchen domain contains several low-level actions that are applied to different ingredients or intermediate preparations. In such a way, a sequence of sub-tasks is required to identify a given task among others.

The domain used in this scenario includes recipes for two types of quiches (Lorraine and ham), two types of cakes (ham and olive), and a meringue. The domain also includes other tasks, such as hydration, that are commonly performed in parallel with other tasks. The purpose of this construction is to illustrate specific points of interest in the recognition process and to demonstrate the use of a hierarchical model. For instance, the two quiche recipes share some sub-tasks and can only be differentiated by a single ingredient. In this case, we aim to recognize that a quiche has been made, even if we cannot determine the specific ingredient used. The choice between recipes based on certain ingredients is illustrated by the presence of both a ham quiche and a ham cake. Doing any other recipe, the meringue should not disturb as it contains a specific task and a specific ingredient not shared with the other recipes.

As an illustration of a hierarchy, the preparation of a ham cake is broken down into the task of preparing a cake with ham as a specific ingredient. This task can be further divided into multiple partially ordered tasks, such as preparing the mixture, preparing the ham, or even baking the cake. Each of these tasks can be decomposed until it reaches basic actions, such as mixing ingredients or picking and placing an object. Furthermore, this task of preparing a ham cake occurs during the preparation of starters, which is a sub-task of the meal preparation process. This hierarchical structure is beneficial for modeling purposes and enables our system to generate hypotheses at various levels.

This domain has been written for the Hierarchical Agent-based Task Planner (HATP) [21] due to its ability to take into account humans in the planning problem. Nevertheless, as the internal process doesn't depend on the used domain language, it could also be used with ANML [22] or HDDL [23].

IV. Process principles

To introduce our system of hierarchical task recognition, we first introduce its link to the DACOBOT architecture. Then, we explain the inputs required by our system and the internal structure used in the recognition process. Finally, we present our recognition process in the nominal case.

A. Integration in the knowledge flow

The DACOBOT architecture provides a complete knowledge flow, from perception to decision making, that is used as an input of our system. A representation of this knowledge flow is presented in Fig. 1. The situation assessment, managed by Overworld [24] [1], provides semantically abstracted geometric facts from the fusion of several perception sources. These geometric facts are then stored in our semantic knowledge base, managed by Ontologenius [25] [2], and used for semantic reasoning. This reasoning makes it possible to infer new information, using inverse axioms or chained axioms. Reasoning also reduces noise by checking the consistency of each fact. For example, the fact "the cutting board is over an apple" can be discarded thanks to the coherence checking, even if it has been perceived. Once checked and reasoned about, facts are stored in the episodic knowledge base, managed by Mementar [26], whose role is to organize all this semantic knowledge in time. Finally, our recognition system uses the entire knowledge flow of the DACOBOT architecture. This ensures that temporally aligned and coherent geometric facts are used as observation inputs for the recognition process.

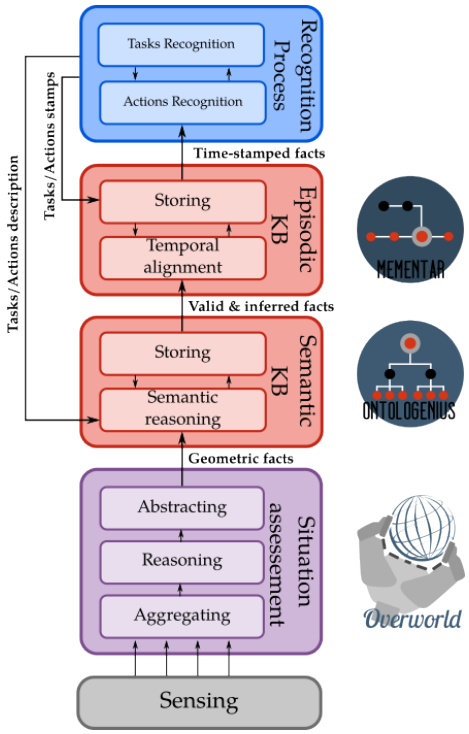

Fig. 1. Knowledge flow for Action Recognition System in the DACOBOT architecture.

