Ref. Ares(2023)8902545 - 29/12/2023

This project is funded by the European Union
G.A No 101016906

Project Number: 101016906
Start Date of Project: 2021/01/01
Duration: 48 months

Type of document D6.3 – V1.0

Human-Robot Awareness

Dissemination level	PUBLIC
Submission Date	2023-12-29
Work Package	WP6
Task	T6.4
Type	Report
Version	1.0
Author	UNIROMA1
Approved by	UNIROMA3, DTI

DISCLAIMER:
The sole responsibility for the content of this deliverable rests with the authors. It does not necessarily reflect the opinion of the European Union. Neither the Directorate-General for Communications Networks, Content and Technology, Artificial Intelligence and Digital Industry nor the European Commission are responsible for any use that may be made of the information

This project is funded by the European Union G.A No 101016906

Executive Summary

The term awareness refers to the state or ability to perceive, feel, or be conscious of events, objects, or sensory patterns. In this level of consciousness, a person (or a robot) can recognize and understand what is happening around him/her (it). This definition is quite generic and open to many different interpretations.

Even though awareness (especially robot and environmental awareness) is a goal of several different tasks in the CANOPIES project, the specific goal of T6.4, which will be the main topic of this deliverable, is a higher level of awareness on both human and robot side about complex activities that are performed in the environment. In particular, the main idea behind the task is (i) for the robot, to be aware of what is its role in the high-level activities the human wants to be performed in the vineyard, and (ii) for the human, to be aware of what the robot is doing or is going to do in the future to contribute to those activities. Also, in T6.4, we refer to explicit awareness, i.e., the one that can be achieved by explicit communication as indicated in Deliverable D6.2, whereas other tasks (es. T6.1 and T6.2 are more related to implicit communication).

In order to obtain this goal, a pipeline has been implemented strongly relying on machine learning and symbolic artificial intelligence (automated planning in particular). The pipeline can be integrated with other awareness modules from other tasks.

The deliverable describes the pipeline and shows initial experiments in the context of the CANOPIES scenario.

Table of Content

1 Introduction 5
1.1 The concept of awareness in literature 5
1.2 Awareness in CANOPIES T6.4 6
2 Architecture and Implementation 7
2.1 Converting speech into LTL specification of activities 8
2.2 Planning with activity specification as goal 10
2.3 Querying the robot 11
2.4 Environmental awareness of the robot and integration in the CANOPIES Ecosystem 11
3 Example of complex action (sub-activity): Harvesting all bunches in a specific position 12
3.1 Model 12
3.1.1 Example 13
3.2 Reward function 14
3.2.1 Including time to harvest 14
3.3 Dynamic programming for action selection 15
4 Experiments 16
4.1 Conversion of natural language into an LTFf formula 16
4.2 Querying the system for the current status of activity 18
5 Future developments 18
6 Bibliography 19
7 Appendix A – Experiments with ChatGPT4 20

Abbreviations and Acronyms

LLM	Large Language Model
LTLf	Linear Temporal Logic on finite traces
PDDL	Planning Domain Definition Language
FOND	Fully Observable Non-Deterministic
JSON	JavaScript Object Notation

This project is funded by the European Union G.A No 101016906

1 Introduction

The concept of awareness is defined as the capacity or state of being conscious, feeling, or perceiving events, objects, or sensory patterns. Within this realm of consciousness, an individual (or a robot) possesses the capability to identify and comprehend the occurrences in their surroundings. This explanation is fairly broad and subject to a wide range of interpretations. In the introduction to this document, we start by defining what Human-Robot awareness is and what is not.

In first place, the term human-robot awareness implies both a human awareness and a robot awareness. Noteworthy, robot awareness plays a central role in the CANOPIES project independently of T6.4 – Human-robot awareness. All the tasks that are related to the perception of the robot of the surrounding environment (e.g., its position in the environment, the number and condition of grapes in the vineyard) already contribute to the robot awareness with respect to the environment and the human. Even though this awareness is integrated in the Human-robot awareness, the goal of T6.4, which will be the main topic of this deliverable, is a higher level of awareness on both human and robot side about complex activities that are performed in the environment. In particular, the main idea behind the task is (i) for the robot, to be aware of what is its role in the high-level activities the human wants to be performed in the vineyard, and (ii) for the human, to be aware of what the robot is doing or is going to do in the future to contribute to those activities.

In this sense, the system implemented in Task 6.4 can potentially use all of the robot awareness and robot enactment functionalities implemented in other tasks of the project (additional details are provided in Section 2.4).

The remaining of this document is structured as follows. The following subsections of Section 1 will introduce the concept of awareness in literature and how it applies to the CANOPIES project and, in particular, to Task 6.4. Section 2 describes into details the architecture implemented in the context of Task 6.4 and how it integrates with other tasks in the CANOPIES Project. Section 3 introduces, with illustrative purposes, an example of sub-activity, related to the harvesting activity, which has been implemented in the context of Task 6.4. Section 4 shows experiments conducted on the architecture described in Section 2. Section 5 defines the next activity to be conducted in the context of the project with respect to awareness.

1.1 The concept of awareness in literature

Different definitions of awareness in Human-Robot Interaction are provided in [1]. Particularly of interest for T6.4 are the following definitions:

- “an understanding of the activities of others, which provides a context for your own activities” [2]
- “given two participants p1 and p2 who are collaborating via a synchronous collaborative application, awareness is the understanding that p1, has of the identity and activities of p2” [3]
- “the participants’ understanding of how their tasks will be completed” [4]

- "knowledge about such things as people's roles and responsibilities, their positions on an issue, their status, and group processes" [5]
- "the general sense of who is around and what others are up to" [5]

Even though authors in [1] propose other definitions, they are more related to the kind of awareness which is the topic of other tasks in the project, so we will focus on works that envision awareness at the high-level activities (or *process*) level. In recent contributions on the topic though, the term Human-Robot awareness in this sense is much related to the term Human-Robot (or Multi-robot) collaboration in complex activities. In T6.4 though, we are not interested in collaboration. Instead, our aim is to provide a way to make the human and the robot involved in a complex activity aware one each other in an explicit manner. This explicit communication distinguishes what is done in Task 6.4 with what is done in Task 6.1 and Task 6.2.

