The Design of Interactive Framework for Space-Exploration Robotic Systems

Wei Shi(✉), Shengyi Jin, Yang Zhang, Xiangjin Deng, Yanhong Zheng, Meng Yao, and Zhihui Zhao

Beijing Institute of Spacecraft System Engineering, Beijing 100092, China
20115266@qq.com

Abstract. The deep space-exploration spacecraft or robot need to perform missions in complex and harsh environments and far away from the earth. Restricted by large communication delay and low-bandwidth, the operator on the earth can't interact frequently with spacecraft or robot. For the reasons, the system design of spacecraft is required to powerfully autonomous and reliable. This paper is based on the application of the sampling robot for extraterrestrial planets, described the design of interactive framework for task-level Command control of robotic systems, establish a standard planning operators(POs) sets that can cover the operating space basically, and shows how to improve system autonomy through interactive planning and learning. Under this framework designation, with a small amount of task-level command and state feedback telemetering between the operator on the earth and the spacecraft, it can meet the mission.

Keywords: Interactive-framework · Planning operators (POs) · Space-exploration robotic systems · Task-level command

1 Introduction

Space is fundamentally one of the most challenging domains that humans have ever tried to explore. For exoplanet soil/rock sampling tasks, restricted by incomplete initial environment information, uncertain execution duration and large communication delay [1], and so on. For this reasons, space- exploration spacecraft or robot is required to be powerfully autonomous and reliable. However, it is limited by the current computing and storage level of computers on spacecraft, the development of artificial intelligence technology, and the high reliability and safety requirements of spacecraft missions, making spacecraft have powerfully autonomous is a great challenge. Lightweight, low volume, simple and convenient application need to be considered. This study, we propose a design of interactive framework for application of space- exploration robotic systems.

2 Related Work

With the continuous development of space action, the field of exploration is expanding. Space-exploration spacecraft have also evolved from simplicity to complexity, from a single mission to a variety of tasks, from relying mainly on manual operation to gradually autonomous development. “Surveyor” (1966–198), “Luna16/20” (1970–1972)”, “Luna24” (1976), “Viking” (1976), “Sojourner” (1997), “Hayabusal” (2003), “Mars express” (2003), “Rosetta” (2004), “Curiosity” (2011), “Jade Rabbit” and so on. These successful space exploration actions represent a significant trend.

In order to realize the goal of space exploration spacecraft/robot with autonomous and reliability, a complete system framework is necessary. Base on the design of framework, through experiments and knowledge learning, the intelligent level of spacecraft is gradually optimized and improved to meet the application requirements of various missions.

The US Air Force Research Laboratory (AFRL) Space Vehicles Directorate, the team of Orbit Logic, PnP Innovations, and Emergent Space are developing an Autonomous Planning System (APS) framework. The core of APS is specialized Autonomous Planning Agents (SAPAs) and Master Autonomous Planning Agent (MAPA), SAPAs generate plans, and the plans are integrated by [2].

Many artificial intelligence techniques have been used to learn and reason in a way that humans describe the world [4, 5]. In the described method, a set of POs is generated using a specific use case developed by the expert system, and these operators are then optimized using an improved PRODIGY planner [6] and the PO is optimized using a modification of the version space method named OBSERVER. In [7], a TRIAL plan and learning strategy is proposed. It learns and optimizes POs when generating plans and executing plans. Until the system deadlocks, external teachers are called to participate in controlling the completion of tasks. In [8], a system architecture design solution named EXPO was proposed. This system uses PRODIGY as a benchmarking program to improve knowledge in multiple domains.

The characteristics of the space exploration mission determine that the robot system has a high rate of undefined operations in an unknown environment, and it is extremely important to be able to react and process quickly when encountering undefined operations. Therefore, we place greater emphasis on the online learning capabilities provided by the architecture. Teacher intervention occurs during the execution of the task, providing only the operational instructions that need to be performed, and returning the control of the task to the system after the operation. Our strategy for action guidance is similar to other types of tele-operation technology [6]. The robot system provides a set of functions (actions) in advance, and the teacher (served by the ground operator) simply indicates which functions (actions) to perform in a given situation. After instructing actions, the robot is allowed to perform actions and generate state transitions for learning [9]. In particular, the teacher does not supervise the behavior and does not provide clues as to the relevant attributes of the world that should appear in the successful execution of the task [10]. This greatly reduces the number of human-computer interactions, and it should be noted that, depending on the complexity of the application, indicating all relevant properties of the world in every possible situation faced by the robot may be complicated. To alleviate this burden, we propose a method of automatically learning world-related attributes that should exist to allow POs to perform successfully [11].

