Planning and acting framework under robot operating system

A. A Al - Moadhen¹, A. M Abdulhussein² and H. G Kamil¹
¹College of Engineering, University of Kerbala, Kerbala, Iraq
²University of Kerbala, Kerbala, Iraq
E-mail: ahmedh1333@gmail.com

Abstract. A well-known Operating System (OS) in the field of robotics is the Robot Operating System (ROS), which provides a set of tools and software libraries that are used to build many modern robotics systems. A planning system is concerned with generating a sequence of actions (plan) that can then be used by the robotic system to achieve its tasks. However, autonomous systems cannot depend only on planning systems to accomplish such tasks. In order to support such systems, it is necessary to develop an acting system, whose role is to complement the planning system by endowing the robotic system with additional flexibility to accomplish its tasks. In this paper, a state-of-the-art framework, RosPlanAct, is described. This architecture is used for embedding task planning and acting systems into the ROS. The planning section uses the environment model (planning domain) and the task specifications (planning problem) to generate an appropriate sequence of actions (plan) for any robot tasks. Then, the acting part of the system receives this plan and acts upon its schedule to achieve the specified task. Throughout this paper, there are three main issues that the actor must cope with to avoid failure. These issues are plan refinement, re-planning, and reacting to exogenous events. The specification of the RosPlanAct architecture and a case study in autonomous mobile robot are explained. The case study involves an autonomous vehicle in scenarios that clearly demonstrate the flexibility, stability, and robustness of the presented approach.

1. Introduction
In applications which are performed without, or with very little, human involvement, autonomous agents are the best choice. These agents are entities which are dependent on software to enable them to operate autonomously without explicit human involvement and which thus can control their own actions [1]. These agents may be autonomous robots in manufacturing processes or service environments, or parts of robotised systems such as spacecraft, Autonomous Underwater Vehicles (AUVs), Autonomous Unmanned Vehicles, or autonomous cars. Frequently, these agents deliberate their planning and acting functions to meet their mission requirements.

In robotics, Planning is the process of computing the required task actions and the necessary motions for a robot to achieve a given goal. Drew McDermott [2] noted that planning involves “reasoning about possible courses of execution”. Planning can thus support the robot in navigation, manipulation, preserving human safety, and gathering the necessary information to complete its task [3]. The abstract descriptions of a task’s aims and the robot's environment (domain) are provided to the robot planner by a human. Then, the planner computes the necessary sequence of actions (plan) to accomplish the given task autonomously or semi-autonomously. It is thus important for the robot to be able to plan on several interrelated levels, from the level that deals with signal control to higher-level task systems. The robot’s planning system is highly integrated with other components of the robot, and it can exploit information from the robot’s peripheral units as well as being directed by operator instructions to control the acting processes and drive the systems to meet the task requirements.

Although the current capacity of robot planning, which has been used to create several interesting robotics applications, is quite good, the distribution of robotics system in the real world is limited and there is a substantial gap between the generated plans and their execution [4]. Generally, robot planners retain an internal (initial) state and have the ability to compute new states or determine how these could change when different robot actions are applied [5].
Planning is affected by acting; however, researchers in the Artificial Intelligence (AI) field have developed many planners, but very few actors [6]. This may be one of the main reasons for the comparatively low deployment of automated planning applications. Sometimes, acting on the plan requires a reasoning process, both before and during acting, that allows the agent to predict and decide what piece of work to do and how do it, allowing it to combine several actions together in order to achieve the aims [7].

The first serious acting system, Planex, emerged during the 1970s [8], which was deployed on the Shakey robot. In this system, STRIPS-based plans were executed without action refinement, as Planex mapped plan actions into commands. The Markov Decision Process (MDP) is another popular model for probabilistic planning [9]. This model can model uncertainty about an agent’s operation and its knowledge of the surrounding environment and offers highly expressible representation. This model has also been extended in order to handle partial observability [10] and multiple cooperating agents [11]; However, solving MDPs is computationally intractable in general, and this problem has limited their application to time-sensitive tasks such as in search and rescue robots.

When offline planning cannot address all potential runtime contingencies, planning continually is required which means using different methods to interleave the processes of generating a plan and its execution, as stated in [12] and [13]. A continual plan is based on the minimised version of the initial task to allow faster amendment. A robot can switch between its planning and acting activities when online planning is needed by utilising a principled approach which describes the time and reason for that switching [12].

