LEAF: Using Semantic Based Experience to Prevent Task Failures

Nathan Ramoly, Hela Sfar, Amel Bouzeghoub, and Beatrice Finance

Abstract Using service robots at home is becoming more and more popular in order to help people in their life routine. Such robots are required to do various tasks, from user notification to devices manipulation. However, in such complex environments, robots sometimes fail to achieve one task. Failing is problematic as it is unpleasant for the user and may cause critical situations. Therefore, understanding and preventing failures is a challenging need. In this paper, we propose LEAF, an experience based approach to prevent task failure. LEAF relies on both semantic context knowledge through ontology and user validation, allowing LEAF to have an accurate understanding of failures. It then uses this new knowledge to adapt a Hierarchical Task Network (HTN) in order to prevent selecting tasks that have a high risk of failure in the plan. LEAF was tested in the Hadaptic platform and evaluated using a randomly generated dataset.

1 Introduction

Nowadays, we are facing an emergence of use of robots and smart homes. As there is a growing need for domestic health-care, in particular for elderly people, service robots provide a welcomed help. During their everyday routine, such robots perform various tasks, from reminding the user to take his/her medicine to using some devices. However, in home environment, multiple problems can block the robots' task plan. For example, it may encounter a breakdown or have to difficulty to prop-

Nathan Ramoly, Hela Sfar, Amel Bouzeghoub
SAMOVAR, Telecom SudParis, CNRS, Paris-Saclay University, Evry, France
e-mail: name.surname@telecom-sudparis.eu

Beatrice Finance
DAVID laboratory, University of Versailles Saint-Quentin-en-Yvelines, Versailles, France
e-mail: beatrice.finance@uvsq.fr

erly understand the context. In a nutshell, the robot is likely to encounter Failure Situations in its plan.

Overcoming failures is a common problem in robotics. Most works tackle this issue by reacting to it [17, 8]. In those cases, whenever the robot fails, it understands the cause and tries to find an alternative solution by generating a new plan. However, doing so is energy and time consuming [6] and it delays the reach of the goal. By proactively avoiding a failure instead of reacting, the robot can be quicker and more efficient to reach its goal, which is essential for domestic health-care application. To do so, the robot has to understand the cause of the failure and adapt its planning when it encounters them again. Some works have addressed this issue [9, 16], but the constraints of the home environment induce more challenges. For such application, it is essential to satisfy the user needs and to consider various and highly semantic data.

In this paper, we propose a solution to learn failure causes, evaluate them and prevent further failure called LEAF (Learning, Evaluating and Avoiding Failures). LEAF aims to learn failures' causes from previous encountered situations in order to prevent repeating them in future plans. It uses reasoning on semantic based context knowledge as well as a user validation to ensure the efficiency and accuracy of cause identification. LEAF also adjusts the planning phase to generate failure-free plans.

The main contributions of the paper are as follows:

1. A semantic model to represent and store situations that enables reasoning and interoperability with other systems.
2. A method to extract the causes of failures that includes the user in the loop to guarantee the quality of causes identification.
3. An improvement of HTN planner that takes into account the detected causes and to select safer sub-plans (i.e. avoiding task failures).

The remaining of the paper is divided as follows. Section 2 illustrates our needs through a scenario. Section 3 reviews the related works and points out their limits. Section 4 presents definitions and notions required for the understanding of our work. The contributions are described in Section 5, while the experiments are addressed in Section 6. Finally Section 7 concludes the paper.

2 Motivating scenario

In order to motivate our proposal, let us consider this scenario:
Scenario 1: Nono, a service robot, operates in the home of a user named Katleen. Her house is equipped with various sensors, including motion sensors, microphones and Katleen's smart phone. One of the main role of Nono is to remind Katleen to take her medicine. Based on a schedule, the robot decides when it must vocally alert Katleen. Whenever it has to do so, it generates a plan to achieve this task. Nono has two options to alert Katleen. Firstly, if it knows the location of Katleen, it can go directly to her and talk to her. This is the prior solution as it is direct and has

impact on the user. Secondly, if it doesn't know her location, it can send a message to her phone. For both cases, the user is expected to provide an acknowledgement. However, sometimes Nono tries but fails to vocally alert the user. This can be due to various causes. Let us consider two examples of failure situations: (1) Katleen is listening to music through her phone with headsets. Therefore, she may not hear the voice of the robot; (2) the room may be filled with noise, for example from television or phone discussions. Hence, if the volume of the robot is set to a low value, Katleen may not hear Nono. After executing several times the task 'vocal alert', Nono obtains a history of situations for this task, represented in Table 1.