3 The Design of Interactive Framework

The framework design described in this paper provides two different modes of operation: the ground development mode and the task execution mode. The ground development model is mainly used for training and improving system intelligence, and the task execution mode is used for deep space exploration of spacecraft on-orbit task execution. The core of the two model frameworks is the planning module and the learning module. The planning module searches, generates, and optimizes the sequence of motion controls needed to achieve the mission objectives based on established execution rules and constraints. The learning module is used to interact with the operator to create, edit, and repair these execution rules and constraints.

Schematic of the Interactive framework is shown in Fig. 1. The framework includes planning module (PM), learning module (LM), Teaching module (TM), Operation basic set module(OBM), Control rule set module (CRM), Emergency response module (ERM), Control execution interface module (EIM), execution module (EM), environment Perception and modeling module (EA&M). The space-exploration spacecraft system, the application examples corresponding to two different operation modes in the framework are shown in Fig. 2.

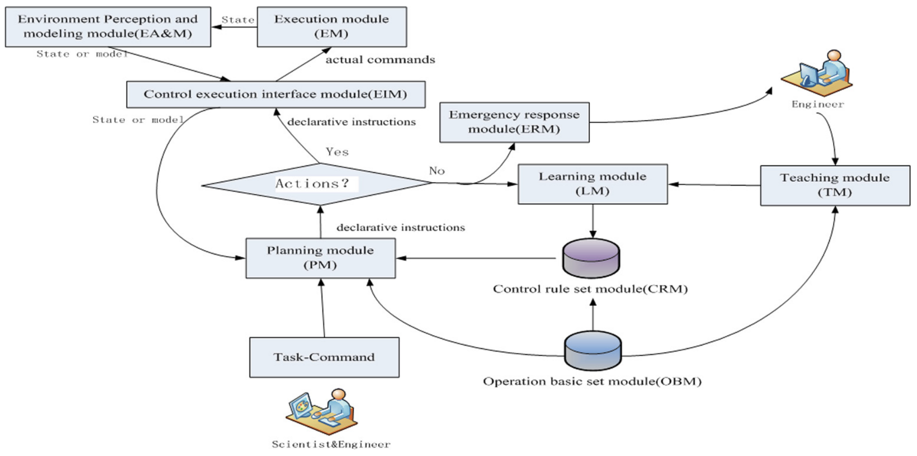
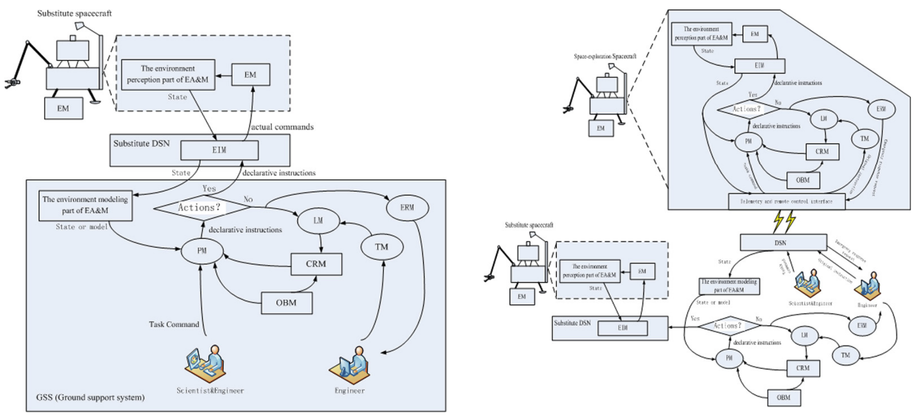


Fig. 1. Schema of interactive framework


Substitute spacecraft
The environment perception part of EA&M
EM
State
actual commands
Substitute DSN
EIM
declarative instructions
The environment modeling part of EA&M
State or model
Yes
No
Actions?
LM
ERM
PM
TM
CRM
OBM
Task Command
Scientist&Engineer
Engineer
GSS (Ground support system)
Space-exploration Spacecraft
Telemetry and remote control interface
DSN