A language (IPLEXIL) for plan execution was illustrated in [14], enabling the user to create nodes as computational abstractions. IPLEXIL concentrates on acting and it does not share data with the planner, working solely with space applications. The ROS based execution system created by [15] is called SMASH, and its operation is based on an automata-based approach. In this system, the user can write a list of state machines hierarchically; each state is responsible for executing a specific robotic component's action.

The system in [16] offered immediate translating of plan actions into robot commands. It extended this notion to public plans, defined as a class of plans applicable in a set of states leading to the specified goal. The survey conducted by [17] showed a critical relationship between planning and acting, and explained that it is insufficient to minimise the acting to simply triggering a motor from a sensor input reactively by translating planned actions or observed events to commands.

This work attempts to close the gap between deploying full autonomy of the robotic system and real world applications by developing a new framework called RosPlanAct. This framework integrates planning and acting capabilities under ROS middleware that coordinates their activities. The following sections will describe the paper components and the main techniques and tools used. The main building blocks of RosPlanAct are the robot’s task planning and acting. This paper will aim to integrate planning and acting under ROS middleware in order to facilitate the production of the necessary commands to operate the robot intelligently.

The paper is organised as follows: Section 2 begins by laying out the RosPlanAct framework architecture and looks at how each part works. Section 3 presents the knowledge management section of the framework and examines how the environment ontology is updated. Section 4 presents the way in which the robot carries out planning and acting under the RosPlanAct framework, while section 5 describes some important insights via a case study investigating the framework behaviour both in simulation and reality. Finally, section 6 gives the conclusions, critiques the results, and offers suggestions for future work.

2. RosPlanAct Framework Architecture

The RosPlanAct framework has several components that provide the necessary software interfaces to handle various robot capabilities. The proposed framework provides some capabilities in such a way as to estimate the necessary motion and direction plan required to navigate the robot in its
environment. It further provides the means to trigger the robot, such as by performing a generated plan and executing the plan’s actions.

The RosPlanAct framework is run with domains and planners, using parts of the Planning Domain Definition Language (PDDL), such as PDDL3 [18] PDDL2.2 [19], and PDDL 2.1 [20], to automate the planning and acting processes under ROS middleware and to coordinate the activities of the low level control signals. An overview of the RosPlanAct framework is shown in figure 1.

**Figure 1.** General overview of the RosPlanAct framework which consists of Knowledge Management, Planning, and Acting ROS nodes.

RosPlanAct includes three main ROS nodes: a Knowledge Management Node (KMN), a Planning Node (PN), and an Acting Node (AN). The KMN consists of a collection of modules responsible for parsing, managing, and editing the knowledge ontology and transforming this ontology into PDDL format. The KMN is intended to collect and combine all up-to-date features of the robot state and its environment.

The PN encapsulates four main units: a Problem Generator (PG), a Planner (PL), Plan Delivery (PD), and Information Gathering (IG). The PN’s responsibilities are

- to create the robot’s initial state automatically by converting the knowledge (facts, predicates, properties) stored in the knowledge ontology into a PDDL problem model;
- to pass a PDDL problem instance and environment model (planning domain) to the planner and post-processes in order to generate the plan for the specified task, then to validate this plan;
- to deliver each generated plan to the acting node, and to decide when to update or re-plan the task in cooperation with the acting node; and
- to gather additional necessary information when the available knowledge is insufficient to plan the task.

The AN encapsulates six main units: these are the Skills Handler (SH), Plan Refinement (PR), Re-planning (RP), Reacting to External events (RE), Methods Management (MM), and Acting
Management (AM) units. Each of these units has its own functions and techniques. The AN is responsible for

- receiving the outcomes of the PN (plans) and determining the necessary skills to perform the plan’s actions;
- passing the actions, action methods, and the specified skills to the AM in order to refine them and generate the necessary commands for the execution platform;
- reacting to external events appropriately; and
- evaluating the generated commands so as to perform them or trigger the RP unit to generate a new or modified plan as required.

The KMN is responsible for establishing a knowledge ontology from the available information and any new information that can be obtained from the robot platform. This information is used to guide the low-level robot executor by being mapped to the commands or real coordinates of the waypoints in the environment. The KMN consists of three units: Ontology Parsing (OP), Ontology Management (OM), and Ontology Editing (OE). The KMN is responsible for supporting the PN to:

- construct the PDDL problem and domain files when mapping PDDL action models to ROS action messages;
- enforce the messages of ROS with data; and
- receive the new information from the robot platform and embed it into the ontology.