Authors in [6], for example, model collaboration activities in a multi-robot context by using Petri Nets and message exchange to implement *Multi-robot* collaboration. Even though in the original CANOPIES description of work, Petri nets were suggested as the identified solution for Task 6.4, successive considerations, brought the consortium to consider LTL – Linear Temporal Logic as the approach to follow to describe complex activities (see for example Deliverable 8.2). From the point of view of Task 6.4, in particular, Petri nets have two major limitations:

1. Petri nets must be defined in advance. This is just fine for certain kinds of applications, but when humans are involved, they could prefer a more natural way to represent high level activities.
2. The technique is mostly designed for the Multi-robot collaboration, when in general message exchange is not the more natural way of interacting in the Human-robot collaboration paradigm.

Also, vast majority of the works focusing collaboration are more related to the enactment of collaboration than to actual awareness.

1.2 Awareness in CANOPIES T6.4

In order to overcome the limitations identified in the literature with respect to Human-Robot awareness involving high level activities, we defined the following design goals for the module developed in the context of T6.4:

- Do not rely on a fixed set of high-level activities. Conversely, the objective is to let humans to define processes on the fly, according to the needs encountered on the field, being able to be aware of what the robot is going to do in order to satisfy the expressed goals.
- Use natural language as mean to communicate intentions and information both in the Human-to-Robot and in the Robot-to-Human directions.
- Constrain human high-level activities to tasks actually executable by the robot on the environment. This must be obtained by modelling the environment and possible actions in a formal way.

- Integrate high-level activity awareness with other kinds of robot awareness implemented in other tasks of the project.
- Integrate the human awareness module with the decision-making module and, in general, with the entire Human-Robot interaction component, developed in the context of T6.3 (see Deliverable 6.2) for the enactment of high-level processes.
- Awareness must be obtained using explicit means of communication, as the ones defined in T6.3, and differently from what is done in T6.1 and T6.2.

2 Architecture and Implementation

Figure 1 shows the architecture of the Human-Robot Awareness module developed in the context of T6.4 (results of this research have been submitted for peer reviewing [11,12]. The main idea is to translate natural language descriptions of high-level activities from the human into a plan executable on the environment of which the human can monitor the execution by using natural language. In order to generate the plan, the system relies on a PDDL – Planning Domain Definition Language describing the actions available on the environment in terms of preconditions and postconditions. By specifying a goal, expressed in terms of predicates on the environment, a planner is able to define an execution plan that can be executed and queried. In order to specify the goal, the idea is to turn a high-level activity expressed in natural language (e.g., through a microphone) into a logic formula. Automated planners usually take this goal formula in the form of a conjunction of truth predicates. We instead employ LTLf – Linear Temporal Logic (on finite traces) as an intermediate representation formalism, which simplifies the translation from natural language. Once a specific plan is defined, during its execution, the human can ask questions about the current status of the activity and the next steps to be executed.

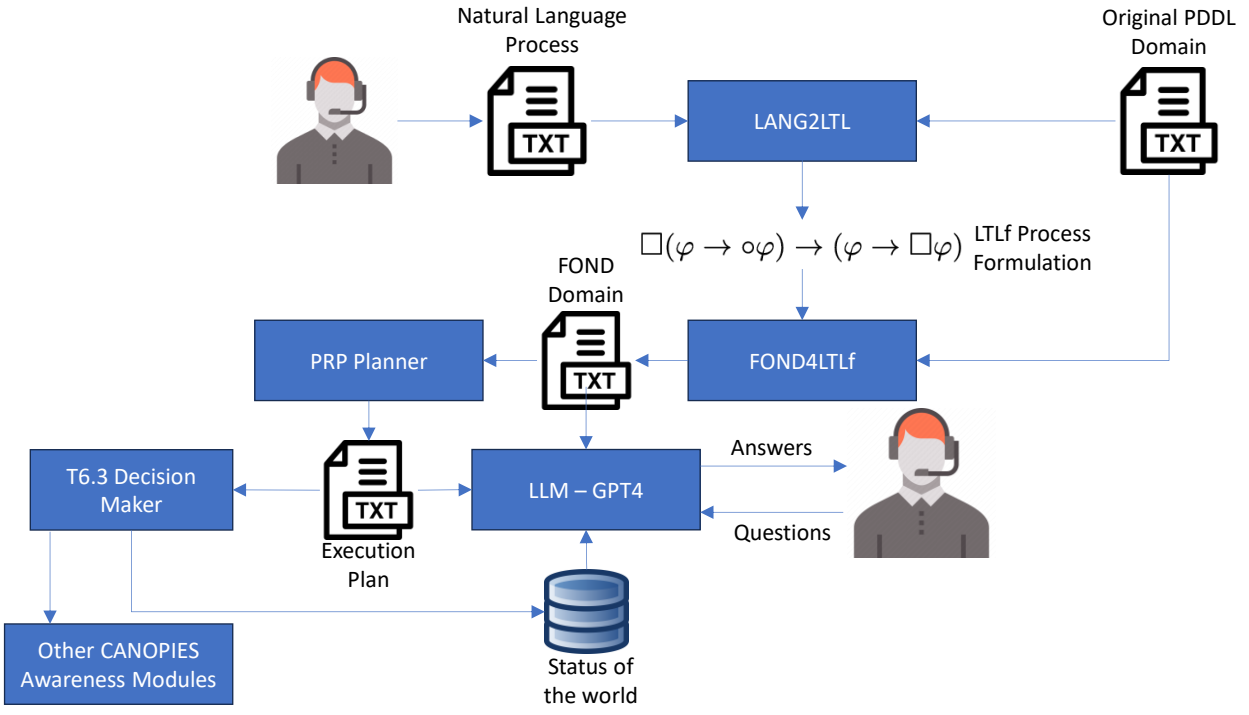

Figure 1 – The architecture of the Human-Robot awareness module developed in the context of Task 6.4.