(a) the ground development mode (b) the ground development mode

Fig. 2. Schematic of two mode

The ground development mode is used in the development stage before the mission is executed. The model is shown in Fig. 2(a). The system consists of Substitute spacecraft, Substitute DSN (Deep Space Network) and GSS (Ground support system). The EM and the environment perception part of EA&M functions are provided by Substitute spacecraft, and the EIM function is provided by Substitute DSN. The PM, LM, TM, OBM, CRM, ERM and the environment modeling part of EA&M functions are all included in the GSS. Initially, the OBM consists of a limited number of meta-operations that described in accordance with the prescribed the declarative instruction. The manual operator interacts with the LM of the system by the EIM to form the original CRM. The PM is based on this CRM, receive task-command and planning to generate an initial operation sequence, through the EIM controls the EM sequentially complete the state transition specified by the operation sequence. Meanwhile the EA&M continuously monitors and establishes or updates the environmental state model, and feeds back the latest environmental state model to the PM, which executes according to the feedback status update or control operation sequence. If the planning module finds that the feedback status is undefined or does not find a solution in the CRM, then the ERM is activated. The operator interacts with the LM of the system to complete the CRM and exit the ERM. The PM continues to complete the control execution of the planning and operation sequence until the task-command requirements are met.

The mission execution mode is shown in Fig. 2(b). Unlike the ground development model, the system consists of spacecraft, DSN, and GSS in mission execution mode. In addition to EM and the environment perception part of EA&M functions, the spacecraft is also loaded with PM, LM, TM, ERM and optimized OBM and CRM. The environment modeling part of EA&M which need large computational complexity and large storage requirements,is retained in the GSS. The GSS also includes a complete ground development mode framework to synchronize Substitute spacecraft with the mission spacecraft state.

When the system is running, the task commands received by the PM may be task commands such as "detect and collect samples at target A", requiring the system to understand these task commands and translate them into instructions that control the EM. The declarative description of the states of the world that currently exists is need to be provided. These descriptions need to be unambiguous and can be programmed to generate control sequences for controlling EM without confusion. Based on this, the planner attempts to find a sequence of actions that converts the current state to a state that satisfies the target of the task command specification.

3.1 The Declarative Description of Actions and States

In this section, we introduce the declarative description of actions and states in this study. Here we define a finite states space Š and a finite actions space Â. State s∈Š and $s = \{d_1, d_2 \ldots\ldots d_m\}, m \in N$. d_i is a subdivision unit that describes a state and a predicate is map the properties. 's' is regard as a state that can be described by multiple logic predicates. A action a∈Â may take arguments representing the objects to which the action is applied.

In the planning, the planner is given a description of the initial state, 's_ini' and the target state description 'Des'. Using these elements, the planner searches for sequences of actions that allow the changes form 's_{ini}' reach to the goal '*Des*'. This search is performed using a set of POs, '$\widetilde{PO}$', each of which encodes the expected changes after performing an action. A PO is represented as

$$p_i = \{s_p, a, e, P\} \tag{1}$$

Where 's_p' is the states of precondition requirements, 'a' is the action, 'e' is the expected results, and 'P' is a probability estimate, which indicates the probability of reaching the 'e' when 's_p' is observed and action 'a' is executed.

$$P = \Pr(e | s_p, a) \tag{2}$$

A PO encodes the changes that should be observed when an action 'a' is executed. The 's_p' contains the initial values of the state descriptors changed by the action 'a', while the 'e' contains their final values. The action 'a' describes the action that needs to be executed. For example, a 'PO' for an action of move the robot end actuator from 'A' position to 'B' position by 'default' mode and description of 'velocity' may have the following parts:

$$\begin{gathered} s_p = \{Poweron(\text{robot}),\ Ishold(\text{robot}),\ At\text{position}(\text{A})\}, \\ a = \text{Move}(\text{robot.end\_actuator, 'default', velocity}),\ e = \{At\text{position}(\text{B})\} \end{gathered} \tag{3}$$

