# AI PLANNING AND REASONING FOR A SOCIAL ASSISTIVE ROBOT

Bilal Hoteit[1,] Ali Abdallah[2], Ahmad Faour[2], Imad Alex Awada[1], Alexandru Sorici[1] and Adina Magda Florea[1]
*[1]Politehnica University of Bucharest, Splaiul Independentei 313, Bucharest, 060042, Romania*
*[2]Lebanese University, Badaro, Museum, Beirut, 6573/14, Lebanon*

**ABSTRACT**

Social robot in service is radically changing the ways of performing tasks and it becomes a distinct and valuable nascent. To achieve persist autonomy, robotic systems implement a closed-loop consisting of at least planning, reasoning and acting phases. From the continual loop perspective, this paper presents the ROSPlan framework, as a task planning and executing platform, and it introduces the integration of the ROSPlan framework in a service robotic system. Moreover, it discusses the planning, observing, monitoring, and executing processes in detail.
A use case scenario is proposed which is suitable for the experiment requirements, and it presents a service robot accomplishing an assistant task in educational campus. Considering the implementation of the ROSPlan framework, TurtleBot3 robot using Gazebo simulates the execution phase of the proposed scenario.

**KEYWORDS**

Service Robot, Robotic Control System, Automated Planning, Task Planning, Artificial Intelligence

## 1. INTRODUCTION

A service robot is earning more interest in the field of scientific research, because it enables the robot to achieve complex tasks that could assist staff in carrying out their jobs. Service robot could be asked to achieve several tasks where every task is attached with a predefined plan consisting of several basic behaviors, as demonstrated in (Gavril, 2019). In practice, a plan for any mission could not be easily scripted due to the variance of the environmental conditions, the initial position of the robot, the goals, and other dynamic factors. For instance, to navigate to various landmarks, robot should deal with a proper ordering of landmarks based on his initial location every time. On the one hand, to accomplish a mission, robot should handle several heterogeneous actions or skills. On the other hand, a plan construction becomes tedious and, in some cases, arbitrarily complex.

Planning approach increases the self-management property of the control system (Belta, 2007). Planning requires a domain model representing the states and actions which supports the planner to generate a causally valid structure that composed of organized actions. AI planning (Long, 2005) could be used to manage the robotic control system, and it could rebuild the structure to optimize cost or to fix execution failure. Several planning techniques have been applied to different types of robots and in several domains.

AI planning and robotic control approaches have different mechanisms and representations that consider as a big challenge for integrating planning and robotics systems. In particular, service robot must adapt to their dynamic environment. Robotic control systems should handle uncertainty and consider exogenous events to successfully achieve their goals. Thus, classical planning is not adequate to the robotic systems. Task planning (Hanheide, 2017). which is related to the robotics domain, is in growing interest. Task planning has been proposed to support a robot, resulting in a more intelligent behavior. Several tools and frameworks have been proposed to facilitate the integration of robotics systems and planning approaches, and increased reliance on AI systems in various fields. Most of the proposed frameworks faced several problems, and even they have implemented a non-adopted mechanism or non-standard language.

Robot operating system (ROS) (Quigley, 2009) integrates the architecture components seamlessly and serves as a middleware between the higher and lower-level architecture layers. ROS embodies several packages that provide robot-specific skills to support the robot accomplishing several tasks such as navigation, manipulation, etc. External ROS-based planning systems can be integrated with ROS through specific wrapper interfaces.

ROSPlan framework (Cashmore, 2015) has been proposed as a modular and common framework, and it adopts two standards: ROS platform and PDDL language (Aeronautiques, 1998). Presently, ROSPlan has been limited to temporal planning and deterministic approach. ROSPlan is not appropriate for neither Probabilistic nor Conditional planning without a proper extension, as described in section 2.

Historically, classical planning is classified as an intractable problem. AI Planning might not guarantee to generate plans in a reasonable time. Moreover, the probabilistic or conditional planning requires a computational demand and might be not scale to handle huge and complex tasks as demonstrated in Kaelbling et al. (1998) and Pryor et al. (1996) respectively. Regardless of the required knowledge about either the probability distribution over the domain space or the history of the observation, modeling a dynamic space is a serious challenge.

For the above, we adopt to use the ROSPlan framework as a generic symbolic planner or a task planner for our service mobile robot. Taken into consideration: 1) the present ROSPlan drawback; 2) the planning issues; and 3) our proposed scenario. We push towards a deterministic approach to somehow decrease the planning generation time and to facilitate domain modeling. Therefore, we adopt a re-planning strategy during execution failure considering the unexpected state and the original goals. The main contribution of this paper is: 1) to present the ROSPLAN framework; 2) to generate a reasonable plan to achieve somehow a complex mission; 3) to adapt for a dynamic environment in a more intelligent way than using pre-compiled plan; 4) to update or advise the plan due to either the inaccuracy of the sensors or during execution failure.