We will now go deeper into details of the architecture depicted in Figure 1.

The system takes as input an utterance describing a high-level activity. This utterance is a string that can be obtained from the speech to text module implemented in the context of T6.3. This utterance is fed as input to a module (LANG2LTLf) in charge of converting the utterance in a LTLf formula using predicates taken from the planning domain (see details in Section 2.1). The LTLf formula and the planning domain are fed into a tool (FOND4LTLf) producing a new planning domain, which integrates the original one (that is a deterministic domain) adding predicates and (non-deterministic) actions thus allowing to support LTLf formula as goal of a planning problem. The new domain and problem are fed into a non-deterministic planner that produces a plan to be executed. Noticeably, the planner is a FOND – Fully Observable Non-Deterministic one, thus allowing to check the status of the system after the execution of a specific action in the plan. At this point, a LLM – Large Language Model (GPT-4 in our case) is used to turn the plan into a text and to answer questions, in natural language, about the execution of the plan. The plan (or execution policy) is also fed into the Decision-making module developed in T6.3. This module may in turn use other CANOPIES modules in order to update a knowledge base with the status of the world to be used as part of the prompt for the dialog with the human.

2.1 Converting speech into LTL specification of activities

The process of transforming sentences from natural language into a logical formalism, such as Linear Temporal Logic (LTL), is not a simple task. Firstly, one must identify expressions within the sentence that refer to the concepts which will be the symbols of the formula. This process is called "referring expressions extraction." Afterward, the rest of the sentence needs to be analyzed to understand the temporal relationship between these concepts and translate this relationship into an LTL formula, referred to as the "symbolic translation" Phase. Finally, if one intends to connect the sentence to a known logical domain, as in this case, each extracted concept must be associated with a specific symbol in the domain. This last Phase is termed "symbol grounding."

After a thorough examination of the systems documented in the literature, we conducted specific tests on [7] and [8]. Subsequently, we opted to use [8] for our task due to its modular design and its extensive training in recognizing a broader range of formula patterns. These features enable us to utilize the [8] tool on a different problem with minimal fine-tuning data, differently from the more extensive data requirements associated with [7].

The operation of Lang2LTL [8] can be summarized in three phases.

1. Referring expressions extraction:
    a. Input: Natural language sentence (e.g., "You cannot call the support robot without visiting the line 1 right before, and you cannot visit line 1 without calling the support robot right after that").
    b. Output: Set of sub-phrases referring to a concept (e.g., {"support robot", "line 1"}).

    This Phase is implemented using a chat-GPT prompt. Specifically, the task is explained to chat-GPT, along with a dozen examples (input-output). The model is then given a new sentence to process, and it responds with the desired output.

2. Symbolic translation:
    a. Input: "Symbolic" sentence, where referring expressions are replaced by placeholder symbols (e.g., "You cannot call the a without visiting the b right before; and you cannot visit b without calling the a right after that").
    b. Output: LTL formula using placeholders as symbols (e.g., G (b <-> X a)).[1]

    In this Phase, the original sentence is modified by replacing each referring expression with a placeholder symbol (such as a, b, etc.). The authors of Lang2LTL retrained certain chat-GPT models on the symbolic translation task with pairs of data (symbolic sentence, LTL formula) and made the trained models available. The fact that they replaced referring expressions with symbols before using the sentences for training makes this module fairly domain independent.

3. Symbol grounding:
    a. Input: LTL formula not grounded (e.g., G (b <-> X a)).
    b. Output: LTL formula grounded in the chosen domain (e.g., G (robot_at_rob_l1 <-> X call_support)).

    Finally, placeholder symbols need to be associated with symbols in the domain of interest. This is done using a "landmarks" file, where each landmark is a pair (identifier, information). The identifier is a domain symbol that can be associated with a placeholder symbol, and the information includes additional fields used for grounding. In this Phase, GPT is used again, mapping both the referring expressions extracted in Phase 1 and the information related to all landmarks into the latent space of the model. Each referring expression is then associated with the landmark that minimizes the cosine similarity between the two latent points. This Phase is more than a simple substitution as the expressions identified in Phase 1 are not, generally, corresponding to predicates of the planning domain. So, here, the LLM is used to associated expressions found in Phase 1 with terms that are meaningful for the planning domain.

In particular, to apply the tool to the CANOPIES project, we:

- Extend the prompt file used in Phase 1 because the original file is specialized in identifying concepts related to spatial locations (as [8] focuses on navigation tasks).
- Create a custom landmarks file for our PDDL domain in Phase 3, using domain symbols (predicates, constants, etc.) as identifiers and providing brief natural language descriptions of what each symbol represents.

[1] For the reader interested in the LTL syntax, please refer to [10]. In particular, the syntax used in this deliverable is the following: X – Next, WX – Weak Next, U – Until, F – Eventually, G – Globally. Also, the reader is invited to read Deliverable 8.2, where the same language is used to express multi-robot activities. For a full explanation of this example, see Section 4.

This project is funded by the European Union G.A No 101016906

2.2 Planning with activity specification as goal

After the application of LANG2LTL, as described in Section 2.1, we obtain an LTLf formula that describes the task that we want the robot to perform.

To formalize a planning procedure that takes into account such a formula as the ultimate goal of the agent, we devised a non-deterministic PDDL formalization of the vineyard domain. In this formalization, the domain has been discretized into several meaningful locations, and in these locations, there are grapes that can be ripe or not (or even unknown status if something went wrong) and that can be harvested or not, accordingly. In the domain we also formalized the presence of a support robot that can intervene when the harvesting box of the main robot is full. The actions we decided to include are the main ones that enable the most important robot activities like: moving in the vineyard, harvesting the grapes, calling the support robot and so on.

The main non-determinism comes from the state of the grapes.

The problem file of our PDDL description is very straightforward: we describe the objects that live in our domain and what is the initial state. The goal, instead, is not known a priori, but it changes dynamically, before providing the PDDL to the planner, depending on what is the LTLf formula that we want to satisfy.