Where $Poweron(\text{robot})$ is a predicate that takes a value '*true*' when 'robot' is power on, $Ishold(\text{robot})$ is a predicate that takes a value '*true*' when robot is no move and hold on at the current position, $At\text{position}(\text{A})$ is a predicate that takes a value '*true*' when the current position of the robot end actuator is at 'A'. Where $\text{Move}(\text{robot.end\_actuator, 'default', velocity})$ is a predicate that move the robot end actuator with 'default' mode and 'velocity'. And *At*position(B) is a predicate with a value *true* when the robot end actuator at 'B' position. In this case, the state descriptor that changes with the action is *At*position(A). However, for this change to occur, it is also necessary that robot is hold on, which is a fact specified by the descriptor *Ishold*(robot) at remains unchanged by the action.

The PO provides a concise description of the action state and execution results. Of all the descriptors associated with completing the task, the particular PO only considers those descriptors associated with its effect, and other descriptors may be ignored or ignored. For instance, if the PO in (3) is used in a task that also requires other actions, e.g. the current position is not exactly at the 'A' position, a private PO, $p_k = \{s_k, a_k, e_k, P_k\}$, is need be completed

$$\begin{aligned} \mathrm{s_k} &= \{\mathrm{Poweron(MA), Ishold(MA), not(Atposition(A))}\} \\ \mathrm{a_k} &= \mathrm{Move(MA.end\_teminal), e_k} = \{\mathrm{Atposition(A)}\} \end{aligned} \tag{4}$$

This is because the a private PO, 'p_k', not relevant to 'move the robot end actuator from 'A' position to 'B' position'. In general, the number of descriptors used in the state representation is much larger than the number of descriptors used in a particular PO [16].

In the proposed framework, not all of the POs are available for planning and some will only be handled by the learning method. We make this distinction when needed. The training instances for learning comprise state transitions of the form:

$$\mathrm{p_t} = (S_t, a_t, S_{t+1}) \tag{5}$$

Where t is the time step, s_t is the state before executing the action, a_t is the executed action, and s_{t+1} is the state after executing the action. A *positive* instance $\mathrm{p}_{t,i}^{+}$ for the PO $p_i = \{s_i, a_i, e_i, P_i\}$ is defined as that where all of the predicates in the precondition are observed in the state before the execution of the action, the action coincides with the action of the state transition, and the predicates in the effect are observed in the state after the execution of the action:

$$\mathrm{p}_{t,i}^{+} = \{s_i \subseteq S_t, a_i = a_t, e_i \subseteq S_{t+1}\} \tag{6}$$

By contrast, a *negative* instance $\mathrm{p}_{t,i}^{-}$, represents a state transition where the PO was applied but the expected changes did not occur.

$$\mathrm{p}_{t,i}^{-} = \{s_i \subseteq S_t,\ a_i = a_t,\ e_i \not\subseteq S_{t+1}\} \tag{7}$$

We say that an operator $\mathrm{p_i}$ covers an instance when $s_i \subseteq S_t, a_i = a_t$.

3.2 The Algorithm Description

The main algorithmic description of the interactive framework is presented in Algorithm 1. Note that the initial set of POs could be the empty set. The predefined operational description has been included in the OBM is defined as P_{predef}, the description of the operation that was learned has been included in the CRM is defined as $P_{learned}$.

The main algorithm includes two key steps algorithm, $\widetilde{PO} \longleftarrow Learned(\widetilde{PO}, p_{t-1})$ and $A = \text{FindPlan}(S_{ini}, \widetilde{PO}, Des)$. FindPlan Algorithm can use binary tree search algorithm, here we focus on the *Learned* algorithm.

```
Algorithm 1 description
 Specify goal **Des**
 Initialize PO set $\widetilde{\boldsymbol{PO}}$ $(\widetilde{PO} \subseteq \emptyset$ or $P_{learned} \cup P_{predef} \subseteq \widetilde{PO})$;
 $\boldsymbol{S_{t-1}} \longleftarrow \emptyset$;  $\boldsymbol{a_{t-1}} \longleftarrow \emptyset$;
 Initialize a plan set $\boldsymbol{A} \longleftarrow \emptyset$;
 While  Not(End of plan)
   Get current state $s_t$,Generate training instance  $\boldsymbol{p_{t-1}} = (\boldsymbol{S_{t-1}}, \boldsymbol{a_{t-1}}, \boldsymbol{S_t})$;
   Update PO set with current instance  $\widetilde{\boldsymbol{PO}} \longleftarrow \boldsymbol{Learned}(\widetilde{\boldsymbol{PO}}, \boldsymbol{p_{t-1}})$;
   Set the initial state  $\boldsymbol{S_{ini}} \longleftarrow \boldsymbol{S_t}$;
   Find a series of plans(can be a single plan) to reach Des from current state
 $\boldsymbol{s_{ini}}$, A = FindPlan$(\boldsymbol{S_{ini}}, \widetilde{\boldsymbol{PO}}, \boldsymbol{Des})$
   If $Des \subseteq s_{ini}$
       End of plan;
   Else
       If $A = \emptyset$
           Request an action from the teacher for the ongoing task, $\boldsymbol{a_t} \longleftarrow Teacher(S_{ini}, \boldsymbol{Des})$;
       Else $\boldsymbol{a_t} \longleftarrow A(1)$;
       End If
       Execute $\boldsymbol{a_t}$ and make t++;
   End if
 End while
```