## 2. RELATED WORK

Historically, a domain-specific planner is integrated into several types of systems. On the contrary, several architectures have been proposed and integrated a domain independent planning system with their robotic control systems. Planning depends on a representational model that provides the available actions, the prediction of the action's effects, and where the actions are applicable. The Planning Domain Description Language (PDDL) is a standard modeling language and is used to represent the planning domain. PDDL presented in 1998 (Aeronautiques, 1998) and extended through several variants (Pinto, 2012).

Planning component (Gavril, 2019) has been proposed to control a pepper-type robot by implementing a hand-coding plan composing of several basic behaviors. In (Pinto2012), the authors extended T-Rex (Teleo-Reactive Executive) to a goal-oriented system architecture for autonomous underwater vehicles (AUVs) to enhance ocean science. Automated planning is embedded in T-Rex to provide on-board planning and execution processes. The planning module has a specific look-ahead which decides how far ahead it can plan, and a bounding time which decides the available planning period. In (Jim, 2013), the authors proposed PELEA (Planning and Execution LEarning Architecture) as a flexible modular architecture. The PELEA incorporates several processes like sensing, planning, executing, monitoring, re-planning and even learning from past experiences. Inspired by the Human Autonomic Nervous System, the authors in (Jimoh, 2013) proposed an architecture that incorporates a symbolic AI planning to enhance the self-management property. The architecture extends the Traditional control loop of UTC (Urban Traffic Control) by introducing deliberation that allows the system to reason and generate effective plans, in addition, to reduce traffic congestion. In (Dornhege, 2013), the authors integrate a Temporal Fast Downward/Modules (TFD/M) in the Tidyup-Robot system to facilitate the required tasks composing of both the navigation and manipulation processes. Moreover, the system extended the standard PDDL to PDDL/M which supports semantic attachments to provide the applicability testing for abstract actions.

In fact, there is a need for a standard planning and execution framework to be used in developing robotic architectures. The authors in (Cashmore, 2015) proposed the ROSPlan framework that provides a generic method for task planning in the (AUV). They highlight on how to integrate ROSPlan with the ROS ecosystem of AUVs. In (Cashmore, 2016), the authors introduced an opportunity planning to achieve additional utilities during the execution of AUV missions. The proposed system inserts opportunistic plan fragments into an existing robust plan. The execution of the original plan is suspended and pushed to a stack, whereas the tail of the original plan is re-executed only after the opportunistic plan execution is finished. In contrast of HTN planning (Erol, 1994), using a bottom-up approach, authors in (Cashmore, 2016) decomposed the problem into a strategic and a tactical layer. whilst, similar to an HTN model, the strategic plan composes of macro actions that encapsulate the tactical plans which are constructed from the original actions of the domain. The system plans for each task at the tactical layer independently, then it finds a valid execution order of the tactical plans. The aforementioned systems adopt the temporal planning technique, use POPF (Coles, 2010) as a planner, and relie on an extension version of PDDL to describe their domain.

On the contrary, conditional planning (Sanelli, 2017) and probabilistic planning (Canal, 2019) techniques have been used with an extended version of the ROSPlan framework. In the former, Sanelli et al. proposed a framework to generate a robust conditional plan. Based on a multi-layered architecture, the framework composed of a Planning System, a Plan Translator and a Plan Executor. The extended ROSPlan framework relies on the Contingent-FF (Hoffmann, 2015) and formulates the short term HRI as a conditional planning problem. Then, the conditional plan is translated into a Petri-Net Plan which is executed by the PNP executor module. In the latter, authors in (Canal, 2019) extended the ROSPlan framework to handle probabilistic planning domain and problems which are written in Relational Dynamic Influence Diagram Language (RDDL) (Sanner, 2010). They implement a custom interfaces to link the RDDL planner, PROST planner (Keller, 2012), that uses the RDDLSim server with the KB that represents an RDDL ontology. Moreover, they implement an online plan dispatcher that provides RDDL Client/Server protocol to interleave the plan execution and the deliberation process.

## 3. ROSPLAN FRAMEWORK

Since ROSPlan framework provides an innovative action and sensing interfaces that are compatible with ROS, it could be integrated easily with any ROS-enabled robotic system to automate the planning process and coordinate the activities of lower level controllers. ROSPlan is capable to reason with any version of PDDL and can easily wrapped several planners such as Temporal Fast Downward (Eyerich, 2012), UPMurphi (Della, 2012). With an extended approach, a non-standard models and planners can be used. ROSPlan consists of several components and interfaces where each component is an atomic module that performs a single function. ROSPlan is a scalable platform where components can be integrated in different manners to construct different set of architectures, whereas any component could be replaced by a more advanced one.