After this formalization and description step, we need to feed these files into a planner to obtain a policy. However, this is not straightforward because of two reasons:

1. actions are non-deterministic.
2. the goal is not just a conjunction of fluents[2], as is usually done in automated planning, but a complete LTLf formula over them.

Therefore, we need to perform some translation steps in order to be able to plan in this domain. The first step is to eliminate the LTLf formula as a goal. In order to do this, we use the tool FOND4LTLf [9] which is a tool that compiles Fully Observable Non-Deterministic (FOND) planning problems with temporally extended goals (LTLf in our case) into classical FOND planning problems. The output of this step is an extended version of our original domain and problem file, which encapsulates the satisfaction of the LTLf formula in their specification, thus allowing us to have a classic conjunction of fluents as a goal.

Lastly, we want to obtain a policy for our planning task, and to do so we use the Planner for Relevant Policies (PRP) tool (https://github.com/QuMuLab/planner-for-relevant-policies), that takes as input the transformed files of the previous step.

After these processing steps, we can obtain the policy that takes into account the non-determinism and is able achieve our goal, while ensuring the satisfaction of the LTLf formula.

[2] In artificial intelligence, a fluent is a function changing value over time. A predicate is a type of fluent with Boolean values.

2.3 Querying the robot

After obtaining the policy, the robot acts accordingly in order to achieve the goal. In the context of awareness, we want to be able to understand what the robot is doing and why, and this should happen in the most natural way possible. Although it's feasible to track the problem's state space to determine the robot's current state and anticipate the next steps based on the unachieved parts of the goal, it's ideal for a farmer working with the robot to comprehend its actions in the simplest and most efficient manner possible. Therefore, we devised a Q&A mechanism based integrated with calls to LLM (specifically, GTPT4) that in a very natural way makes the robot explain to the user what it is doing, why, and what it is going to do afterwards.

The pipeline is the following:

1. the user engages with the robot and asks a question (e.g. “what is your next action and why”?).
2. using some off-the-shelf methods, this question is translated into a text string.
3. a query to GPT4 that contains the policy generated by the planner, the PDDL files and this question is made.
4. the response is extracted, translated to speech using again off-the-shelf method, and the answer is provided to the user.

With this procedure, we achieve awareness on robot actions, from the human perspective.

2.4 Environmental awareness of the robot and integration in the CANOPIES Ecosystem

From the point of view of the integration in the final system, the awareness module is integrated in the dialog manager developed in the context of T6.3. In particular, as a part of a dialog, the awareness module can be called to instruct the robot about a new high-level activity or to be queried about the current execution status of a previously submitted activity.

As discussed in Section 1, the CANOPIES project developed a set of tools, which already contribute to what can be defined Robot awareness. Just to provide few examples, all of WP4 and other WP6 tasks provides the robot with a set of functionalities contributing different facets of what can be defined environmental awareness (e.g., grape quality assessment, human action recognition, navigation in the environment). All of these functionalities are integrated in the awareness module developed in T6.4 thanks to the decision-making module, which keeps all the knowledge acquired about the environment and, if the information is not available, call the specific service.

Indeed, as described in the previous sections, the natural language specification is eventually grounded in a plan that is a sequence of basic actions available in the PDDL domain. These actions are then translated by the decision-making module defined in T6.3 to call services developed in the CANOPIES ecosystem that extract information (as mentioned above) and/or execute actions. These actions can be either very fine grained or can be considered themselves as activities made up by other actions. For these activities, it could be impractical to specify the details using natural language as a too much fine-grained detail would be needed. In these cases, a single PDDL action can be mapped by the decision-making module to a complex activity.

As an illustrative example, we propose in Section 3 such a type of activity, which is related to harvest all the grape bunches in a specific position. Results of this research have been recently submitted for publication [13]. Note that this activity is one of the crucial and more complex operation in CANOPIES and can be extended to the case of pruning.

3 Example of complex action (sub-activity): Harvesting all bunches in a specific position

So far, the developments in the projects mainly focused on perception, detection, estimation, and actuation, etcetera. However, often the robot will detect a lot of bunches, but it can only harvest one bunch at a time. Depending on the position and quality of the bunches, making good decisions on which bunches to harvest and in which order can make a significant difference in terms of performance. This performance is mainly measured by the quality of the harvested bunches and the time taken for the robot to do the harvesting. For example, a bunch of low quality could be in the way to harvest another bunch of high quality. Should the bunch of low quality be harvested in order to be able to reach the bunch of high quality as well? A significant part of the time taken for the robot to autonomously harvest table grapes is the time it takes to move the base with caterpillar tracks through the field. Therefore, in this example, we assume that the robot stands still at one position in the field and only considers the grape bunches within reach from there by only moving its top part (body and arms). Once the robot finishes its task at this position, it will move to the next position/tree and does the same there.

The state x can be described by a column in which every element b_i corresponds to one of the n bunches within reach of the robot:

$$x = [b_1, b_2, \dots, b_n] \qquad (1)$$

Every element b_i is a tuple containing information about that specific bunch:

$$b_i = (h_i, q_i, o_i, r_i) \qquad (2)$$

in which h_i indicates whether the bunch is harvested or not, q_i is the (expected) quality, o_i is a list of other bunches the bunch is occluded by, and r_i is a list of other bunches due to which it is not reachable.

The lists o_i and r_i for $i \in \{1, \dots, n\}$ can be used to create a graph representation (both for occlusions and reachability), providing the main structure to base the decision-making on.