4 Learning and Planning Operators

Learning and planning are the most important components of the interactive framework. The following sections focus on these two aspects.

4.1 PO Evaluation

How to evaluating each PO to determine its probability in Eq. (2) is a very important issue. In our application, the evaluation of each PO is strictly limited to the current state and the risk in the implementation of the PO. we divide the PO in the task into two categories.

The first category, $p_{def} = \left\{s_{def}, a_{def}, e_{def}, P\right\}$, p_{def} can be verified by a large number of tests, And when $s_{def} \in S_t = e_{t-1}$, Calculate P according to the formula

$$P = \frac{n^+}{n^+ + n^-} \tag{8}$$

Where n^+ is the number of positive experience instances covered by the PO, n^- is the number of negative instances covered.

The second category, $p_N^1 = \left\{s_{def}^+, a, e, P\right\}$, $p_N^2 = \left\{s_t, a, e_{def}, P\right\}$, where p_N^1 represents a positive experience instances state, by executing action a, reaching a expected results. p_N^2 represents an inverse operation of p_N^1 by executing action a to reach a positive experience instances state from an arbitrary state, So the probability P of p_N^1 and p_N^2, generally using the distance between e and s_{def}^+, s_t and e_{def} as the estimation evaluation method, at the same time to ensure that the robot movement safety type will be constrained according to the configuration of the robot and the recognition environment model, according to the configuration of the robots assuming a function:

$$\mathrm{F}(\mathrm{X}, \mathrm{Y}_t) = \begin{cases} 0, \forall \mathrm{x}_i \in \mathrm{X}, \forall \mathrm{y}_j \in \mathrm{Y}_t, & \min_{\mathrm{x}_i \in \mathrm{X}, \mathrm{y}_j \in \mathrm{Y}_t} \left|\mathrm{y}_j^2 - \mathrm{x}_i^2\right| > C \\ D, & other \end{cases} \tag{9}$$

Where X describes the robot's own configuration description and Y_t describes the configuration of the moving part on the robot in the current state. When the function is 0, the moving part of the robot is described in the current state. No contact is made with the body structure. In addition, we also need to define the function:

$$\mathrm{G}(\mathrm{E}, \mathrm{Y}_t) = \begin{cases} 0, \forall \mathrm{e}_i \in \mathrm{E}, \forall \mathrm{y}_j \in \mathrm{Y}_t, & \min_{\mathrm{x}_i \in \mathrm{X}, \mathrm{y}_j \in \mathrm{Y}_t} \left|\mathrm{y}_j^2 - \mathrm{e}_i^2\right| > C \\ D, & other \end{cases} \tag{10}$$

Where E describes the reconstruction description of the task environment, and when the function is 0, the moving part of the robot is not generated in the current state and the task execution environment contact. Here, P can be expressed without consider the measurement error:

$$P = \Pr\left(e \middle| s_p, a, \mathrm{F}(\mathrm{X}, \mathrm{Y}), \mathrm{G}(\mathrm{E}, \mathrm{Y})\right) = \int_t^{t+1} \frac{1}{2} E^{\left(-\frac{\left(y_t^2 - s_{def}^{+\,2}\right)^2}{2}\right)} \mathrm{dt} \times \mathrm{F}(\mathrm{X}, \mathrm{Y}) \times \mathrm{G}(\mathrm{E}, \mathrm{Y}) \tag{11}$$