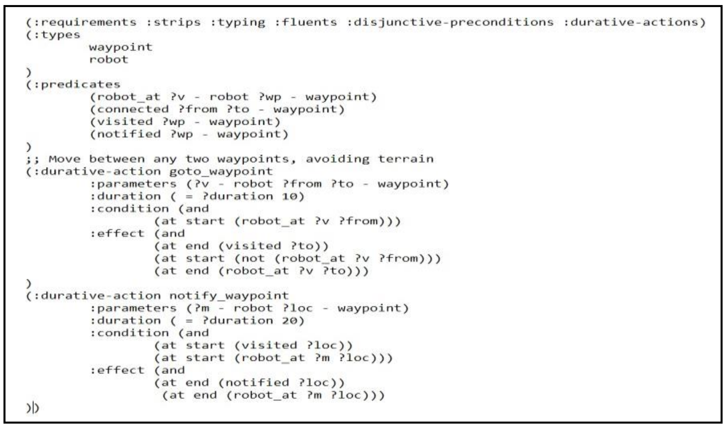
Knowledge Base node launches with two parameters, a domain and problem file. The node implements many services to fetch and update the domain and state variables. Problem generator node produces a new problem instance, and it relies on KB services to get the domain and state details. The problem instance is published as a message to a ROS topic, or it can be written to a file. The planner node is a wrapper for the AI planner, and it publishes the generated plan to a ROS topic or writes the solution to a file. The parsing node translates the plan into a ROS representation that is ready for execution through transforming each PDDL action into a ROS action message. These ROS messages are collected into a structure specifying the execution details. The dispatching node executes the parsing plan through publishing the action messages at the right time. This node is restricted with the same representation of the parsing node, whereas a custom algorithm is implemented to provide a simple plan dispatcher or an esternal plan dispatcher. Also, the dispatching node provides a service informing whether the plan is executed successfully. Action interface node simplifies the linking of PDDL action and ROS action message.

## 4. OUR PROPOSED ARCHITECTURE

Relying on several agents, service robot could perform several functions as vision recognition, speech recognition, and navigation. The overall performance of the robotic system becomes highly contingent on the proper architecture design due to the expanding of the application domains and the increasing number of components. Pepper (Gavril, 2019) is a social assistive robot interacting with and supporting administrative staff in their daily tasks. Pepper is focused on social interactions such as: speaking, guiding, reminding, observing. The robotic system implements a planning component that manages and controls several components (navigation, vision, speech), whereas each component is supported the robot to achieve a specific task. The planning component provides somehow a hard-coding plan that consists of several basic skills or behaviours.

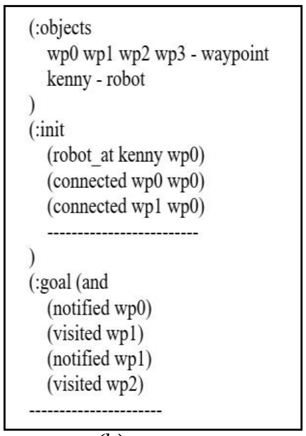
This paper aims to replace the planning module with a symbolic task planner specifically the ROSPlan framework. It investigates the integration of the ROSPlan framework with ROS-enabled robotic system. The proposed system implements a closed-loop between the ROSPlan framework and ROS ecosystem. The loop drives the planning, observing, monitoring, and executing processes to enable the robot accomplishing a complex mission and adapting with the dynamic environment. We describe the implementation phase, and we present the domain model, the problem instance, the generated plan, in addition to the plan execution. Also, we implement a specific strategy for the re-planning phase.

Our proposed scenario entails a robot having the speech and navigation skills. Figure 1(a) illustrates the domain model for the proposed scenario, where the domain has been represented in aextended PDDL format.



```
(:requirements :strips :typing :fluents :disjunctive-preconditions :durative-actions)
(:types
        waypoint
        robot
)
(:predicates
        (robot_at ?v - robot ?wp - waypoint)
        (connected ?from ?to - waypoint)
        (visited ?wp - waypoint)
        (notified ?wp - waypoint)
)
;; Move between any two waypoints, avoiding terrain
(:durative-action goto_waypoint
        :parameters (?v - robot ?from ?to - waypoint)
        :duration ( = ?duration 10)
        :condition (and
                (at start (robot_at ?v ?from)))
        :effect (and
                (at end (visited ?to))
                (at start (not (robot_at ?v ?from)))
                (at end (robot_at ?v ?to)))
)
(:durative-action notify_waypoint
        :parameters (?m - robot ?loc - waypoint)
        :duration ( = ?duration 20)
        :condition (and
                (at start (visited ?loc))
                (at start (robot_at ?m ?loc)))
        :effect (and
                (at end (notified ?loc))
                 (at end (robot_at ?m ?loc)))
))
```