3.1 Model

The state x_k has the form of the state x as described above, with the time indicator k. The action/decision u_k at time k is given by

$$u_k = \begin{bmatrix} u_{k,1} \\ u_{k,2} \end{bmatrix} \qquad (3)$$

in which $u_{k,1}$ is the index of the bunch to be harvested (-1 if no bunch will be harvested) and $u_{k,2}$ indicates whether to keep the bunch (1) or throw it away (0). The actions cause the state to change over time, according to

$$x_{k+1} = f(x_k, u_k) = \begin{cases} x_k & if\ u_{k,1} = -1 \\ x_k\ setting\ h_{u_{k,1}} = 1 & otherwise \end{cases} \quad (4)$$

3.1.1 Example

In this section, we will show an example of application of the algorithm devised for the harvesting sub-activity. Examples at the higher level are left to Section 4. Let us consider an example with 7 bunches. The quality of bunches 1, 3 and 5 are good enough to be harvested, for bunch 2 it would be better not to harvest it (yet) and for bunch 4 it is uncertain. Because of bunches being in the way for each other, first only bunches 3 and 5 are reachable for the robot. Both are of high quality and can be harvested. Let us assume that the robot first harvests bunch 3. The state could now for instance be:

$$x_k = \begin{bmatrix} 0 & 0.4 & \{2,3\} & \{2,3\} \\ 0 & 0.7 & \{3\} & \{3\} \\ 1 & 0.8 & \{\} & \{\} \\ 0 & 0.9 & \{2,6\} & \{2,6\} \\ 0 & 0.6 & \{\} & \{\} \\ 0 & 0.2 & \{1,2\} & \{1,2\} \end{bmatrix} \quad (5)$$

which is also depicted in Figure 2. Occlusions and reachability dependencies could be different from each other, and they also have a different influence on the optimal decision, but first we assume that they are the same.

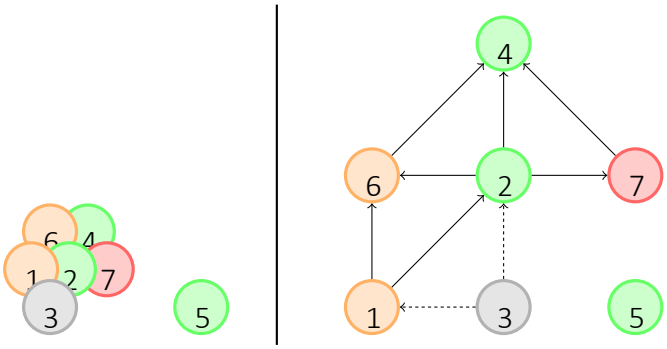

Figure 2 – Example reachability graph representation of bunches

At this point, there are five choices possible:

- Harvest bunch 1 and
  - keep it: $u_k = [1 \quad 1]^T$
  - throw it away: $u_k = [1 \quad 0]^T$
- Harvest bunch 5 and

- keep it: $u_k = [5 \quad 1]^T$
- throw it away: $u_k = [5 \quad 0]^T$

- Stop harvesting here: $u_k = [-1 \quad 0]^T$ (the value of $u_{k,2}$ is in fact irrelevant here)

As bunch 5 is not connected to any of the other bunches in the reachability graph, it forms its own group and can be considered separately. For the other group, the question is now whether bunch 1 should be harvested or not. In order to compare different choices, we define a reward (and/or cost) functions to assign scores. Then, the problem of finding the best sequence of action can be solved using Dynamic Programming.

3.2 Reward function

The reward function assigns a score to each action to enable a qualitative comparison between different (sequences of) actions. This function in general does not only depend on the action itself, but also depending on the state and hence we write $g(x_k, u_k)$. A simplified cost function for this application would be to only consider the quality of the harvested bunch:

$$g(x_k, u_k) = g_q(x_k, u_k) \qquad (6)$$

in which

$$g_q(x_k, u_k) = \begin{cases} 0 & if\ u_{k,1} = -1\ (stop) \\ q_{u_1} & if\ u_{k,2} = 1\ (keep) \\ 0 & if\ u_{k,2} = 0\ (throw\ away) \end{cases} \quad otherwise\ (harvest\ bunch\ u_{k,1}) \qquad (7)$$

Instead of only having a reward for the harvested quality, we also would like to take into account a cost for the time it takes to execute the actions.

3.2.1 Including time to harvest

In order to include the time it takes to harvest the bunches in the reward function, the state will include the pose of the robot arm. Then, the decision to harvest the next bunch comes with a cost (negative reward) related to the time it takes to:

- move the arm from the current position to the position to harvest the bunch $t_m(x_k, u_k)$
- cut the peduncle $t_h(x_k, u_k)$
- move the arm to the position to place the bunch in the box and place it there (if applicable) or to throw it away $t_r(x_k, u_k)$
- harvest the bunch if a human would have to do it $t_{hh}(x_k, u_k)$ (as a deduction of the cost)

In the resulting state, the new pose of the arm will be included (compared to (1)):

$$x = \begin{bmatrix} p \\ b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} \quad (8)$$

in which p contains all relevant information about the pose of the arm that is needed to calculate the time it will take to move. The new reward function of a single action becomes:

$$g(x_k, u_k) = g_g(x_k, u_k) + g_t(x_k, u_k) \quad (9)$$

in which g_t is the cost (negative reward) related to the time to do the movements and g_g is the reward related to the result of having harvested the bunch (can be the quality, but also the fact that a bunch of low quality that should be removed has been thrown away. The cost function related to the time is given by

$$g_t(x_k, u_k) = -\alpha t(x_k, u_k) \quad (10)$$

in which α is the cost per second and $t(x_k, u_k)$ is the time in seconds it will take:

$$t(x_k, u_k) = \begin{cases} 0 & if\ u_{k,1} = -1 \\ t_m(x_k, u_k) + t_h(x_k, u_k) + t_r(x_k, u_k) - t_{hh}(x_k, u_k) & otherwise \end{cases} \quad (11)$$

3.3 Dynamic programming for action selection

To find the best sequence, we apply dynamic programming. This is a recursive method that explores all options and find the optimal action sequence by recursively exploring the different options and optimizing the tail of the resulting problems. The total reward function, which also shows the recursive structure, is defined by:

$$J_k(x_k) = \min_{u_k} g(x_k, u_k) + J_{k+1}(f(x_k, u_k)) \quad (12)$$

with

$$J_h(x_h) = 0 \quad (13)$$

in which h is the horizon. In general, the horizon could be limited if the computational complexity of a problem is too high. In this application, a logical value for this horizon is the number of bunches, as there will (without future extensions) never be more actions than that.