B. From planning domain to graphs

As shown in Fig. 1, the initial stage of the recognition process relies on an action recognition process. This process, presented in [3], directly uses the knowledge flow provided by the DACOBOT architecture. Considering a description of the set of possible actions in the form of ordered facts, it provides as an output the recognized actions with their parameters. As an example, a 'place' action is described with

[1] https://sarthou.github.io/overworld/
[2] https://sarthou.github.io/ontologenius/

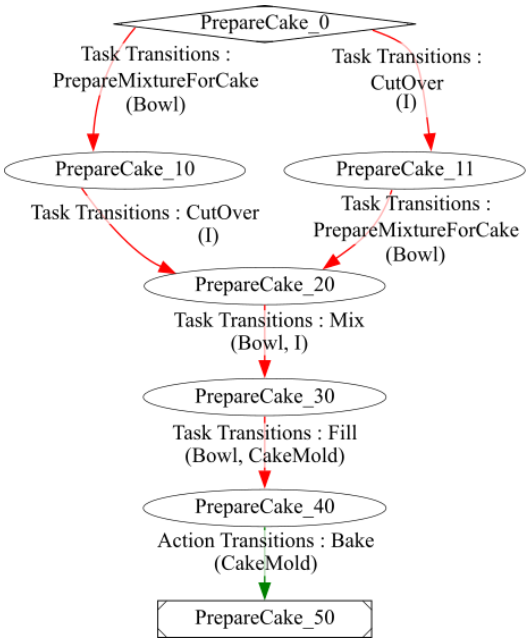

Fig. 2. Graph representation of the only methods of the task PrepareCake. In this graph edges are transitions that represent the completion of the tasks or the actions linked to them. Red edges refer to task transitions and green to action transitions. Nodes are waiting state.

the sequence: `(?A hasHandMovingToward ?S),(NOT ?A isHolding ?O),(?O isOnTopOf ?S)`. In this sequence, `A`, `O`, and `S` represent variables for respectively the agent, the object to place, and the location where to place the object. This process can handle partial observations by using optional facts in a sequence. In this case, the only required fact is that the agent is no longer holding the object. As a consequence, some parameters can be not assigned, such as the location `S` where to place the object in our example. Such recognition is said to be incomplete.

```
task PrepareCake(Ingredient I)
{
  { # method
    preconditions{};
    subtasks{
       Bowl = SELECT(Bowl,{});
       CakeMold = SELECT(Mold,{});
       1: PrepareCakeMixture(Bowl);
       2: CutOver(I);
       3: Mix(I,Bowl)>1,>2;
       4: Fill(CakeMold,Bowl)>1,>3;
       5: Bake(CakeMold)>4; #Action
    };
  }
}
```

Listing 1. A PrepareCake task described following HATP syntax

To be informed about the tasks to be recognized, the task recognition process relies on a hierarchical planning domain. An example of a 'PrepareCake' task description is provided in List. 1. First of all, a task considers as an input a list of typed parameters representing constraints about the entities involved in the task, in our example the ingredient to be used 'I'. Then, a task is defined by a set of **methods** representing alternative ways to achieve it. Each of these methods defines a partially ordered set of so-called **sub-tasks** which can either be other tasks or some actions, that can be recognized by the action recognition process. For the need of this work, the description of the preconditions of the methods is not considered. Nevertheless, where a task can consider parameters as input, each method can define additional local parameters that will be used as arguments in the set of sub-tasks. Finally, the set of sub-tasks can be numbered to express scheduling constraints between these sub-tasks.