*(a)*



```
(:objects
   wp0 wp1 wp2 wp3 - waypoint
   kenny - robot
)
(:init
   (robot_at kenny wp0)
   (connected wp0 wp0)
   (connected wp1 wp0)
   -------------------------
)
(:goal (and
   (notified wp0)
   (visited wp1)
   (notified wp1)
   (visited wp2)
----------------------
```

*(b)*

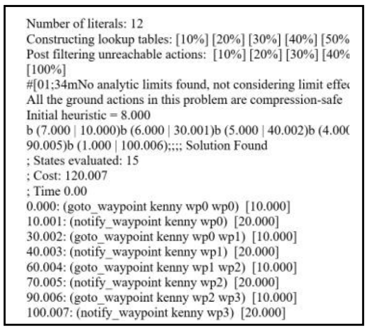
Figure 1. (a) the Domain File. (b) the Problem File for the Reminder Task

A problem file is used to insert the objects and their attributes, the facts such as the real coordinates of the landmarks, the available paths, and the goals into KB. A part of the planning problem for our scenario is shown in Figure 1 (b). The problem file is re-generated during either a first attempt of planning or during a re-planning phase, considering information that are available in KB. This process re-formulates the initial state and the remaining goals.

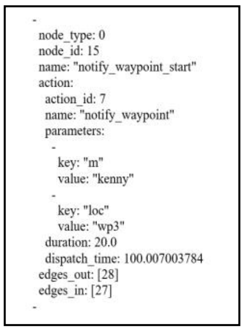
The planner POPF, a temporal and metrical planner, solves the planning problem, and generates the temporal plan. Figure 2 (a) illustrates the generated plan. The sequence of the generated actions is converted into ROS action messages that are closely tied and compatible with our ROS ecosystem as shown in Figure 2 (b). In addition, Figure 2 (c) illustrates a part of the Esternal parsed plan that considers the action timeout and is published through the parsing node. Sequentially, the dispatching node, denoted as the executor, executes each ROS action message in the parsed plan by relying on their relevant node which encapsulate the lower-level robot controllers.

## 4.1 Execution Phase

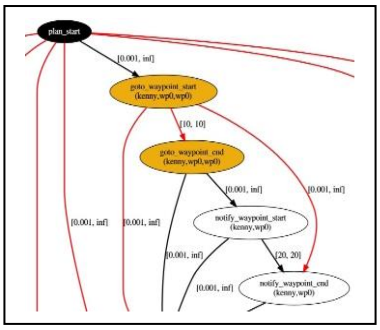
An action interface node implements an abstract execution method to connect ROS action message to an existing library through ROS service or ROS action-lib mechanisms. The action interface node, denoted as a controller, subscribes to an action dispatch topic that is published by the executor, whereas the controller publishes a feedback messages on an action feedback topic to inform the executor about the execution status.



```
Number of literals: 12
Constructing lookup tables: [10%] [20%] [30%] [40%] [50%
Post filtering unreachable actions:  [10%] [20%] [30%] [40%
[100%]
#[01;34mNo analytic limits found, not considering limit effe
All the ground actions in this problem are compression-safe
Initial heuristic = 8.000
b (7.000 | 10.000)b (6.000 | 30.001)b (5.000 | 40.002)b (4.00
90.005)b (1.000 | 100.006);;;; Solution Found
; States evaluated: 15
; Cost: 120.007
; Time 0.00
0.000: (goto_waypoint kenny wp0 wp0)  [10.000]
10.001: (notify_waypoint kenny wp0)  [20.000]
30.002: (goto_waypoint kenny wp0 wp1)  [10.000]
40.003: (notify_waypoint kenny wp1)  [20.000]
60.004: (goto_waypoint kenny wp1 wp2)  [10.000]
70.005: (notify_waypoint kenny wp2)  [20.000]
90.006: (goto_waypoint kenny wp2 wp3)  [10.000]
100.007: (notify_waypoint kenny wp3)  [20.000]
```



```
-
  node_type: 0
  node_id: 15
  name: "notify_waypoint_start"
  action:
    action_id: 7
    name: "notify_waypoint"
    parameters:
      -
        key: "m"
        value: "kenny"
      -
        key: "loc"
        value: "wp3"
    duration: 20.0
    dispatch_time: 100.007003784
  edges_out: [28]
  edges_in: [27]
-
```


plan_start
[0.001, inf]
goto_waypoint_start (kenny,wp0,wp0)
[10, 10]
goto_waypoint_end (kenny,wp0,wp0)
[0.001, inf]
[0.001, inf]
[0.001, inf]
[0.001, inf]
notify_waypoint_start (kenny,wp0)
[0.001, inf]
[20, 20]
[0.001, inf]
notify_waypoint_end (kenny,wp0)