Sit.	Status	Start	End	Observed context data (**failure causes are bold**, *inferred data are italic*)
S_1	Failure	18/03/17 18:30:50	18/03/17 18:40:55	(katleen isLocatedIn livingroom), **(katleen isDoing music)**, (livingroom hasSoundLevel 5db), (nono hasVolumeLevel 30db)
S_2	Failure	18/03/17 20:10:30	18/03/17 20:15:30	(katleen isLocatedIn livingroom), (katleen isDoing tv), (livingroom hasSoundLevel 75db), (nono hasVolumeLevel 30db), ***(katleen vocUnreachTo nono)***
S_3	Success	19/03/17 10:30:50	19/03/17 10:34:15	(katleen isLocatedIn livingroom), (katleen isDoing tv), (livingroom hasSoundLevel 40db), (nono hasVolumeLevel 30db)
S_4	Failure	19/03/17 11:00:50	19/03/17 11:03:55	(katleen isLocatedIn livingroom), **(katleen isDoing music)**, (livingroom hasSoundLevel 20db), (nono hasVolumeLevel 30db)
S_5	Success	19/03/17 15:06:35	19/03/17 15:08:55	(katleen isLocatedIn bedroom), (katleen isDoing reading), (livingroom hasSoundLevel 25db), (nono hasVolumeLevel 30db)
S_6	Failure	20/03/17 10:15:50	20/03/17 10:18:00	(katleen isLocatedIn livingroom), (katleen isDoing phoning), (livingroom hasSoundLevel 65db), (nono hasVolumeLevel 30db), ***(katleen vocUnreachTo nono)***

Table 1: History of task 'vocal alert'

The aim of this work is to prevent the failure situations. To achieve this, we aim to identify causes, using history, in order to prevent task execution when failure causes are observed again. In scenario 1, Nono shall be able to identify context data (katleen isDoing music) (katleen vocUnreachTo nono) as failing causes for the task 'vocal alert'. Thus, if a situation $S =$ (katleen isLocatedIn livingroom), **(katleen isDoing music)**, (livingroom hasSoundLevel 20db), (nono hasVolumeLevel 30db) occurs, the robot would understand that the task 'vocal alert' is likely to fail, thus would opt for the text message solution. By doing so, Nono avoids wasting time going to the user and trying to alert Katleen.

3 Related work

Understanding and explaining plans and failures is an active research direction. In this section, we discuss some of the works that are related to our proposition.

The work proposed by Hanheide et al. [8] aims to provide a global planning solution for robots operating in an open and uncertain environment, such as a smart home. They address the issue of explaining task failure. To do so, they compare the actual and expected context observations. The idea is to discover what particular unexpected data caused the failure of the task. To do so, the robot generates a dedicated plan and relies on a diagnostic knowledge. Afterwards, the robot is able to make a new plan that avoids the identified causes. Although this technique is able to adapt the plan by understanding the missing elements, it does not learn from previous failures in order to prevent the occurrence of future failures.

The proposition of Sariel and Kapotoglu [16, 9] shares similar objective from our proposal. To the best of our knowledge, it is the main work about learning causes from previous experiences to prevent future failure. In this approach, based on previous failed situations, the robot is able to determine failure causes and avoid using tasks that are expected to fail. To do so, the authors use Inductive Logic Programming (ILP), an experiential learning framework that builds an experience by deriving hypothesis from failure situations. Context observations are stored and labeled as success or failure. The hypotheses are then adjusted and associated with a probability. A low probability implies there was a lot of ambiguity. With these hypotheses, the planning process is adjusted to prevent future failing situations. This technique improves the efficiency of planning for the object manipulation case study.

However, such a solution faces numerous issues. First, this solution relies on a simple model that does not include any reasoning. Reasoning would allow to infer further data from already acquired context data. For instance, in scenario 1, if the volume difference between the robot and the ambient noises is too low, the user can not be vocally reached. Sariel's solution however, does not consider 'relations' between context data as possible failure. Secondly, depending on the situations available in the history, this solution can be biased. For example, it may identify two context data as cause while only one actually explains the failure. Lastly, by encountering redundant, yet non related to failure, context data, the solution of Sariel et al. can identify wrong causes of failure.

To overcome these limitations, we propose a system, named LEAF, to identify failures. Firstly (i), LEAF relies on an ontological representation, that enables reasoning on context data. Furthermore (ii), it considers each context data independently to prevent bias. In fact, by doing so, non related context data are not associated to potential real cause, as it is in [9, 16]. Indeed, if a situation can be explained by multiple context data, each of them is considered as a potential cause. Moreover, (iii) it relies on user validation. By relying on user's feedback, our solution ensures the quality of the identification of the failure causes. Finally, (iv) it provides an adjustment of the HTN planner to take into account the causes in the planning process. The overall proposition is detailed in Section 5. Additionally, the next section introduces some background and definitions for a better understanding.