The optimal action to take at time k is the minimizer of (12). Because of the recursive structure to solve the problem, the whole optimal sequence of action over the horizon can be stored. However, in most applications it is good practice to apply the first action and compute the optimal again at the new time step, either because the horizon shifts with time, and/or there might be disturbance or noise due to which the quality of the information based on which the decision was made is less accurate than the new information that is available.

4 Experiments

In this section, we show some experiments showing how the developed system applies to the CANOPIES context. We will show experiments about robot awareness in Section 3.1 and experiments about human awareness in Section 3.2.

4.1 Conversion of natural language into an LTFf formula

Example 1:

Situation to model: There is a specific location in the vineyard from which the support robot can be called, and the main robot can only go to that location to call the support robot (and not for other activities).

Target LTL formula: G(robot_in_special_location ↔ X robot_calls_support_robot)

Execution of Lang2LTL:

- Input sentence: "you cannot call the support robot without visiting the line 1 right before, and you cannot visit line 1 without calling the support robot right after that"
- Referring expressions found: {a="support robot", b="line1"}
- Symbolic sentence: "you cannot call the a without visiting the b right before; and you cannot visit b without calling the a right after that"
- Symbolic LTL: G (b <-> X a)
- Grounded LTL: G (robot_at_rob_l1 <-> X call_support)

Example 2:

Situation to model: When the robot detects that the bin is full of grapes, it must call the support robot afterward (it doesn't necessarily have to call it immediately; it can perform other operations before calling).

Target LTL formula: G (bin_is_full → F robot_calls_support_robot)

Execution of Lang2LTL:

- Input sentence: "after you detect the bin is full, you have to call the support robot later"
- Referring expressions found: {a="bin is full", b ="call the support robot"}
- Symbolic sentence: "after you detect the a; you have to b later"
- Symbolic LTL: G (a -> F b)
- Grounded LTL: G (full_b → F call_support)

Example 3:

Situation to model: When the robot detects that the bin is full of grapes, it must immediately call the support robot (it cannot perform any other operations before calling it).

Target LTL formula: G (bin_is_full → X robot_calls_support_robot)

Execution Lang2LTL:

- Input sentence: "if the bin is full, call the support robot immediately"
- Referring expressions found: {b=”bin”, a =”support robot”}
- Symbolic sentence: "if the b is full; call the a immediately"
- Symbolic LTL: G (b -> F a)
- Grounded LTL: G (full_b -> X call_support)

Example 4:

Situation to model: The robot can visit the second row of the vineyard only after visiting the first row.

Target LTL formula: ! robot_in_row2 U robot_in_row1

Execution of Lang2LTL:

- Input sentence: "Do not go to line 2 untill you visit line 1"
- Referring expressions found: {a=”line 2”, b =”line 1”}
- Symbolic sentence: "Do not go to a until you visit b"
- Symbolic LTL: !a U b
- Grounded LTL: ! robot_at_rob_l2 U robot_at_rob_l1

Example 5:

Situation to model: If the robot has found that the grape is unripe in row 1, then it should not visit row 1 anymore.

Target LTL formula: G (unripe_grape_in_row1 → X G (! robot_in_row1))

Execution of Lang2LTL:

- Input sentence: "if the first grape is unripe, don't come back to line 1"
- Referring expressions found: {b=”first grape”, a =”line 1”}

- Symbolic sentence: "if the b is unripe; don't come back to a"
- Symbolic LTL: G (b → X G (! a))
- Grounded LTL: G (unripe_g1 -> X G (! robot_at_rob_l1))

4.2 Querying the system for the current status of activity

Experiments for this part of the system were conducted by first employing the web interface of ChatGPT 4, which allows to upload files as part of the prompt (see Appendix A for an illustrative example). At a later stage, when the technique was integrated in the main pipeline, GPT4 API was used instead. To overcome the limitation of the API in terms of the possibility to upload a file, the content of the file is embedded in the JSON request.

Question:

> {"role": "system", "content": f"You are an harvester robot in a vineyard for the 'CANOPIES' project. The vineyard is formalised in PDDL. The domain file is {domain}. The problem file is {problem}. You act according to this policy {human_policy}. Your goal is to describe me what you are doing depending on what state you are. The description must be provided in terms of next action to do and explanation of why you are doing it. Ignore in the state description the turndomain() and not(turndomain()) predicates. Be very short when answering."},

Answer 1:

> I will execute the action of harvesting the ripe grape g2 at location l2 into the empty box b, because the preconditions are met, and it aligns with the next step in the policy to handle a ripe grape when the box is empty and the robot is free.

Answer 2:

> I will execute the action harvest_grape with the parameters rob, g2, b, and l2 in order to pick the ripe grape g2 and place it into the empty box b that I am carrying since I am in location l2 and the box is not full. This action will transition the system state by putting grape g2 in hand and marking me as not free.

Answer 3:

> I am going to harvest the ripe grape g2 because I have an empty box b and I am free to perform actions.

5 Future developments

Activities directly connected to Task T6.4 ended at month 36. Still, integration of developed functionalities, as well as further improvements, as part of T9.4 – Field Validation of the Human-Robot Interaction.

6 Bibliography

[1] Drury, J. L., Scholtz, J., & Yanco, H. A. (2003, October). Awareness in human-robot interactions. In SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483) (Vol. 1, pp. 912-918). IEEE.

[2] Dourish, P., & Bellotti, V. (1992, December). Awareness and coordination in shared workspaces. In Proceedings of the 1992 ACM conference on Computer-supported cooperative work (pp. 107-114).

[3] Drury, J. L. (2002). Extending usability inspection evaluation techniques for synchronous collaborative computing applications. University of Massachusetts Lowell.

[4] Gutwin, C., Stark, G., & Greenberg, S. (1995). Support for workspace awareness in educational groupware.

[5] Gutwin, C., Greenberg, S., & Roseman, M. (1996). Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation. In People and Computers XI: Proceedings of HCI'96 (pp. 281-298). Springer London.

[6] Ziparo, V. A., Iocchi, L., Lima, P. U., Nardi, D., & Palamara, P. F. (2011). Petri net plans: A framework for collaboration and coordination in multi-robot systems. Autonomous Agents and Multi-Agent Systems, 23, 344-383.