*(a)* *(b)* *(c)*

Figure 2. (a) A Fragment of the Plan. (b) A ROS notify_action Message. (c) Esternal Parsed Plan

Then, the executor can make a decision about the next step of the plan execution. For instance, the controller that matches the *goto_waypoint* action, communicates with the *move_base* node using a SimpleActionClient mechanism. So, the controller implements a *move_base* action client to connect with a *move_base* action server. First, the controller fetches the real values of the target location from the KB to create a valid *move_base* goal. Then, *move_base* action client sends the goal to *move_base* action server to navigate to the goal, and waits for the result. Based on the response of the *move-base* action server, namely the action client *goalstate*, and optionally on a monitoring phase that is described in more detail in the next section, the controller publishes an action feedback message that could be either positive (*action achieved*) or negative (*action failed*) feedback message.

Furthermore, the controller publishes action enabled as an initial feedback informing the executor that the action has been picked up. The controller, based on a *durative-time* parameter, could cancel the execution and publishes a time out feedback message to inform the executor that the action has exceeded the time and it is aborted. The duration time parameter of the controller is different from the timeout action of the planner. The executor, based on the schedules time of the planned action, could cancel the action without waiting for the feedback of the controller. In this case, the executor will send a cancel request to the controller that is continuously monitored an action cancellation message to stop the execution process at any time.

## 4.2 Monitoring Phase

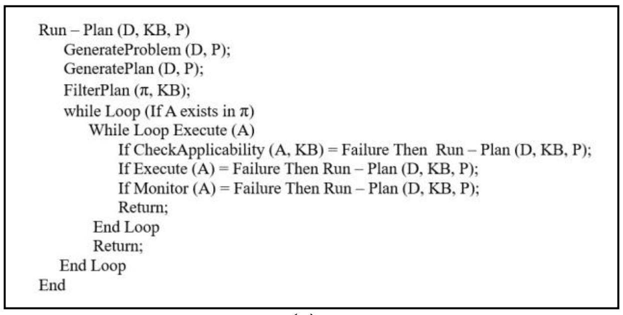
KB supports the controller to implement a monitoring process. The controller checks the PDDL parameters to reveal if the action is applicable. For instance, the controller, that matches the *goto_waypoint* action, checks the preconditions of the action based on the *robot_at* predicates of the KB. When the action is not applicable, the controller publishes a negative feedback message to inform the executor that the action could not be executed.

In general, ROS-enabled robotic system consists of several libraries that provide ROS action messages previously, while also provide interpreted sensors data that are somehow a prerequisite for various robot functions. These interpreted sensors data that is provided through the observation process, supports another type of monitoring. This monitoring process checks the final status of the action execution and reveals if the real environment matches the expected one. This monitoring is implemented for the *notify_waypoint* action.

We implement a custom action interface matching the *notify_waypoint* action with their associated ROS action messages. This controller is supported by either the monitoring phase to check its real effects or the knowledge base to check its applicability. The controller is simply inherited from the *goto_waypoint* action interface. On-contrary, the controller is linked to a ROS library that provides the TTS operation through a ROS service mechanism. In this case, the controller could not solely depend on the response of the TTS service to validate the execution process. So, another monitoring process is implemented to check if the execution is achieved successfully. A node holding a speech recognition capability should be run. This node converts the speech into text and publishes the text on a specific topic. Thus, the controller is subscribed to this topic to reveal the execution's status. For instance, to enhance the NLP capabilities, the proposed system relies on *baidu_asr* and *sound_play* ROS packages. The former provides the *recognizer* node that converts the voice to a text and publishes a string message on the output topic. While, the latter provides a *sound_play* node that translates the text on the *robotsound* topic into sounds. The *notify_waypoint* controller publishes a text to the *robotsound* topic and scans the output topic to check the execution.