4 Background

4.1 Ontology

An ontology is generally defined as representation of a shared conceptualization of a particular domain. It can easily be shared across people and application systems. It relies on the Resource Description Framework (RDF) [11]. An ontology is described through a set of RDF triples (*subject, predicate, object*). The set of all triples can be seen as a graph where nodes are concepts or instances and vertices are predicates

among them. Using ontologies has multiple benefits, such as the inference of further data. In our work, we use the ontology in order to model Context Data (CD, Definition 1), Situation (ST, Definition 2), and Causes of Failure (CE, Definition 3).

4.2 Definitions

In this section, we define the concepts required for comprehension of our work.

Definition 1 CONTEXT DATA (CD): *a context data is a piece of information about the environment provided by robots' sensors, environment's sensors (smart devices) or a knowledge base. Formally, it is a 4-tuple (subject, predicate, object, t). subject is an entity of the environment (i.e. user, robot or thing (physical object)), object is a context data about the subject, predicate is the relation between the subject and the object, and t is the timestamp of observation of this contextual data.*

A CD is modeled as a RDF triple annotated with t. We distinguish two types of CD: High level CD and Low level CD. Low level CD are numeric observables generated from sensors; while high level CD are symbolic observables at the appropriate level of abstraction to make sense.

Property: An *activity* is a high level CD where *subject* can be either a user or a *robot* and *predicate* is equal to "isDoing".

Example 1: In Table 1, for S_1, (katleen isDoing music) is a high level context data obtained through a complex activity recognition process. While (katleen isLocatedIn livingroom) is a low level context data given by, possibly, a single sensor.

Definition 2 SITUATION (ST): *A situation is a set of CD that have occurred at a given time interval. In this work, we are interested in the situation during one robot's task. A situation can be seen as a 'snapshot' of the state of the environment during one task. Formally, we define a situation as a 5-tuple* $(\{CD_i\}, status, task, t_s, t_e)$. $\{CD_i\}$ *is a set of context data captured during a time interval* $[t_s, t_e]$*; task is the task the robot was doing during this situation; status is the outcome of the execution of the task, it is set to "null" when the situation is created and is set either to "success" or "failure" after the task execution;* t_s *is the start time of the execution of the task and* t_e *is the end time of its execution. In the rest of the paper, we refer to situations as either failure situation or success situation when their status are respectively equal to failure or success.*

Definition 3 CAUSE (C): *A cause is a CD that fully or partially explains a failure situation. Formally, we define a cause as a couple* (CD, ST) *where CD is the context data that causes the failure of the situation ST.*

All observed situations are stored in an history H. We denote the history for one task t as H_t. We denote all the causes of one task t as C_t. Each task relies on an ontology O_t that includes both H_t and C_t.

Example 2: let us consider this situation: $(\{(katleen, isLocatedIn, kitchen, 15:01:10), (katleen, isDoing, music, 15:3:44)\}$, Failure, Alert, 15:00:52, 15:03:00). This is a failure situation caused by the 'music' activity of the user.

4.3 Planner

In this work, we rely on the Hierarchical Task Network (HTN) planner. HTN aims to find a *solution* to a *planning problem* by decomposing *tasks* into *subtasks*. HTN relies on two types of *tasks*: *primitive tasks* and *compound tasks*. *Compound tasks* are realized by *subtasks*, while *primitive tasks* are 'ready-to-run' non-decomposable tasks. The result of the planning process is called *solution*, and is a totally ordered set of *primitive tasks*. A *method* indicates how to decompose a compound task a sequence of *subtasks*, primitive or compound, based on preconditions. HTN planning consist in selecting a method for each compound task. The selection is perform by checking the validity of method's preconditions. For example in scenario 1, if Katleen's location is known, meaning there is a predicate (katleen isLocatedIn room), the robot can use the 'vocal alert' branch. For a more detailed overview and definition of HTN please refer to dedicated works [5]. HTN was selected for various reasons. Firstly, it offers good performance by having a reduced search space compared to other planners, such as STRIPS-like solutions [3]. Moreover, it is a popular planner as it is used for several applications [14, 5], particularly in robotics [17, 10, 15, 13]. Furthermore By using HTN, LEAF has the possibility to be easily integrated with some of these works.

5 Proposition

In this section, each component of our contribution, LEAF, is presented. It relies on three independent steps, depicted in Figure 1, namely:

1. **Acquiring Situations**: The robot acquires situations and stores them in the history whenever a task ends.
2. **Extracting Failure Causes**: Causes are extracted based on the history and validated by the user.
3. **Enhancing Planner**: The extracted causes are then taken into account to enhance the planning process through an upgraded HTN.