In our application, the second category PO is verified once, the probability P_N of *OP* p_N is can be temporarily looked as the first category PO. If p_N is positive experience instances, $n^+ \leftarrow n^+ + 1$. When the environment changed, calculate P_N according to the formula:

$$P = \Pr(e|\mathrm{G}(\mathrm{E},\mathrm{Y})) = P_N \times \prod\nolimits_t^{t+1} \mathrm{G}(\mathrm{E},\mathrm{Y}_t)dt \qquad (12)$$

4.2 PO Generation

The states processing rules contained in CRM are limited and cannot completely cover or cope with complex and variable state transitions in real tasks. PM search generation POs that reflect the best state transition is a typical optimal path planning problem. Finding the optimal path in a large number of states space requires a lot of computational time investment. In order to improve system reliability and reduce time investment, ERM is started to generate new POs in two cases. (1) When planning, it is found that there is undefined actions definition, and the corresponding operation rules need to be added in the CRM. (2) When the execution of a PO has an unexpected effect.

Generation from an Action Instruction

Undefined actions may prevent the generation of plans, thereby triggering an action instruction request to ERM be started. After executing the instructed action, a_{inst}, a new PO is generated to fill the $P_{learned}$, and $P_{learned} \subseteq \widetilde{\mathrm{PO}}$ For descriptors this PO, first, the changed descriptors are extracted from the observed state transition,

$$\begin{aligned} s_c &= \{s_i \in S_t | s_i \notin S_{t+1}\} \\ e_c &= \{s_i \in S_{t+1} | s_i \notin S_t\} \end{aligned} \qquad (13)$$

Where s_c and e_c are the sets of the changed descriptors, respectively, S_t is the state before and S_{t+1} is the state after executing the action. Then, a new PO, $\mathrm{p}_{\mathrm{inst}}$, is created by using s_c, a_{inst} and e_c as the precondition, action, and effect parts, respectively. The new PO becomes available immediately for planning. However, the new PO only considers the changed descriptors in its precondition, so some unchanged causative descriptors may be missing. In this case, the execution of the newly generated PO may have an unexpected effect.

Generation from Unexpected Effects

Without considering the system failure, the PO extracted and executed from the set of operations defined by OBM and CRM has an unexpected effect, indicating that there is a constraint defect in the initial state definition of the execution of the PO. When the system state does not satisfy the real execution condition of the PO, the PM plans and starts the execution of the PO according to the state definition of its.

The refinement process starts by bringing together all the POs that code the same changes an the failing PO set, p_{exe}, which have been accumulated in the PO set: $\widetilde{\mathrm{PO}}_{exe} = \{p_{exe} | a_i = a_{exe}, e_i = e_{exe}\}$, where a_{exe} and e_{exe} are the action and effect parts of p_{exe}, respectively. In this case, first try to find a candidate OP, $p_c = \{e_{exe}, a_c, e_c \in s_t\}$, if such a candidate p_c can be found, then add p_c to the CRM. Since e_{exe} is an unexpected state, such a candidate OP cannot be found which is a high probability event to bring the system control into a transition state. The transition state is generally selected from the safe parking state, which is a limited state set S_{interim}. And the safe parking state selection strategy is handled by the proximity principle $s_k = \min_{s_i \in S_{\text{interim}}} \left| e_{exe}^2 - s_i^2 \right|$,The problem turns into finding a PO set $\overset{\smile}{\text{PO}}_b$:

$$\overset{\smile}{\text{PO}}_b = \{ p_j | s_j = e_{exe}, a_j = a_b, e_j = s_k \} \tag{14}$$