In order to handle the task recognition process, the domain has to be converted into an internal structure adapted to the recognition. To do so, we define a graph structure for each individual method of the tasks described in the domain. An example of such a graph is presented in Fig.2 for the 'PrepareCake' task. Indeed, as a task can be used as a sub-task of several higher tasks in the hierarchy, the domain is not flattened. An advantage of such a choice is the possibility to recognize tasks at any level. For example, a CutOver task can be recognized even if the higher task it is part of is not.

In this graph, the nodes represent steps in the achievement of the task and its edges, being transitions over the nodes, represent conditions to evolve from one node to another. More precisely, these transitions are the completion of the sub-tasks used in the method. They can thus be triggered thanks to **observations** that correspond to the tasks or the actions represented on the transitions. Depending on the transitions' type, observations will either come from the output of the action recognition system (with complete or incomplete actions) or from the task recognition system itself. A task is considered to be finished when one of these graphs reaches its final state. In such a case, this completion generates an observation that could be used to trigger other graphs.

To create a graph, we first propagate each scheduling constraint through the set of sub-tasks in the method. Here, for example, the Fill(4) sub-task is constrained to be run after 1,2,3 as Mix(3) is constrained to be run after 1 and 2. Using the same reasoning, Bake(5) is required to be run after all the other tasks. Thanks to this propagation, we create the graph of all possible ways to complete the task according to this scheduling constraint.

The graph used as an internal structure is linked to a **table of variables**, which represents all the variables present as parameters for sub-tasks. In our case, the table is composed of: $Bowl$, I, and $CakeMold$. This table will be used in the following to keep track of the variables instantiations.

The set of agents that have participated in completing a task is filled with each agent having completed a sub-task. If Agent_A and Agent_B respectively complete the sub-tasks 'PrepareMixtureForCake' and 'CutOver', they will be included in the set of participating agents for the task 'PrepareCake'.

C. The recognition process

If several agents are working simultaneously on the very same task, we cannot use a single graph instance per method

to handle the task recognition. To manage such situations, we introduce the notion of **graph factory**. A graph factory is responsible for a single method.

To initiate a recognition, a graph factory only focuses on the transitions from the initial state of the graph it is in charge of. When an observation is submitted to a factory if it can trigger one of the initial transitions, an **active graph** is created by the factory. For example, if the observation "PrepareCakeMixture($bowl\_1$)" is submitted to the factory in charge of the method 'PrepareCake', this factory will generate an active graph. Since a transition has been triggered, the active graph will already reach the 'PrepareCake_10' node. In addition, the 'Bowl' variable of the generated graph will be bonded to the individual $bowl\_1$.

Let's assume that a new observation 'CutOver(ham_0)' arrives. Before reaching a factory this observation will be used to try to evolve all active graphs. If it succeeds to evolve active graphs, the observation will not be submitted to the related factories of the evolved active graphs. Otherwise, the observation will be sent to all the factories. In our example, the observation 'CutOver(ham_0)' allows the evolution of the previously created active graph, reaching the 'PrepareCake_20' node. It also results in the instantiation of the variable I, being bound to ham_0. From the current node, the only possible transition is 'Mix(I, Bowl)'. However, according to the table of variables, we are not waiting for any mix, but specifically the one specified by the current values of the variables, which is 'Mix(ham_0,$bowl\_1$)'.

To prevent false or inconsistent recognition, a cleaning process is initiated after each task completion. It consists of a pruning of all active graphs that have used at least one observation involved in the finished task. Additionally, a timeout is implemented for each graph to remove the ones that have not evolved for a while.

V. Additional features to enhance the recognition

In this section, we present additional features enhancing the basic mechanism of our recognition process. First, we present a support for missing observations allowing for partial observations. Then, we present a mechanism to emit hypotheses about the high-level tasks in progress, not waiting for task completion. Finally, regarding the HRI field, we show that thanks to its integration into the DACOBOT architecture, the task recognition process is directly applicable to the Theory of Mind, enabling to estimate the tasks recognized by other agents.