3. **RosPlanAct Knowledge Management Node**

To use the RosPlanAct framework, the operator should initiate the knowledge base. In this paper, the information in the knowledge base is represented as ontology. The PN uses the ROS interfaces to query the KMN about the necessary information for planning, including objects instances, predicates, or facts which have been implemented by the OM unit. Figure 2 shows an algorithm that is used to update the knowledge ontology when new information is available.

In order to take advantage of the structural similarities between many robotic planning domains, the KMN is modular. For example, as will be explained in section 5, an Asynchronous Remote-Controlled Car (ARCC) can move in an area represented as waypoints; thus, both the knowledge base and the domain contain waypoints.

The KMN interface is used by the PN to generate the PDDL problem model by applying an initial necessary state and the intended goals. This has been implemented as an interface made up of ROS services. In this case, the KMN is responsible for constructing a PDDL problem instance and planning domain, then sending these to the planning node.

4. **Planning and Acting under RosPlanAct**

4.1 **Planning**

The Planning Node receives a PDDL problem model from the KMN, and then sends this problem to a PDDL based planner, which delivers the generated plans and triggers re-planning if necessary. This is shown in figure 3. To generate the PDDL problem, the PN parses the planning domain file, then requests information from the KMN in the form of object, facts, and predicates that establish the starting state, and the current goals. Then, the problem and domain files are handed to a PDDL based planner to produce a plan. Any PDDL based planner can be embedded in RosPlanAct, allowing it to deal with the structure requirements of the domain.
This section presents the procedure to deal with a robot’s planning tasks that must be acted on continually in order to work in the physical world. Thus, the planner actions should ultimately be decomposed to sequence of commands performed by the robot system. Such decomposing is often non-trivial and requires further consultation, such as specifying how to pick up an object.
The goal is thus to integrate the PN presented in a more general architecture pursuant to automating planning and acting in robotics. For this reason, in a service and domestic robot environment, where a robot must perform several tasks such as navigation, preparing a house, and delivering things to people, tasks should be modelled and provided to the planner and AM in the planning and acting nodes, respectively. Thus, the environment model and the robot abilities are expressed together as the planning domain.

By integrating planning and acting, the robot can reach a set of goals which are implemented as a list of tasks based on its state in the world. The robot must therefore prepare and execute a sequence of actions that fulfils its aims with respect to both modelled and exogenous events and the limited availability of information about its environment.

The execution of the plan involves decomposing the actions into preliminary commands and observing the overall operation to ensure that the progress of the world state matches the robot’s expectations. Figure 4 expresses the sequence of steps from objectives to commands. In the case of inconsistencies that would prohibit the robot from performing its aims, it should update its current plan, if possible or re-plan in order to return to its normal behaviour.

Figure 4. From objective to commands
Figure 5 illustrates the proposed architecture of the acting section. This architecture has three main components that communicate and synchronise by passing messages to maintain the agent's internal state. Practically, the ROS (Robot Operating System) middleware is already used to run many robot tools as created by the robotics society [21].

The Skills Manager is responsible for coordinating the skill handler for each primitive action in the plan; it can also be obtained after refining complex action. The skill handler is used to execute a related action by decomposing it into one or several commands provided by the robot platform. The skill handler is started when it receives a primitive action and stopped when it succeeds or fails. When an action is executed, it provides feedback which expresses the status of the executed action.

The acting manager is the main focus of the acting node architecture. Its main responsibility is to execute a plan and monitor the system status. When the acting system receives the delivered plan from the planning node, the acting manager dispatches the ready-to-execute actions to the skill manager, then nominates the related skill handlers. It also has responsibility for dealing with the feedback obtained from the environment or issued by the skill manager concerning the execution of actions. When the feedback causes a contradiction between the observed and the predicted states, it informs the planning node to either update the current plan or generate new one.

![Figure 5. Proposed acting part](image_url)
5. Case Study
The workability of the proposed framework in this paper was demonstrated by offering a two-part case study that used the presented techniques. One section involved simulation under the ROS environment, while the second one was applied to a smart car\(^3\) in the real world. This means that the proposed framework was embedded into ROS messaging system and the controller of the smart car.