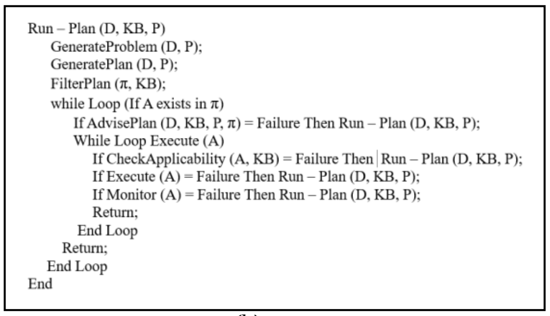
## 4.3 Re-Planning Phase

In general, when the action is executed successfully, the controller updates the knowledge base with the action effects and publishes the action achieved feedback message. In this case, the executor is continuing the dispatching process. On the other hand, the executor could trigger a re-planning phase due to several issues, such as: the dispatching time is over, the action is not applicable, the action duration is exceeded, or even the action is failed. In the re-planning case, the system re-formulates the mission as a new problem considering the new information such as the unexpected state, the remaining goals, as well as the predicates and facts in KB. The problem is constructed with new initial state and the new plan is generated for the remaining goals.



```
Run – Plan (D, KB, P)
    GenerateProblem (D, P);
    GeneratePlan (D, P);
    FilterPlan (π, KB);
    while Loop (If A exists in π)
        While Loop Execute (A)
            If CheckApplicability (A, KB) = Failure Then  Run – Plan (D, KB, P);
            If Execute (A) = Failure Then Run – Plan (D, KB, P);
            If Monitor (A) = Failure Then Run – Plan (D, KB, P);
            Return;
        End Loop
        Return;
    End Loop
End
```

*(a)*



```
Run – Plan (D, KB, P)
    GenerateProblem (D, P);
    GeneratePlan (D, P);
    FilterPlan (π, KB);
    while Loop (If A exists in π)
        If AdvisePlan (D, KB, P, π) = Failure Then Run – Plan (D, KB, P);
        While Loop Execute (A)
            If CheckApplicability (A, KB) = Failure Then Run – Plan (D, KB, P);
            If Execute (A) = Failure Then Run – Plan (D, KB, P);
            If Monitor (A) = Failure Then Run – Plan (D, KB, P);
            Return;
        End Loop
      Return;
    End Loop
End
```

*(b)*

Figure 3. (a) re-plan only when necessary. (b) re-plan could be done before any action execution

We implement two re-planning strategies, the first approach only considers the execution failure, whereas the second approach responds to a plan validation mechanism, as illustrated in Figure 3 (a) and (b) respectively. Both approaches can break the continual closed-loop and trigger a re-planning phase. In the case of algorithm 1, the re-planning is triggered only when necessary, during any execution failure. On contrary, algorithm 2 implements an advice and validate process, where the plan is checked before any action execution. With such implementation, the executor can trigger the re-planning process if either the plan is invalid or the plan may cause failure in future.

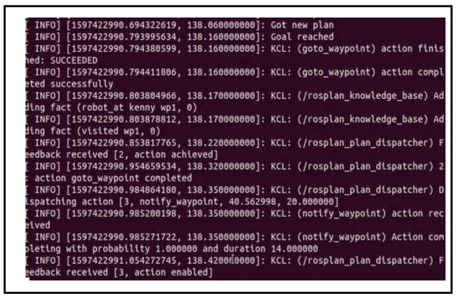
Based on our proposed scenario, the environment is somehow predictable since it is a partial specified environment, in addition, this paper adopts an active sensing approach that is only updating the KB after a successful action execution. However, there is no continuous updating for the KB that could be occurred based on a passive sensing approach. Thus, the plan validation only considers the intersection of the facts and predicates of the KB with the preconditions of actions in the plan.

## 5. IMPLEMENTATION AND RESULTS

We evaluate our approach with a case scenario that presents a complex mission for a service robot interacting with humans. The robot navigates to five exam rooms and alert students about the remaining time for the exam. We run the proposed mission in the Gazebo simulation using TurtleBot3 robot. Gazebo works smoothly along with ROS, and can import Maps easily. Based on the map, we choose 5 real values as a valid location whereas the navigation program could run efficiently on the TurtleBot3 in the Gazebo simulation. We installed two out of box ROS packages, *sound_play* and *baidu_asr* to provide the NLP capabilities.

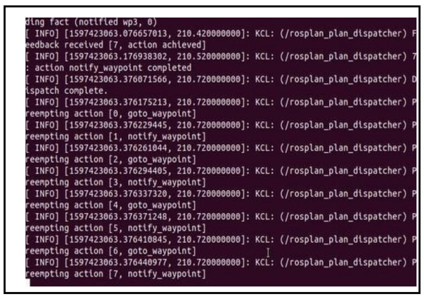
The system initializes the KB by inserting the initial states and several facts from the domain and problem file, such as the locations and the connected way-points. We assumed a 'most likely' duration for each action based on our scenario, such as 10 seconds for the navigation action and 20 seconds for the notification action. The same duration is used for the controller. The planner, POPF, is used to generate a temporal plan. Then, the cycle is continue as presented in the previous section. Two custom action interfaces have been implemented to support the robot to move and say. The *goto_waypoint* action controller relies on the *move-base* node to navigate the points, while the *notify_waypoint* action controller relies on the *sound_play* node to play sounds on the sound card. Denotes as to grounding process, the controllers fetch the state variables and predicates from KB, which are required for both PDDL actions. For instance, way-point *wp1* is changed to its real value [0.0, -0.5, 0.0]. Then, the controllers check the preconditions of their associated actions such as, for the *notify_action*, the *visited (wp1)* predicate in the KB should be true. Moreover, the controller optionally checks the current position of the robot through subscribing the *amcl_pose* topic.