A. Managing missing information

As previously discussed, effective management of the recognition process in everyday environments requires addressing several issues. One such issue is the necessity for the recognition process to be resilient to missing information since it is unrealistic for the robot to continuously monitor all the agents in the environment. Although the action recognition process has addressed this issue by incorporating optional facts into the detection process, until now the presented version of the task recognition process requires a full observation of the sub-tasks.

Taking back our example, being in the state 'PrepareCake_20', the process is waiting for an observation that matches the realization of the sub-task 'Mix($bowl\_1$,ham_0)'. Therefore, receiving the observation 'Fill($bowl\_1$,$cakemold\_3$)' does not allow for evolution even if it is part of the task. To solve this issue, we introduce a look-ahead mechanism that allows every node to test the transitions of its direct child nodes. In our example, from state 'PrepareCake_20' we can thus test the transitions of state 'PrepareCake_30'. As previously, this test uses the constraints on the variables if they are instantiated. If the test succeeds, the missed transition is bypassed and the unbounded variables are instantiated if possible. In our example, we directly pass to node 'PrepareCake_40' and we instantiate the variable *'CakeMold'* to $cakemold\_3$.

As the bypassed transition can be due to an unfinished task having thus not raised any observation, we have implemented a cherry-picking mechanism aiming at filling the missed information thanks to unfinished graphs. For example, having bypassed the transition 'Mix($bowl\_1$,ham_0)', we check among all the active graphs if a mix task is under recognition. If such a graph exists, even with incomplete parameters, this graph is picked to fill the missed transition. The graph is thus considered as finished and is removed from the set of active graphs. As usual, all other graphs sharing common observations are also purged from the set of active graphs. In addition, all the unbounded variables of the picked task are linked to the task under recognition in order to instantiate them if possible. Thanks to this mechanism we can extract as much information as possible to explain the current situation.

If no active graph can be found, the missed task or action is reconstructed with the available piece of information. However, even if all the variables can be determined, such reconstruction will always be incomplete as the agent having performed it will stay unknown.

Thanks to these mechanisms, our task recognition system can handle missing information and extract or generate it in a coherent manner. Each graph factory can also use these mechanisms to generate active graphs not only with the transitions of the initial state.

B. Hypothesis to provide estimation of shared goal

An additional feature provided by our system is the ability to generate hypotheses about the higher level task in progress. Where tasks usually need to be finished in order to generate observations, to provide hypotheses we propagate each active graph as a **hypothetical observation** attached with a completion rate. The observation is sent to graph factories in a classical manner. If the graph factory can evolve a **hypothetical graph** is created. Hypothetical graphs then follow the same mechanism of evolution as active graphs and can also use the look-ahead mechanism. Concerning observations, they are also only sent to graph factories from where no hypothetical graph of the same type has been evolved. A consequence of the hypothesis mechanism is

that a same method can be represented with active and hypothetical graphs simultaneously. In such a case, only the hypothetical graph generates hypothetical observations. For instance, in node PrepareCake_40 of our example, the hypothetical graph of PrepareCake would generate the observation 'PrepareCake(ham_0),4/5'. This observation triggers the evolution of the PrepareCake hypothetical graph created through the PrepareCake active graph. The calculation about the estimation is

$$\sum_{sub-tasks} completion\_rate * \frac{1}{nb\_sub - tasks}$$

where $nb\_sub - tasks$ refers to the number of the sub-tasks in the higher task triggered by the propagation of the hypothetical observation.