[7] NL2LTL – a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas. Francesco Fuggitti, Tathagata Chakraborti. AAAI 2023.

[8] Grounding Complex Natural Language Commands for Temporal Tasks in Unseen Environments. Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, Ankit Shah. Arxiv 2023.

[9] De Giacomo, G., & Fuggitti, F. (2021). FOND4LTL: FOND Planning for LTL//PLTL/Goals as a Service. ICAPS 2021 Demos.

[10] De Giacomo, G., & Favorito, M. (2021, May). Compositional approach to translate LTLf/LDLf into deterministic finite automata. In Proceedings of the International Conference on Automated Planning and Scheduling (Vol. 31, pp. 122-130).

[11] Kaszuba, S., Sabbella, S., Leotta, F., Nardi, D. (IN PREPARATION) Multimodal Human-Robot Interaction in Precision Agriculture

[12] Argenziano, F., Umili, E., Leotta, F., Nardi, D., (IN PREPARATION) Human-Robot Awareness in Precision Agriculture

[13] Beumer, R., Saraceni, L., Ciarfuglia, T., Nardi, D., (IN PREPARATION) Using dynamic programming to optimize harvesting in Precision Agriculture

7 Appendix A – Experiments with ChatGPT4

In this section, we show how it is possible to prompt engineering the famous chatbot ChatGPT4 to serve to the goal of task T6.4. In particular, we show one of the preliminary experiments executed in order to prove LLM could be used in order to assess the suitability of LLMs to provide a useful tool to provide human awareness. The experiments correspond of a single chat consisting of a series of questions and answers.

Question:

domain.pddl

File

human_policy.pol

File

problem.pddl

File

I am going to provide to you several files: "domain.pddl" and "problem.pddl" are domain file and problem file in PDDL formalism, and file "human-policy.pol" is the policy obtained to achieve that goal. Describe me in maximum 5 lines of text what is the objective that the policy is trying to achieve. When you answer me, I do not want that you describe me the content of the file I passed you, nor that you try to explain what they could me. Just give me in 5 lines of text the objective the objective the policy is trying to achieve.

Answer:

My goal is to ensure that all locations (like l3) are cleared, which likely involves tasks such as checking, harvesting, and managing grapes (ripe, unripe) at various locations, moving between these locations, and possibly handling exceptions or uncertainties. I can work with support elements like another robot (support0) and interact with objects like grapes (g0, g1, etc.) and a box (b). The policy is designed to achieve these tasks efficiently and effectively within the given vineyard setup.

Question:

If I tell you that you are in location l1 and you have checked the grape in that location and has found out that the grape is ripe, what are the next 3 actions that you will do according to the policy?

Answer:

Determining my next actions based on the current state (me in location l1, grape in l1 is ripe and checked) from the policy file is not feasible with my current capabilities[3]. Extracting and interpreting specific rules from a policy file, especially one designed for a PDDL (Planning Domain Definition Language) environment, requires parsing and logic inference beyond what I can perform.

To find out the next actions according to the policy, you would typically need a PDDL solver or planner that can interpret the domain, problem, and policy files, and then simulate or deduce the actions based on the current state of the system. This process involves complex logic and decision-making algorithms specific to PDDL and the particular domain of the problem.

Question:

Consider the problem file, what is the initial state?

Answer:

The initial state of the problem, as defined in the "problem.pddl" file, includes the following conditions:

1. I am a robot named rob located at l0.
2. A support robot, support0, is present.
3. Grapes g0, g1, g2, and g3 are located at l0, l1, l2, and l3 respectively.
4. Locations l0, l1, l2, and l3 are all initially unchecked.
5. I am free.
6. A box b is in possession of the robot rob.
7. Adjacencies are established between locations: l0 adjacent to l1, l1 to l2, and l2 to l3.
8. The box b is full.

This setup represents the starting point for my task in the vineyard as per the problem's scenario.

Question:

What are the next 2 actions that you will do in order to move towards reaching the goal?

Answer:

[3] Here the unfeasibility response from ChatGPT is only due to an incorrect prompting. The rest of the example shows how this can be corrected by fine-tuning the prompt

Based on the initial state described in the "problem.pddl" file and the domain details from the "domain.pddl" file, my next two actions will be to move towards achieving the goal would typically follow the logical sequence of actions defined in the domain. Given the goal is to clear a location (specifically l3 in this case), and considering the I start at l0, the sequence might be:

1. Move Action: I would need to move from its current location (l0) to an adjacent location to eventually reach l3. Since l0 is adjacent to l1, the first action would likely be moving from l0 to l1. This is based on the move action defined in the domain, which requires me to be at the starting location and that the destination is adjacent.
2. Check or Harvest Action: Once I reach l1, it may need to perform actions relevant to the grapes, such as check_grape (to determine if the grapes are ripe or unripe) or harvest_grape (if it finds the grapes ripe). The choice between checking and harvesting depends on the specifics of the domain's logic and the current state of the grapes (ripe or unripe) at l1.

These actions align with the domain's definitions and the goal of clearing l3, assuming typical PDDL logic and the constraints provided in the domain and problem files.

Question:

After the Move action, in which state will you result according to the policy?

Answer:

Based on the "domain.pddl" and "problem.pddl" files you provided, and considering I perform the Move action from l0 to l1, the resulting state will be as follows:

1. Robot's Location: I will now be at location l1.
2. Grapes' Locations: The locations of the grapes remain unchanged: g0 at l0, g1 at l1, g2 at l2, and g3 at l3.
3. Unchecked Locations: If the Move action does not inherently change the checked status of locations, then l0, l2, and l3 remain unchecked. Whether l1 remains unchecked depends on whether moving to a location automatically checks it, which isn't specified in the provided details.
4. Robot's Status: I remain free, assuming the Move action does not change this status.
5. Box Status: The box b, initially full and with the robot, remains in the same state unless the Move action inherently changes its status.