5.1 Working under the Simulation Environment
The planner used in this case study was the popf planner [22]. An excerpt of the planning domain is shown in figure 7, while an excerpt of the planning problem is shown in figure 6. The generated plan after triggering the planning node for the rectangle path (shown in figure 11- a), is shown in figure 8. This plan was prepared for delivery to the acting node by using the algorithm in figure 3, and the resultant delivered plan is shown in figure 9. Figure 10 shows the RosPlanAct nodes and topics under ROS environment.

\(\begin{align*}
\text{Figure 6. Excerpt of planning problem} \\
\text{Figure 7. Excerpt of planning domain}
\end{align*}\)

\(\begin{align*}
0.000: (\text{move car p1 p2}) & [0.001] \\
0.001: (\text{move car p2 p4}) & [0.001] \\
0.002: (\text{move car p4 p3}) & [0.001] \\
0.003: (\text{move car p3 p1}) & [0.001]
\end{align*}\)

\(\begin{align*}
\text{Figure 8. The generated plan} \\
\text{Figure 9. The posted actions}
\end{align*}\)

\(\begin{align*}
\text{action_id: 1} & \quad \text{action_id: 2} \\
\text{name: move} & \quad \text{name: move} \\
\text{parameters: [‘car’, ‘p1’, ‘p2’]} & \quad \text{parameters: [‘car’, ‘p2’, ‘p4’]} \\
\text{duration: 0.001} & \quad \text{duration: 0.001} \\
\text{dispatch_time: 0.000} & \quad \text{dispatch_time: 0.001} \\
\text{action_id: 3} & \quad \text{action_id: 4} \\
\text{name: move} & \quad \text{name: move} \\
\text{parameters: [‘car’, ‘p4’, ‘p3’]} & \quad \text{parameters: [‘car’, ‘p3’, ‘p4’]} \\
\text{duration: 0.001} & \quad \text{duration: 0.001} \\
\text{dispatch_time: 0.002} & \quad \text{dispatch_time: 0.003}
\end{align*}\)

\(^3\)For more details about the car, see https://www.sunfounder.com/robotic-drone/smartcar/arduino-smart-car-v2-0/smart-car-kit-v2-0-for-arduino.html
5.2 Experiment in the Real World

In this section, the presented approach was examined in a case study that investigated the ability of a smart car to plan and act in the real world. The scenario was run in physical attempts and involved moving the car from source to destination points in four different paths. These paths were a rectangle, triangle, polygon, and polylines, as shown in figure 11. In the diagram, every point is represented by its Cartesian coordinate \((x, y)\).
Figure 11. Car paths: a) Rectangle path, b) Triangle path, c) Multiline Path, and d) Polygon path

Figure 12 shows the smart car, and figure 13 shows the smart car circuit. The experiments were repeated for 30 times for each path; figures 14 to 17 show the results of execution.

Figure 12. Smart Car

Figure 13. Car Circuit
Figure 14. The result of executing the rectangle path in three different modes: execution without planning (Exe), execution with planning only (ExePlan) and execution with planning and acting (ExePlanAct).

Figure 15. The result of executing the triangle path in three different modes: execution without planning (Exe), execution with planning only (ExePlan) and execution with planning and acting (ExePlanAct).
Figure 16. The result of executing the polyline path in three different modes: execution without planning (Exe), execution with planning only (ExePlan) and execution with planning and acting (ExePlanAct).

Figure 17. The result of executing the polygon path in three different modes: execution without planning (Exe), execution with planning only (ExePlan) and execution with planning and acting (ExePlanAct).

It can be noted from these diagrams that execution without the support of planning and acting systems led to a high number of failed trials with fewer successful trials. Execution with support from both planning and acting systems led to a higher number of successes with fewer fails. This suggests that when robotics systems are supported by planning and acting abilities, they can:

1. refine complex actions into more primitive actions and then translate these into the required commands to be executed directly;
2. update a plan or trigger the planner to re-plan and generate a new plan where the generated plan cannot accomplish the given task; and
3. access a library of skills that are used to recover the robot from unexpected situations.
The mission attempted to achieve persistent autonomy, with the goal being to show a reduced frequency of aid requests during planning a trajectory and acting upon this. This requires long-term capabilities on the part of the car, which can be achieved by planning long-horizon activities, and scheduling sequences of actions that will achieve its objectives. In order to ensure that the planning and acting models remain robust during robot operations requires careful matching of the robot and its environment models to the real world, including dynamically updating the models based on continuous sensing actions.