C. Multiple knowledge streams to estimate from agent's point of view

To effectively assist in a shared task with multiple agents, a robot has to estimate each agent's knowledge about the progress of the task, as well as what has been done by each agent and what they are currently doing. For instance, in a restaurant kitchen, each cook is responsible for a specific part of the dish that will be served, so all the cooks must estimate each other's progress to ensure timely completion. Similarly, waiters must also estimate the progress of the kitchen staff, but at a higher level, to ensure the timely service of the dish. To provide this estimation, our system uses the features of each software within the DACOBOT architecture to manage multiple knowledge streams in parallel. These streams are generated by the situation assessment and are fed based on the estimation of what each agent can observe in the environment. To handle these multiple streams, we manage dynamically one recognition process by knowledge stream, and thus per agent. This allows us to estimate the recognition process from their point of view.

This feature can also be useful to identify belief divergences in the current progress of the shared task. Such divergences can occur, for example, if one agent adds salt to the preparation but the other cook is unaware of this. In such cases, thanks to the estimation process, the robot can communicate to avoid duplication of the task.

VI. Analysis of a recognition

In this section, we present the results of our recognition process through a step by step analysis of an entire task recognition from action observations to hypothesis generation. The entire section considers a Pepper robot observing two cooks achieving the shared task of preparing a ham cake. We consider that the robot does not know the task to be achieved and has to correctly identify it. The used task domain is the one presented in Sec. III. To simplify the demonstration, we only assume that the 'Mix' sub-task in the 'PrepareCake' method is no longer a task but an action that can be directly recognized by the action recognition. First, we focus on the task recognition from a set of observations. Then, we analyze the hypothesis generation. Lastly, we analyze the task recognition in the case of the Theory of mind thanks to estimates from different points of view.

	Observation	Initiated	Evolved
1	Mix(A,f_0,b_4)	PrepareCakeMixture	
2	Pick(B,g_0)	Hydrating	
3	FillWater(B,g_0)		Hydrating
4	Drink(B,g_0)		**Hydrating**
5	Hydrating,{B}		
6	Pick&Place(B,h_0,a_6)	CutOver	
7	Mix(A,e_0,b_4)	PrepareQuicheBase	PrepareCakeMixture
8	Pick(B,k_10)		CutOver
9	Mix(A,cr_2,b_4)		PrepareCakeMixture PrepareQuicheBase
10	Cut(B,h_0,k_10)		**CutOver**
11	CutOver(h_0),{B}	PrepareCake PrepareQuiche	
12	Mix(A,m_3,b_4)		PrepareCakeMixture PrepareQuicheBase
13	Mix(A,ch_5,b_4)		**PrepareCakeMixture** ~~PrepareQuicheBase~~
14	PrepareCakeMixture(b_4),A		PrepareCake
15	Pick&Place(A,m_0,t_1)	Fill	
16	Pick(A,b_4)		Fill
17	PoorInMold(A,b_4,m_0)		**Fill**
18	Fill(b_4,m_0),{A}		PrepareCake
19	Bake(B,m_0,o_3)		**PrepareCake** ~~PrepareQuiche~~
20	PrepareCake(h_0),{A,B}	PrepareHamCake	**PrepareHamCake**

TABLE I

OBSERVATIONS, WITH INITIATED AND EVOLVED ACTIVE GRAPHS. BOLD GRAPHS REPRESENT FINISHED GRAPHS AT A STEP. CROSSED-OUT ONES REPRESENT DELETED GRAPHS. GREYED ROWS REPRESENT OBSERVATIONS GENERATED BY THE TASK RECOGNITION PROCESS.

A. From observations to recognition

To demonstrate our recognition process we consider the observations presented at the left of Tab.I. When the 1st observation occurs, the recognition process does not have any active graph. The observation is thus sent to all the graph factories. Because the Mix observation is specified with flour ingredient, the only triggered factory is the one of PrepareCakeMixture. This latter thus generates a new active graph. When the 2nd observation occurs, the only available active graph cannot be evolved thanks to it and as previously, the observation is sent to the factory. Once again, only one can be triggered and a new active graph, of type Hydrating, is generated. With the 3rd observation, the Hydrating graph can be evolved. As a consequence, the observation is sent to the factories except the one in charge of the Hydrating task. Finally, when the 4th observation occurs, this latter evolves the Hydrating graph and finishes it (in bold in Tab.I),

generating in return a task observation and removing it from the pool of active graphs. This new observation (in grey in Tab.I) is sent to the only remaining active graph and to the factories but does not allow any progress.