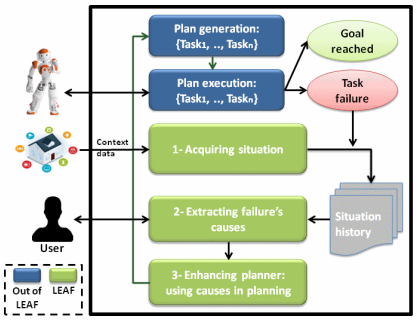


Fig. 1: The architecture of LEAF

5.1 Acquiring situations

The first step of our approach is to acquire the current situation. Whenever the robot finishes a task t, it stores the corresponding situation in an ontology. This ontology was designed from a previous work [1] and enhanced with new concepts and relations, such as *Activity*. The observed CD are inserted in the ontology under the Context Data concept. An exemplary representation of a subset of a situation can be found in Figure 2. The property of the *Situation* concept are also stored in the ontology (not represented in Figure 2).

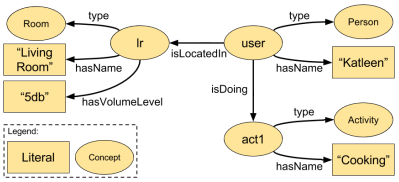


Fig. 2: Ontological representation of a subset of S_1

Once the situation is acquired, a rule based reasoning is applied to infer new data. It is essential to have a complete understanding of the context. These rules are provided by an expert. For example, let us consider situation S_2 in Table 1 where (katleen vocUnreachTo nono) is inferred by applying the following rule using Jena's formalism:

ruleVocUnreachable: (?user isLocatedIn ?room) $\wedge$ *(?room hasSoundLevel ?sndLvl)* $\wedge$ *(?robot hasVolumeLevel ?robotLvl)* $\wedge$ *difference(?sndLvl ?robotLvel ?soundDiff)* $\wedge$ *greaterThan(?soundDiff 25)* $\rightarrow$ *(?user vocUnreachTo ?robot)*

This rule states that if a user is in a room that is too loud for the current speaking volume of the robot, then the user cannot be reached vocally by the robot.

After applying these rules, more contextual data are generated. The enriched situation is then stored in the history H.

5.2 Extracting failing causes

Whenever a task fails, the robot tries to identify the causes of the failure.

In our approach, based on the history of situations and the previous identified causes, the robot identifies possible causes and asks for confirmation from the user. Through the involvement of the user in the process, we aim to provide high quality learning that is compliant with the user. The process of extracting the failing causes is composed of three steps: (1) Selecting data to be validated by the user: the robot selects context data (possible causes) that are to be validated by the user according to to the robot's current needs for the learning process. (2) Requesting user validation: the robot requests the user to provide validation about the selection of context data. (3) Getting user feedback: the robot receives the user feedbacks and updates its knowledge accordingly. The following subsections are reviewing each steps.

[0] https://jena.apache.org/documentation/inference/

5.2.1 Selecting context data to be validated by the user

In order to build its experience, the robot requires user validation concerning possible causes. The first step is to identify what are the possible causes to be validated. In other words, the objective is to extract some context data that may are possible causes of the failure. This selection is important to quickly identify the causes without wasting user's validations. We consider that the robot should ask only a few validations from the user in order not to disturb him/her. This selection of causes has two modes: cold start and warm process. They are described below.

Cold start:
LEAF is subjected to the problem of cold start or cold boot. In fact, despite the main process relying on the history and previously identified causes, it also needs a procedure to start the learning process without any prior knowledge. Thus, the selection of context data to be validated is particular for the first encountered situations. The cold start procedure is applied once a minimum number of failing situations are encountered to a task t. In our experiments, we launched the procedure after 3 failure situations. The principle of the cold start procedure is to compute a 'causality' score for each context data based on its occurrence in the situations belonging to the history of the task. For one given context data cd, the 'causality' score $score_{cd}$ is computed as expressed in Equation 1.

Let $Ssucc_{cd}$ be the set of all successful situations in the history H_t that contains the context data cd: $Ssucc_{cd} \subset H_t$, $\forall$ S $\in Ssucc_{cd}$ where $cd \in S$. Let $Sfail_{cd}$ be the set of all failure situations in H_t: $Sfail_{cd} \subset H_t$, $\forall$ S $\in Sfail_{cd}$ where $cd \in S$. Consequently, $\mid Ssucc_{cd} \mid$ and $\mid Sfail_{cd} \mid$ are respectively the number of success situations and the number of failure situations containing the cd, and $\mid Sfail_{cd} + Ssucc_{cd} \mid$ is the total number of situations in H_t containing cd. Finally, the $score_{cd}$ is computed as follows:

$$score_{cd} = (\mid Sfail_{cd} \mid - \mid Ssucc_{cd} \mid)/(\mid Sfail_{cd} + Ssucc_{cd} \mid) \quad (1)$$