This state assumes that the Move action solely changes the robot's location without affecting other predicates, such as the status of grapes or the box. The actual change in

state could vary if the domain's actions have additional effects not detailed in your summary of the files.

Question:

new-domain.pddl

File

new-problem.pddl

File

consider the modified files I give you, If I tell you that you are in location l1 and you have checked the grape in that location and has found out that the grape is ripe, what are the next 3 actions that you will do according to the policy?

Answer:

The updated "new-domain.pddl" and "new-problem.pddl" files describe a slightly different scenario compared to the initial files. The goal and the actions available in the domain have been modified. However, without a specific policy file or a PDDL solver, I can't definitively state the next three actions I will take.

In a general sense, if I am at location l1 and has checked a grape, finding it ripe, the logical sequence of actions might involve:

1. **Harvest Grape**: If the grape is ripe and the robot is free, I may perform a harvest_grape action to harvest the grape.
2. **Move**: After harvesting, I might move to the next location (l2), following the adjacency rules.
3. **Check Grape**: At the new location, I would likely check the grape status, similar to what it did at l1.

These actions align with typical PDDL logic but may not precisely match the specific policy or sequence dictated by your system, especially considering the added complexities like turnDomain and q1, q2 predicates in the updated domain. For accurate action sequencing according to your specific policy, a PDDL solver that can interpret these files and the current state is required.

Question:

human_policy.pol

File

this is your specific policy file, If I tell you that the you are in location l1 and he has checked the grape in that location and has found out that the grape is ripe, what are the next 3 actions that you will do according to the policy?

Answer:

The content of the "human_policy.pol" file appears to be a mapping of variables to specific states and conditions within the PDDL framework. However, determining the my next three actions based on a specific state (me in location l1, grape in l1 is ripe and checked) directly from this policy file is not straightforward. The policy file provides mappings, but it doesn't explicitly lay out the decision-making process or the sequence of actions.

To ascertain the next steps according to the policy, one would typically need a PDDL solver or an interpreter that can process these mappings in conjunction with the current state of the environment as described in the domain and problem files. Such a solver would use the policy mappings to determine the optimal or prescribed sequence of actions to transition from the current state towards the goal state.

In a typical PDDL setup, after checking a ripe grape at l1, my actions would depend on the defined goals, the preconditions, and effects of available actions in the domain, and how these interact with the current state of the environment. Common actions in such scenarios might include harvesting the grape, moving to another location, or performing other tasks related to the grapes or the environment, as defined in the domain. However, without processing the policy file with an appropriate PDDL tool, I cannot provide the exact sequence of actions.

Question:

human_policy.pol

File

try with this policy instead

Answer:

The new "human_policy.pol" file outlines a policy with specific conditions and corresponding actions. This policy is more straightforward in its structure, detailing specific scenarios and the actions to execute in each case.

Based on the current scenario where the I am in location l1, and it has checked the grape in that location and found out that the grape is ripe, the next actions according to this policy would depend on the conditions and the actions defined in the policy for such a scenario.

Given the snippet of the policy file you've provided, it seems that the policy dictates actions based on various conditions like my location, the status of grapes, and the state

of locations (cleared, unknown, etc.). However, without the specific conditions related me being at l1 with a ripe grape, it's challenging to determine the exact next three actions.

The actions listed in the snippet involve different scenarios (like me at l3, handling exceptions, assessing unripe grapes, etc.), which may not directly apply to the current state at l1. To provide an accurate answer, I would need to see the specific part of the policy that deals with the scenario of me being at l1 with a ripe grape. Without this information, any suggestion about the next actions would be speculative.

Question:

if the following scenario holds: robot-at(rob, l2), ripe(g2), free(rob), what are the possible actions that can be applied by you according to the policy=

Answer:

Given the scenario where the robot (rob) is at location l2, grape g2 is ripe, and I am free, we need to look at the policy to determine the possible actions that can be applied. Based on the structure of the policy file you've provided, actions are dictated by specific conditions.

From the snippet of the policy provided earlier, the actions in the policy are defined for particular conditions. However, the snippet does not include conditions directly matching the scenario you described (me at l2, ripe g2, me free). To accurately identify the possible actions for this specific scenario from the policy file, I would need to see the part of the policy that deals with these exact conditions.

Typically, in a PDDL context, given these conditions, likely actions might include:

- Harvest Grape: If the domain allows, and there's a ripe grape (g2) at my location (l2), and I am free, an action like harvest_grape could be applied.
- Move: I might move to another location if required by the overall goal or subsequent tasks.

However, without the relevant part of the policy that matches the given scenario, it's not possible to definitively state which actions from the policy would be applicable. The actual actions would depend on how the policy has been designed to handle such situations.

Question:

User if the following scenario holds: robot-at(rob, l3), unripe(g3), q1(l3), turndomain(), what are the possible actions that can be applied by you according to the policy?

Answer:

Based on the snippet of the "human_policy.pol" file you provided and the scenario described (robot rob at l3, grape g3 is unripe, q1(l3) holds, and turndomain() is true), we can identify a possible action from the policy:

- Assest Vine: If the condition is robot-at(rob, l3), the grape g3 at l3 is unripe, q1(l3) holds, and turndomain() is true, then the action assest_vine rob g3 l3 can be applied. This action likely involves me assessing or managing the vine at location l3.

This action aligns with the policy structure where specific actions are tied to certain conditions. It's important to note that the actual implementation or effect of the assest_vine action would depend on its definition in the domain file.

Question:

Describe me in maximum 3 lines what the next action will be and why in natural language, without mentioning variable or predicates or saying anything else.

Answer:

In the given scenario, the I will assess or manage the unripe grape at its current location. This action is chosen because the policy dictates that when encountering an unripe grape at a specific location, my task is to evaluate or handle the vineyard's condition at that spot.

Question:

Imagine I'm a human that is asking to you "what are you doing?". Describe me in maximum 3 lines what your next action will be and why in natural language, without mentioning variable or predicates or saying anything else.

Answer:

I'm currently assessing the vine here to ensure it's growing properly and to determine the right care it needs. This is part of my routine to manage the vineyard's health and productivity.