When the Pick&Place observation occurs, with regard to the considered domain, because of its parameters the only effect is the creation of a new active graph, of type CutOver. Unlike this, the 7th observation has two effects. First, being tested over the active graphs, it allows a direct evolution of the PrepareCakeMixture graph. Additionally, when sent to the factories, it triggers the PrepareQuicheBase factory. One has to note that since an active graph of type PrepareCakeMixture has evolved, the observation is not sent to the factory in charge of this task. If this had been done, another graph would have been generated because of the look-ahead mechanism.

When the 8th and 9th observations arrive, they respectively evolve the CutOver task for the first, and the two other graphs for the second. The 10th finishes the CutOver task generating a new observation. This latter does not allow the evolution of the remaining graphs and triggers two factories, generating in return two active graphs.

Where the 12th observation evolves two concurrent graphs, the 13th finally allows us to discriminate them. Indeed, the latter observation finishes the PrepareCakeMixture and thus removes it from the set of active graphs. In addition, thanks to the cleaning mechanism, as the PrepareQuicheBase graph shares common observations with the newly finished graph, it is also removed. Consequently, only the finished graph generates a new observation which allows the evolution of one of the remaining active graphs.

Thanks to the observations from 15 to 17, a Fill task is recognized and generates an observation. Where none of the active graphs is waiting for a fill task, because of the look-ahead mechanism, the PrepareCake graph can be evolved. The bypassed transition corresponds to a Mix action. As the cherry-picking mechanism can only be performed on task, the action will simply be reconstructed.

Finally, the 19th allows for the recognition of the PrepareCake graph, discarding in consequence the PrepareQuiche one. This finalization generates a new observation but where the previous ones were only related to a single agent, this one embeds both agents as the sub-tasks composing it have been made by multiple agents. The new observation, in line 20, triggers a factory and directly finishes the graph as composed of a single sub-task.

From these few observations, we can see that the system has recognized a shared task of preparing a ham cake without any knowledge of the shared goal between the two cooks. In addition, the recognition has been successful even if an action has not been perceived. Thanks to the hierarchical representation, the same would have been true if an entire task would have been missed. Finally, more than a cake preparation, the system has been able to recognize another task in between.

B. From recognition to estimation

To illustrate our hypothesis mechanism, we will focus on the calculations done after some observations. Tab.II summarizes some interesting cases of hypotheses calculations. When receiving the first observation that generates a PrepareCakeMixture active graph, this latter generates an hypothetical observation. As it is done for usual observation, this one will be sent to the graph factories. Nevertheless, as explained earlier, rather than generating active graphs they will generate hypothetical graphs. Here, the PrepareCakeMixture hypothetical observation will generate a PrepareCake hypothetical graph. Regarding the estimation rate, as the completion rate of PrepareCakeMixture is 1/5 (i.e. one sub-task finished over the five to be done), and because PrepareCake has a sub-task weight of 1/5 (i.e. it requires five sub-task so each has a weight of 1/5), the rate is 4%.

At the step of the sixth observation, the newly created active graph will also generate an hypothetical observation resulting both in the evolution of the PrepareCake hypothetical graph and the creation of a PrepareQuiche one. At step 11, thanks to the completion of the task CutOver with some ham and thus the activation of the graphs PrepareCake and PrepareQuiche, two new hypothetical graphs are generated: PrepareHamCake and PrepareHamQuiche. As a consequence, the tasks PrepareCake and PrepareQuiche are represented both in their active form and their hypothetical form. In such a case, the hypothetical observation is generated by the hypothetical graphs rather than by the active ones. The completion rate of the PrepareCake task is thus 1/5 (i.e. one sub-task finished over the five to be done on PrepareCake) plus 3/5 (i.e. the completion rate of PrepareCakeMixture) * 1/5 (i.e. the weight of PrepareCakeMixture in PrepareCake).