For example, let us suppose that the robot has just encountered the situation S_4 in Table 1 and has previously encountered S_1, S_2 and S_3. The context data cd_1 =(katleen isLocatedIn livingroom), by applying Equation 1, will have the causality score: $score_{cd_1} = (3-1)/(3+1) = 0.5$, since there was 3 failure situations and 1 success situation. On the other hand, the piece of context data cd_2 =(katleen isDoing music), which is a failure cause for the current situation, will have a causality score: $score_{cd_2} = (2-0)/(2+0) = 1$.The context data with the highest causality scores are then selected to be checked by the user. In the previous example, cd_2 would be selected over cd_1 for validation.

Warm process:
Once the cold start procedure is executed, the robot has some initial experience and can use the new knowledge to identify new causes. Whenever the robot is facing a failure situation, it has two options for selecting context data to be validated as a cause by the user: either already identified causes or not registered context data that may be novel causes. The first option consolidates its knowledge while the other allows to explore new possible causes. It important to remember that the number of validations performed by the user is limited. Therefore, this process can be de-

scribed as a **multi-armed bandit problem** [12]. Multi-armed bandit solvers aim to maximize reward by efficiently choosing between exploration, i.e. using resources to explore new possibilities of gain, or exploitation, i.e. ensure gain by using resources from reliable sources. In our case of study, checking an already encountered cause corresponds to the exploitation phase, while checking a new possible cause corresponds to the exploration.

Our proposition is to use a multi-armed bandit approach to select the cause to be asked to the user. The selection of the strategy to choose depends on the context. In fact, if the robot is used to succeed the execution of a particular task and it fails to achieve this task for the first time, this means that probably there is a new cause of failure. Hence, the robot should prioritize exploration of a new cause. On the other hand, if the robot fails multiple times in a row, it implies the robot's current knowledge of causes is not accurate, hence the robot should focus on adjusting its knowledge and exploit.

In this work, we are using a variation of R-UCB[2], that is an improvement of the Upper Confidence Bound (UCB) algorithm[4]. UCB is a well known solution for tackling multi-armed bandit problems. It allows to select the context data with the higher upper confidence bound when exploiting. In other words, when the robot observes multiple causes, it selects the most relevant one by applying the following formula described in Equation 4. In UCB, the selection between exploration and exploitation is performed randomly by following a fixed rate. R-UCB improves the UCB by adapting the exploration/exploitation rate according to the current 'risk'. In R-UCB, the selection rate ε is dynamic and depends on the risk: the higher the risk is, the less the exploration is performed. In this work, we use the notion of reliability instead of the risk. The more the robot has previously failed, the lower the reliability is. If the reliability is low, more exploitation is required as the current knowledge of cause is not good enough to prevent failure. LEAF uses the Equation 2 for computing ε:

$$\varepsilon = \varepsilon_{max} - (1 - R) * (\varepsilon_{max} - \varepsilon_{min}) \tag{2}$$

Where R is the reliability that represents the success rate of a task t over the past N situations. R is computed through Equation 3 :

$$R = nbrSucc_N / N \tag{3}$$

Where $nbrSucc_N$ is the number of successful situations in the past N situations in H_t. For instance, for $N = 4$, with the H_t presented in Table 1, thus considering S_6, S_5, S_4 and S_3, we obtain $R = 2/4 = 0.5$. By using this process, LEAF is able to efficiently balance exploration and exploitation. **When exploring**, a random context data in the situation, that was not previously identified as a cause, is selected. **When exploiting**, the R-UCB algorithm selects the context data with the highest upper confidence bound d_{cd}. d_{cd} represents the confidence in the selection of cd according to its current CB, its occurrence and the number of feedback already provided by the user. A high d_{cd} means cd needs to be check in priority. It is computed as follows:

$$d_{cd} = CB_{cd} * \sqrt{log(F_t)/N_{cd}} \tag{4}$$

Where CB_{cd} is the current causality belief of context data cd (see Section 5.3.3 for belief computation), F is the number of failure situations for the current task

t, and N_{cd} is the number of feedback provided by the user for causality of context data cd (for task t). For instance, let us consider that the robot is in the situation S_{12}=(katleen isLocatedIn livingroom), (katleen isDoing music) with the history H_t presented in Table 1 and cd=(katleen isLocatedIn livingroom); let us assume the user provided feedback eight times for cd, $N_{cd} = 8$, and that the resulting causality belief is $CB_{cd} = 0.75$, in that case: $d_{cd} = 0.75 * \sqrt{log(4)/8} = 0.21$. The context data cd with the highest d_{cd} are the best candidate to be user checked.