The *notify_waypoint* action controller invokes the *say* service that is provided through the *sound_play* node, to accomplish the *notify_waypoint* action. Note that, the input argument is just a small notification message. To validate the execution, the controller scans the *output* topic propagating a message from the *baidu_asr* node. On the other hand, the *goto_waypoint* action controller handles the execution of the *move_base* action service by sending a goal to the *move_base* action server and waiting for the result message. When the execution is succeeded, the controllers update the knowledge base with the predicates such as, *notify(wp1)* and *visited(wp1)* are updated to true. Then, the controllers inform the executor that the execution is accomplished. The executor will handle another action. Figure. 4 illustrated the whole execution.



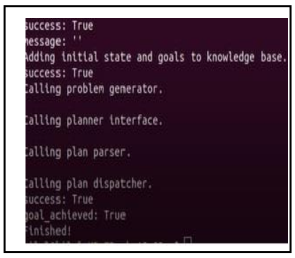
```
[ INFO] [1597422990.694322619, 138.060000000]: Got new plan
[ INFO] [1597422990.793995634, 138.160000000]: Goal reached
[ INFO] [1597422990.794380599, 138.160000000]: KCL: (goto_waypoint) action finis
hed: SUCCEEDED
[ INFO] [1597422990.794411806, 138.160000000]: KCL: (goto_waypoint) action compl
eted successfully
[ INFO] [1597422990.803804966, 138.170000000]: KCL: (/rosplan_knowledge_base) Ad
ding fact (robot_at kenny wp1, 0)
[ INFO] [1597422990.803878812, 138.170000000]: KCL: (/rosplan_knowledge_base) Ad
ding fact (visited wp1, 0)
[ INFO] [1597422990.853817765, 138.220000000]: KCL: (/rosplan_plan_dispatcher) F
eedback received [2, action achieved]
[ INFO] [1597422990.954659534, 138.320000000]: KCL: (/rosplan_plan_dispatcher) 2
: action goto_waypoint completed
[ INFO] [1597422990.984864180, 138.350000000]: KCL: (/rosplan_plan_dispatcher) D
ispatching action [3, notify_waypoint, 40.562998, 20.000000]
[ INFO] [1597422990.985200198, 138.350000000]: KCL: (notify_waypoint) action rec
eived
[ INFO] [1597422990.985271722, 138.350000000]: KCL: (notify_waypoint) Action com
pleting with probability 1.000000 and duration 14.000000
[ INFO] [1597422991.054272745, 138.420000000]: KCL: (/rosplan_plan_dispatcher) F
eedback received [3, action enabled]
```

*(a)*



```
ding fact (notified wp3, 0)
[ INFO] [1597423063.076657013, 210.420000000]: KCL: (/rosplan_plan_dispatcher) F
eedback received [7, action achieved]
[ INFO] [1597423063.176938302, 210.520000000]: KCL: (/rosplan_plan_dispatcher) 7
: action notify_waypoint completed
[ INFO] [1597423063.376071566, 210.720000000]: KCL: (/rosplan_plan_dispatcher) D
ispatch complete.
[ INFO] [1597423063.376175213, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [0, goto_waypoint]
[ INFO] [1597423063.376229445, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [1, notify_waypoint]
[ INFO] [1597423063.376261044, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [2, goto_waypoint]
[ INFO] [1597423063.376294405, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [3, notify_waypoint]
[ INFO] [1597423063.376337320, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [4, goto_waypoint]
[ INFO] [1597423063.376371248, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [5, notify_waypoint]
[ INFO] [1597423063.376410845, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [6, goto_waypoint]
[ INFO] [1597423063.376440977, 210.720000000]: KCL: (/rosplan_plan_dispatcher) P
reempting action [7, notify_waypoint]
```

*(b)*



```
success: True
message: ''
Adding initial state and goals to knowledge base.
success: True
Calling problem generator.

Calling planner interface.

Calling plan parser.

Calling plan dispatcher.
success: True
goal_achieved: True
Finished!
```

*(c)*

Figure 4. (a) Presents the Controllers Execution. (b) Presents the Actions Execution. (c) Presents Final Result

Otherwise, during any failure, the system will trigger a re-planning phase. The proposed scenario is implemented considering algorithm 1. Note that, based on our proposed scenario, the original plan is always valid. The implementation of algorithm 2 is out of scope from this paper.