6. Conclusions and Suggestions for Future Work
In this paper, the RosPlanAct framework under ROS was explained. This framework is proposed as a solution for embedding a task planner, based on the PDDL family, and actor into a robotic system. From the results, the robot with planning and acting abilities displayed a high degree of autonomy with a higher number of successful trials than the robot utilising planning with execution or only execution. This advantage was obtained because using acting endows the robotic system with the ability to update its plan or generate a new one instead of returning a fail.

The RosPlanAct framework can be regarded as a step towards generalising and standardising the integration of PDDL planner and actor systems under ROS middleware. This architecture has shown some success in applications related to robot navigation. The case study presented autonomous behaviour in a smart car and demonstrated the flexibility and robustness in its operation derived from plan updating and re-planning.

This framework requires more investigation on using time periods during plan construction and on acting upon the generated sequence of actions. The framework also requires testing in manipulation action situations such as grasping.

7. References
[1] Wooldridge M and Jennings N R 1995 Intelligent agents: theory and practice Knowl. Eng. Rev. 10 115–52
[2] McDermott D 1992 Robot Planning AI Mag. 13 55–79
[3] LaValle S M 2006 Planning Algorithms (Cambridge, UK: Cambridge University Press)
[4] Alterovitz R, Koenig S, Likhachev and Maxim 2016 Robot Planning in the Real World: Research Challenges and Opportunities AI Mag. 37 76–84
[5] Ghallab M, Nau D and Traverso P 2004 Automated Planning: Theory and Practice (Morgan Kaufmann)
[6] Ghallab M, Nau D and Traverso P 2014 The actor’s view of automated planning and acting: A position paper Artif. Intell. 208 1–17
[7] Ghallab M, Nau D and Traverso P 2016 Automated Planning and Acting (Cambridge University Press)
[8] Fikes R E 1971 Monitored execution of robot plans produced by STRIPS IFIP Congress
[9] Puterman M L 2005 Markov Decision Processes: Discrete Stochastic Dynamic Programming (New York, NY, USA: John Wiley & Sons, Inc.)
[10] Cassandra A R, Kaelbling L P and Littman M L 1994 Acting optimally in partially observable stochastic domains 12th National Conference on Artificial Intelligence (AAAI’94)
[11] Bernstein D S, Givan R, Immerman N and Zilberstein S 2002 The complexity of decentralized control of Markov decision processes Math. Oper. Res. 27 819–840
[12] Brenner M and Nebel B 2009 Continual planning and acting in dynamic multiagent environments Aut. Agent Multi-Agent Syst. 19 297–331
[13] DesJardins M E, Durfee E H, Ortiz C L and Wolverton M J 1999 Continual planning and acting in dynamic multiagent environments AI Mag. 20 13–22
[14] Verma V, Estlin T, Jönsson A, Pasareanu C, Simmons R and Tso K 2005 Plan Execution Interchange Language (PLEXIL) for Executable Plans and Command Sequences International
Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)

[15] Bohren J, Rusu R B, Jones E G, Marder-Eppstein E, Pantofaru C, Wise M, Mösenlechner L, Meeussen W and Holzer S 2011 Towards Autonomous Robotic Butlers: Lessons Learned with the PR2 2011 IEEE International Conference on Robotics and Automation Shanghai International Conference Center pp 5568–5575

[16] Schoppers M J 1987 Universal plans for reactive robots in unpredictable environments 10th International Joint Conference on Artificial Intelligence

[17] Ingrand F and Ghallab M 2017 Deliberation for autonomous robots: A survey Artif. Intell. 247 10–44

[18] Gerevini A and Long D 2005 Plan Constraints and Preferences in PDDL3 (Italy)

[19] Edelkamp S and Hoffmann J 2004 PDDL 2.2: The Language for the Classical Part of IPC-4 in Proceedings of the International Planning Competition International Conference on Automated Planning and Scheduling (Whistler 2004) pp 1–7

[20] Fox M and Long D 2003 PDDL2.1: An extension to PDDL for expressing temporal planning domains J. Artif. Intell. Res. 20 61–124

[21] Quigley M, Conley K, Gerkey B P, Faust J, Foote T, Leibs J, Wheeler R and Ng A Y 2009 ROS: an open-source Robot Operating System ICRA Workshop on Open Source Software

[22] Coles A, Coles A, Fox M and Long D 2010 Forward-Chaining Partial-Order Planning Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)