5.2.2 Getting user feedback

Once LEAF has determined what are the context data that require user validation about their causality, it requests feedbacks from the user. We consider five different user's answers ordered by confidence: {'Yes', 'Probably', 'Partially', 'Possibly', 'No' }; each is respectively associated to a belief value: { 1.0, 0.75, 0.5, 0.25, 0.0}. Based on these feedbacks, the asked context data is associated to a **causality belief** CB and stored in a knowledge base. This causality belief is set as follows: Let B be the belief value from the user's answer. If the context data is new and does not have causality belief CB_{cd} yet, its causality belief is set to B. If the context data was already identified as a cause and has a causality belief, the latter is updated using Equation 5:

$$newCB = (oldCB * N + B)/(N + 1) \tag{5}$$

Where $newCB$ is the adapted causality belief of the context data and N is the number of previous user answers for this context data (for task t). Let us suppose that the robot asks the user for validation about the context data cd_a=(katleen isDoing music) after the failure of the situation S_4. The user answers 'Probably', cd_a was not identified as a cause yet, thus $CB_{cd_a} = 0.75$ (cold start process starts after at least 2 failures). Now, let us suppose that the robot has encountered a similar situation some time later. However, this time the user is more confident and answers 'Yes'. Hence, the causality belief of cd_a is updated as $CB_{cd_a} = (0.75 * 1 + 1)/(1 + 1) = 0.875$.

After analyzing user's answers, the robot has identified causes and associated each one with a causality belief. These beliefs are used in the extracted belief, where they can be seen as 'rewards'. In addition, they are used to adapt the planning process in order to prevent further failures.

5.3 Checking current context

Once the causes are identified and associated to a causality belief, they are used in the planning process - HTN in our case. The objective is to take into account these causes in the planning process: if failure causes for a task t are observed, then t should be avoided in the plan. To do so, HTN was enhanced with an altered planning processes. Indeed, whenever HTN is decomposing, it not only checks the preconditions, but also the failing of subtasks. When a method has to be selected

for a compound task ct in the planning phase, the following process is executed. We denote M_{ct} the set of all method realizing ct.

The first step of the process is to merge failure causes C_{t_i} of all subtasks t_i of method m. The merging is quite simple: all the identified causes of subtasks $t_i \in m$ are extracted from O_{t_i} and put in a set C_m that carries all failure causes of the method m. If a context data is a failure cause for multiple tasks in m, the maximum CB is selected. Please note that compound methods are not taken into account: they are tackled when the planning algorithm decomposes them. For example, regarding the task 'alert', the robot has two methods: m_1 =go to user, vocal alert and m_2 =phone alert. The robot has the history presented in Table 1, through the cause extraction process, for task 'vocal alert'. Two causes were identified: cd_a=(katleen isDoing listeningMusic) and cd_b=(katleen vocUnreachTo nono) with $CB_{cd_a} = 0.875$ and $CB_{cd_b} = 1.0$. Thus, in that case C_{m_1} contains cd_a and cd_b.

Once C_m are generated for each possible method m in M_{ct}, methods' conditions are checked. Methods whom conditions are not verified are excluded, the remaining set is labeled M^*_{ct}.The remaining methods' failure causes are then checked. The idea is to check if the method would succeed in the current context. Let W be the current context (world). And let C_{obs} the set of currently observed causes $C_{obs} = W \cap C_m$. Let A be a constant. For each method $m \in M^*_{ct}$, a confidence value is computed through a following logistic differential presented in Equation 6:

$$conf_m = 1 - 1/(1 + e^{-ln(A * \sum_{cd \in C_{obs}} CB_{cd})}) \quad (6)$$

Equation 6 allows to compute a [0,1] confidence value based on any number of causality belief.With a sum of causality belief of 0, the function return 1, meaning the task can be safely executed. The confidence then quickly decrease as the sum of causality belief increases and has an asymptote at 0. Thanks to this function, one cause with a high causality belief is enough to compute a low confidence, thus preventing execution. The more the causality is high, the lower the confidence is, this enables the comparison of tasks' confidences. For example, if the robot is trying to generate a plan in situation $S_1$3=(katleen isLocatedIn livingroom), **(katleen isDoing music)**, for $A = 0.75$ confidence value of m_1 is: $conf_{m_1} = 1 - 1/(1 + e^{-ln(1*0.875)}) = 0.6$. In situation $S_1$4(katleen isLocatedIn livingroom), **(katleen isDoing music)**, **(katleen vocUnreachTo nono)**, where two causes are observed, thus the task is more unlikely to succeed, for $A = 0.75$ confidence value of m_1 is: $conf_{m_1} = 1 - 1/(1 + e^{-ln(1*(0.875+1.0))}) = 0.41$.