## 6. CONCLUSION AND FUTURE WORK

This study is implemented in a simulated environment, and it used the ROSPlan framework to integrate in one unit the robotic (lower-level) and planning (higher-level) approaches in order to control the behaviour of the robot. Since the approach is basic on the ROS platform, it can be a general one and can be deployed for several tasks and in several domains. This paper highlights on how to implement the interleaving between acting and planning, and it presents the challenging of the integration process to increase the persist autonomy. An innovative approach is introduced to implement the closed loop that manages different operations in details. In particular, it highlights on the planning, sensing, monitoring, and execution operations.

In the future work, this framework will be implemented in a real world using the pepper robot. A scripted system is needed to match the specified mission written in a high-level or human-like language into a symbolic one. For the navigation task, we need to construct the goals based on the domain, locations, and map. Moreover, the system can implement a custom sensing interface that subscribes to a sensor node and continuously updates KB with the current data. We can adopt the visual sensing as a passive technique where KB is updated continuously through the sensors and speakers by implementing a new or separate node. Consequently, considering another complex scenario, we will implement the second approach of the re-planning mechanism, as demonstrated in algorithm 2.

## REFERENCES

Aeronautiques, C., 1998. PDDL| The Planning Domain Definition Language.

Belta et al., 2007. Symbolic planning and control of robot motion: Finding the missing pieces of current methods and ideas.

Canal et al, 2019. Probabilistic planning for robotics with ROSPlan. Proceedings of Annual Conference Towards Autonomous Robotic, pp. 236- 250

Cashmore et al, 2015. Rosplan: Planning in the robot operating system. Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, Jerusalen, Israel, pp. 333-341.

Cashmore et al, 2016. Strategic planning for autonomous systems over long horizons. Proceedings of ICAPS.

Cashmore et al, 2016. Opportunistic planning for increased plan utility. Proceedings of ICAPS.

Coles et al., 2010. Forward-chaining partial-order planning.

Della p. et al, 2012. A universal planning system for hybrid domains. In Applied intelligence, Vol. 32, No. 4, pp 932-959

Dornhege, C. and Hertle, A., 2013. Integrated Symbolic Planning in the Tidyup-Robot Project. Proceedings of AAAI Spring Symposium: Designing Intelligent Robots.

Erol, K. et al, 1994. HTN planning: Complexity and expressivity. Proceedings of AAAI, pp. 1123-1128.

Eyerich p. et al, 2012. Using the context-enhanced additive heuristic for temporal and numeric planning. Springer.

Fox et al., 2011. Modelling Mixed Discrete-Continuous Domains for Planning. In arXiv, p 1110.

Gavril, A. et al, 2019. Towards a Modular Framework for Human-Robot Interaction and Collaboration. Proceedings of 22nd International Conference on Control Systems and Computer Science (CSCS), pp. 667- 674.

Hanheide et al. 2017. Robot task planning and explanation in open and uncertain worlds. In Artificial Intelligence, Vol. 247, pp 119-150.

Hoffmann, J. and Brafman, R., 2005. Contingent planning via heuristic forward search with implicit belief states. Proceedings of ICAPS.

Jim et al, 2013. Integrating planning, execution, and learning to improve plan execution. In Computational Intelligence, Vol. 29, No. 1, pp 1-36.

Jimoh, F. et al, 2013. Autonomic system architecture: An automated planning perspective. Proceedings of IFIP International Conference on Artificial Intelligence Applications and Innovations.

Kaelbling et al., 1998. Planning and acting in partially observable stochastic domains. In Artificial intelligence, Vol. 101, No. 1-2, pp 99-134.

Keller, T. and Eyerich, P., 2012. PROST: Probabilistic Planning Based on UCT. Proceedings of ICAPS, pp. 119-127

Long, D., 2005. Automated planning: Theory and practice. In Assembly Automation

Pinto, J. et al, 2012. Experiments with deliberative planning on autonomous underwater vehicles. Proceedings of IROS 2012 Workshop on Robotics for Environmental Monitoring, Vilamoura, Portugal.

Pryor et al., 1996. Planning for contingencies: A decision-based approach. In Journal of Artificial Intelligence Research, Vol. 4, pp 287-339.

Quigley, M. et al, 2009. ROS: an open-source Robot Operating System. Proceedings of ICRA workshop on open source software, Kobe, Japan, p. 5.

Sanelli et al., 2017. Short-term human-robot interaction through conditional planning and execution.

Sanner S. et al, 2010. Relational dynamic influence diagram language (rddl): Language description. In Unpublished ms.