At step 13, as the active graph PrepareQuicheBase is removed, the related hypothetical graph PrepareQuiche is also removed. Finally, at step 15, as the Mix sub-task has been bypassed, this latter is not considered in the completion rate calculation. However, if a cherry-pick would had been performed in the case of a task, its current completion rate would have been used instead.

From these results, we can see that even if their estimation rate is low, estimations can be generated early in the

Observation	Completion Rate		Weight	Result	
1	PrepareCakeMixture	1/5	1/5	PrepareCake	4%
6	PrepareCakeMixture CutOver	1/5 1/3	1/5	PrepareCake	10.6%
	CutOver	1/3	1/6	PrepareQuiche	5.5%
11	PrepareCake	1/5 + 3/5 * 1/5	1	PrepareHamCake	32%
	PrepareQuiche	1/6 + 2/4 * 1/6	1	PrepareHamQuiche	25%
15	PrepareCake	2/5 + 1/3 * 1/5	1	PrepareHamCake	46,7%

TABLE II

SUMMARY OF SOME HYPOTHESES AND COMPLETION RATES USED TO COMPUTE THEM.

achievement of the task. Indeed, with a larger domain, more hypotheses would be generated leaving a higher uncertainty. Nevertheless, such uncertainty is coherent under the hypothesis that multiple tasks can be achieved in parallel with no relations between them.

C. From an agent's point of view

Having analyzed the system from the perspective of the robot, in this part we will analyze it through the lens of perspective taking. To do so we will consider the observations of Tab. II but from the estimated point of view of Cook_B. Because of the perspective taking we assume Cook_B to not perceive the Mix of observation 9. However, thanks to the look-ahead mechanism the PrepareCakeMixture graph is still recognized at step 13. Because the Mix action preceding Fill is assumed to not have been perceived by the robot, it will consequently be assumed to not be perceived by any other agent. Indeed, from the robot's point of view, the others' estimated knowledge can only be a subset of the robot's knowledge. Similarly to the robot estimation, once the Fill task is recognized the look-ahead mechanism will once again bypass the missed sub-task, resulting at the end by a correct estimation of recognition.

Even if this example shows that task recognition can be handled by our process based on perspective taking, it also highlights its complexity. Indeed, by combining the information missed by the robot with the ones missed by the other agents, we are aware that some recognition could be impossible. Nevertheless, even if some high-level tasks would not realistically be recognized, such a process still provides the robot with new knowledge that could help in decision-making. In addition, as most of the actions have an impact on their environment, they could be recognized afterward by the action recognition process, leveraging the issue at the task level.

VII. Conclusion and Future Work

This paper presents a task recognition system based on previous work on action recognition. The system uses dynamically generated and instantiated graphs to represent the recognition process, which can evolve thanks to observations. The system is integrated into the knowledge flow of a robotic architecture and can be used in realistic environments with occlusion and multiple humans performing multiple parallel tasks. The system can handle missing information, making it suitable for use on robots that not only observe but also handle their own tasks to collaborate efficiently with other agents. In the future, the recognition system could be further enhanced to recognize with even less information. As previously noted, especially in the case of estimation from the point of view of another agent, missing information can be numerous. The reconstruction mechanism could also be improved to better complete the recreated or cherry-picked sub-tasks by incorporating additional information such as the availability of the agents.

References

[1] M. Tomasello, M. Carpenter, J. Call, T. Behne, and H. Moll, "Understanding and sharing intentions: The origins of cultural cognition," *Behavioral and brain sciences*, 2005.

[2] N. Sebanz, H. Bekkering, and G. Knoblich, "Joint action: bodies and minds moving together," *Trends in cognitive sciences*, 2006.