Having the confidence values for each computed method in M^*_{ct}, the method with the highest confidence value is selected for decomposition. Within the standard HTN, the first method matching the conditions would have been selected. However, a final checking is done before decomposing. In fact, if the confidence value is below 25%, meaning there is 75% of chance the robot won't succeed, the decomposition is aborted. As all other methods are less reliable, the compound task $ct$is marked to be non-executable. Thus, HTN will backtrack and try to find another option. If a method is successfully selected, HTN can continue its process. In the end, a plan is generated and HTN has checked the possibilities of failures, minimizing the risk of encountering a failing situation.

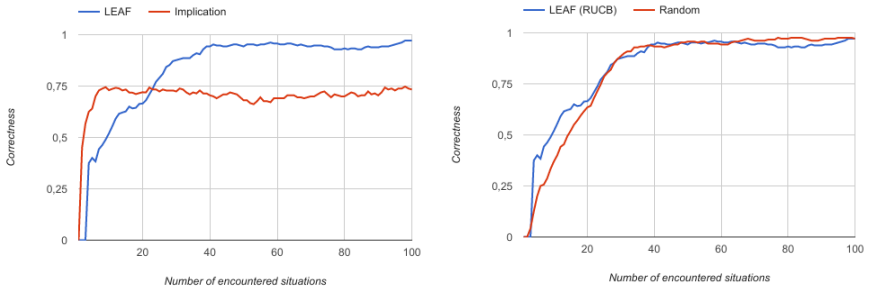
6 Experiments

6.1 Implementation

LEAF has been implemented in Java and relies on Jena[1] to manage ontologies and reasoning. The core code of LEAF is available on github [2]. The implementation is independent from any robots and/or smart environment and holds to the description in Section 5. Nevertheless, we integrated LEAF on a Nao robot[7] using ROS(Robot Operating System) Indigo[3]. Nao is a small robot that is able to walk, identify persons and talk with users. It gathered user validation through vocal interaction thanks to the NaoQi API. The robot was equipped with a DHTN[15] planner. As a proof of concept, we tested our scenario in the Hadaptic[4] platform, that includes a modular room and various sensors. Videos can be found online[5] .

Fig. 3: Picture of the realized tests

6.2 Evaluation



(a) Comparison to literature: LEAF vs implication based approach

(b) Comparison for CD validation: LEAF (RUCB) vs randomly selected cause

Fig. 4: Correctness of task's risk evaluation according to the number of situations encountered

In order to evaluate LEAF learning capabilities, it was applied on a randomly generated dataset. The principle of these evaluations is to simulate various situations and evaluate how LEAF is able to learn the causes and prevent failure situations. The evaluation process is the following. First, a situation is generated and associated with the expected results of tasks' execution. LEAF is

[1] https://jena.apache.org/

[2] https://github.com/Nath-R/LEAF

[3] http://wiki.ros.org/indigo

[4] http://hadaptic.telecom-sudparis.eu

[5] http://nara.wp.tem-tsp.eu/what-is-my-work-about/leaf/

then asked to check if a given task will fail or not in this situation. LEAF's answer is compared to the expected result: if it is correct, the situation is tagged as successful; otherwise, the situation is tagged as failure and LEAF starts the learning process: it determines possible causes to be validated by the user and gets the feedbacks from a virtual user (no human in the simulation process). The process is then repeated on a new situation. The situation generation process consists in feeding the live situation with random, yet controlled, context data such as user's location, object position or user's activity. In our experiments, we considered the Scenario 1 and Scenario 3 possible failure causes. We also created 2 ontological rules, further could have been created, but would have been irrelevant in these experiments. One evaluation run is composed of 100 generated situations. For each step, the correctness is computed as the number of true positive and true negative answers based on the total number of answers. There was 20 runs conducted (for each variant, see below) and the average results are presented in Figure 4. Experiments are divided in two parts: (a) comparing LEAF to the literature, (b) Evaluating RUCB contribution.

LEAF was compared to a *state-of-the-art* like approach in Figure 4a. We used an implication based approach similar to the one proposed by Sariel et al. [16, 9]. It infers implication between data and task's outcomes based on previous encountered situations without the user being in the loop. Results show that the *state-of-the-art* approach learns quicker and reach its asymptote after 10 situations with a correctness of 75%. In the meantime, LEAF achieves only 50% of correctness for the same amount of learning data. Although LEAF is slower to learn, it reaches a better correctness after 22 situations and reaches its asymptote after 40 situations with a correctness of 95%. Such a difference can be explained by the validation phase. In fact, it needs to have enough user feedbacks in order to reach efficient results. In these evaluations, three validations were asked per failing situation. This explains the slow increase compared to the implication based approaches. Nevertheless, by asking the user, LEAF reaches extremely precise results: when trained, LEAF outclassed the state-of-the-art approach by almost 20%. In fact, the implication based learning process suffers of all the limits described in section 3. These experiments underline that LEAF is efficient once trained, but needs to be improved in order to learn quicker. Using a higher number of user validations when the robot is not experienced is a possible way to solve this shortcoming.

Determining what to ask to the user is essential for the efficiency of LEAF. Particularly, LEAF selects between exploration and exploitation, in other word, exploring new causes or improving already identified ones. RUCB (used by LEAF) was compared to a random data selection. Results presented in Figure 4b show that RUCB allows for a quicker learning in comparison a randomly selected data: it is 10% more correct between 4 and 16 situations. After 20 situations, the two approaches obtain similar correctness and, as expected, have the same asymptote around 95% (when experienced, both approaches are similar). RUCB efficiently selects the causes to validate, thus it learns quicker than the random based approach as it does not 'waste' questions to user. It proves RUCB to be pertinent to reach decent correctness faster.

7 Conclusion

In this paper, we presented a novel approach to prevent encountering failure situations and thus enhancing the efficiency of domestic robots. Our solution is based on a live learning process that analyses the failures. We described how failing situations were modeled and how failing causes were extracted using user validation. We enhanced HTN by enabling it to check failing risk when decomposing, ensuring subtasks success when executed. Furthermore, we implemented and tested our approach with a Nao robot and the Hadaptic smart platform. Results underline the validity of our contribution, however further experimentations in real case scenarios using the Evident[6] platform, a fully equipped smart apartment, are to ought be performed.

References

1. Al-Moadhen, A., Qiu, R., Packianather, M., Ji, Z., Setchi, R.: Integrating robot task planner with common-sense knowledge base to improve the efficiency of planning. Procedia Computer Science **22**, 211–220 (2013)
2. Bouneffouf, D.: Drars, a dynamic risk-aware recommender system. Ph.D. thesis, Institut National des Télécommunications (2013)
3. Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence **2**(3), 189–208 (1972)
4. Garivier, A., Moulines, E.: On upper-confidence bound policies for switching bandit problems. In: International Conference on Algorithmic Learning Theory, pp. 174–188. Springer (2011)
5. Georgievski, I., Aiello, M.: An overview of hierarchical task network planning. arXiv:1403.7426 (2014)
6. Ghezala, M.W.B.: Compréhension dynamique du contexte pour l'aide à l'opérateur en robotique. Ph.D. thesis, Institut National des Télécommunications (2015)
7. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: Mechatronic design of nao humanoid. In: Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pp. 769–774. IEEE (2009)
8. Hanheide, M., Göbelbecker, M., Horn, G.S., Pronobis, A., Sjöö, K., Aydemir, A., Jensfelt, P., Gretton, C., Dearden, R., Janicek, M., et al.: Robot task planning and explanation in open and uncertain worlds. Artificial Intelligence (2015)
9. Kapotoglu, M., Koc, C., Sariel, S.: Robots avoid potential failures through experience-based probabilistic planning. In: Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on, vol. 2, pp. 111–120. IEEE (2015)
10. Lallement, R., De Silva, L., Alami, R.: Hatp: An htn planner for robotics. arXiv preprint arXiv:1405.5345 (2014)
11. Lassila, O., Swick, R.R., et al.: Resource description framework (rdf) model and syntax specification (1998)
12. Mahajan, A., Teneketzis, D.: Multi-armed bandit problems. Foundations and Applications of Sensor Management pp. 121–151 (2008)
13. Milliez, G., Lallement, R., Fiore, M., Alami, R.: Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring. In: The Eleventh ACM/IEEE International Conference on Human Robot Interaction, pp. 43–50. IEEE Press (2016)
14. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An htn planning system. J. Artif. Intell. Res.(JAIR) **20**, 379–404 (2003)
15. Ramoly, N., Bouzeghoub, A., Finance, B.: Context-aware planning by refinement for personal robots in smart homes. In: ISR 2016: 47st International Symposium on Robotics; Proceedings of, pp. 1–8. VDE (2016)
16. Sariel, S., Yildiz, P., Karapinar, S., Altan, D., Kapotoglu, M.: Robust task execution through experience-based guidance for cognitive robots. In: Advanced Robotics (ICAR), 2015 International Conference on, pp. 663–668. IEEE (2015)
17. Weser, M., Off, D., Zhang, J.: Htn robot planning in partially observable dynamic environments. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, pp. 1505–1510. IEEE (2010)

[6] https://evident.telecom-sudparis.eu