[3] A. Vigné, G. Sarthou, and A. Clodic, *Primitive Action Recognition Based on Semantic Facts*. Springer Nature Singapore, 2023.

[4] G. Sarthou, A. Mayima, G. Buisan, K. Belhassein, and A. Clodic, "The director task: a psychology-inspired task to assess cognitive and interactive robot architectures," in *RO-MAN*. IEEE, 2021.

[5] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artificial Intelligence*, 2017.

[6] G. Saleem, U. I. Bajwa, and R. H. Raza, "Toward human activity recognition: a survey," *Neural Computing and Applications*, 2022.

[7] X. Wu, Y.-L. Li, J. Sun, and C. Lu, "Symbol-llm: Leverage language models for symbolic system in visual human activity reasoning," *Advances in Neural Information Processing Systems*, 2024.

[8] H. Zheng, B. Zhang, J. Lin, D. Zhao, and G. Shang, "A two-stage modality fusion approach for recognizing human actions," *IEEE Sensors Journal*, 2023.

[9] K. Ramirez-Amaro, Y. Yang, and G. Cheng, "A survey on semantic-based methods for the understanding of human movements," *Robotics and Autonomous Systems*, 2019.

[10] A. Iosifidis, A. Tefas, and I. Pitas, "Multi-view human action recognition under occlusion based on Fuzzy distances and neural networks," in *European Signal Processing Conference (EUSIPCO)*, 2012.

[11] N. Ikizler-Cinbis and S. Sclaroff, *Object, Scene and Actions: Combining Multiple Features for Human Action Recognition*. Springer Berlin Heidelberg, 2010, pp. 494–507.

[12] T. Li, L. Fan, M. Zhao, Y. Liu, and D. Katabi, "Making the Invisible Visible: Action Recognition Through Walls and Occlusions," in *International Conference on Computer Vision (ICCV)*. IEEE, 2019.

[13] D. Minnen, I. Essa, and T. Starner, "Expectation grammars: leveraging high-level expectations for activity recognition," in *Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2003.

[14] D. Moore and I. Essa, "Recognizing multitasked activities from video using stochastic context-free grammar," *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[15] H. Zhuo, "Human-aware plan recognition," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[16] H. H. Zhuo, "Recognizing multi-agent plans when action models and team plans are both incomplete," *Transactions on Intelligent Systems and Technology*, pp. 1–24, 2019.

[17] D. Patterson, D. Fox, H. Kautz, and M. Philipose, "Fine-grained activity recognition by aggregating abstract object usage," in *International Symposium on Wearable Computers (ISWC'05)*. IEEE, 2005.

[18] N. Ghanem, D. DeMenthon, D. Doermann, and L. Davis, "Representation and recognition of events in surveillance video using petri nets," 2004.

[19] P. Szwed, *Modeling and Recognition of Video Events with Fuzzy Semantic Petri Nets*. Springer International Publishing, 2016.

[20] M. Albanese, R. Chellappa, V. Moscato, A. Picariello, V. S. Subrahmanian, P. Turaga, and O. Udrea, "A constrained probabilistic petri net framework for human activity detection in video," *Transactions on Multimedia*, 2008.

[21] L. De Silva, R. Lallement, and R. Alami, "The hatp hierarchical planner: Formalisation and an initial study of its usability and practicality," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015.

[22] D. Smith, J. Frank, and W. Cushing, "The anml language."

[23] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford, "Hddl: An extension to pddl for expressing hierarchical planning problems," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[24] G. Sarthou, "Overworld: Assessing the geometry of the world for human-robot interaction," *Robotics and Automation Letters*, 2023.

[25] G. Sarthou, A. Clodic, and R. Alami, "Ontologenius: A long-term semantic memory for robotic agents," in *RO-MAN*. IEEE, 2019.

[26] G. Sarthou. Mementar. [Online]. Available: https://github.com/sarthou/mementar