Learner Algorithm $\boldsymbol{\overset{\smile}{PO}} \leftarrow \boldsymbol{Learned}(\boldsymbol{\overset{\smile}{PO}}, \mathrm{p_{t-1}})$ description as follow:

```
Learner Algorithm description
Update probabilities $P_i$ of POs in $\overset{\smile}{PO}$ using instance $p_{t-1}$
Get action $a_{t-1}$ from $p_{t-1}$ and get executed PO $p_{exe}$ from $\overset{\smile}{PO}$;
If $a_{t-1}$ was instructed
    Generate from an action instruction OP $p_{inst}$ and set $p_{inst}$ as available for planning $\overset{\smile}{PO} \leftarrow \{\overset{\smile}{PO}, p_{inst}\}$ , Update the rules of $p_{inst}$ into CRM;
Else
    Get state $s_t$ from $p_{t-1}$, and set $\overset{\smile}{PO}_c \subseteq \emptyset$
    If $e_{exe} \notin s_t$
        Generate a candidate operation $p_c$ , $p_c = \{e_{exe}, a_c, e_c \in s_t\}$
        If $\exists p_c$
            Set $p_c$ as available for planning $\overset{\smile}{PO} \leftarrow \{\overset{\smile}{PO}, p_c\}$ ;
            Get executed PO $p_c$ from $\overset{\smile}{PO}$, update the rules of $p_c$ into CRM;
        Else
            Choose the right safe parking status, $s_k = \min_{s_i \in S_{\text{interim}}} |{e_{exe}}^2 - {s_i}^2|$
            While $e_i \notin s_k$
                For make the system state $e_{exe} \overset{\Delta}{\rightarrow} s_k$,
                Generate m candidate operations $\overset{\smile}{PO_c} \leftarrow \{\overset{\smile}{PO_c}, p_{i(1\sim m)}\}$;
                While m>0
                    Select the OPs with the highest probability in $\overset{\smile}{PO}_c$;
                    Use formula $w = \mathrm{GetIndex}(\max_{j \in i_c} P_j)$ ;
                    Set $p_w$ as available for planning $\overset{\smile}{PO} \leftarrow \{\overset{\smile}{PO}, p_w\}$ ;
                    Move $p_w$ from $\overset{\smile}{PO}_c$ and m--;
                End While
                Set other $p_c \subset \overset{\smile}{PO}_c$ as not available for planning;
            End While
            Update the rules of $\overset{\smile}{PO}$ into CRM;
            Planning new POs for $s_k \overset{\Delta}{\rightarrow} s_t$;
        End if
    End if
End if
```

5 Conclusion

We integrated the artificial intelligence technology into space-exploration robotic systems design effectively. In this study, our main goal is to design a framework of robotic systems that can be commonly applied in the field of space exploration, which make the robot system can autonomously plan, made decisions, and complete the tasks required by the command after receiving task-level commands. In this way, we propose to use simple task-level commands to trigger task-related operations, while simple fast learning can be used to generate POs from observed state transitions to quickly increase robot autonomy.

References

1. Colby, M., Yliniemi, L., Tumer, K.: Autonomous multiagent space exploration with high-level human feedback. J. Aerosp. Inf. Syst. **13**(8) (2016)
2. Ella, H., Josh, N., Marc, S., Doug, G., Tim, E., Ken, C.: Onboard Autonomous Planning System, SpaceOps Conferences 5–9 May 2014, Pasadena, CA SpaceOps 2014 Conference (2014)
3. Truszkowski, W.F., Hinchey, M.G., Rash, J.L., Rouff, C.A.: Autonomous and autonomic systems: a paradigm for future space exploration missions. IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. **36**(3), 279–291 (2006). https://doi.org/10.1109/tsmcc.2006.871600
4. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robot. Auton. Syst. **57**(5), 469–483 (2009)
5. Veloso, M., Bala, J., Boloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Dzeroski, S., Fisher, D., Fahlman, S., et al.: The NONK's problems: a performance comparison of different learning algorithms, Technical report CMU-C5-91-197. Carnegie-Mellon University (1991)
6. Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., Blythe, J.: Integrating planning and learning: the prodigy architecture. J. Exp. Theor. Artif. Intell. **7**(1), 81–120 (1995)
7. Benson, S.: Inductive learning of reactive action models. In: Proceedings of the Twelfth International Conference on Machine Learning, pp. 47–54. Morgan Kaufmann (1995)
8. Gil, Y.: Learing by experimentation: incremental refinement of incomplete planning domains. In: Proceedings of the Eleventh International Conference on Machine Leaning (1994)
9. Rybski, P., Yoon, K., Stolarz, J., Veloso, M.: Interactive robot task training through dialog and demonstration. In: 2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 49–56. IEEE (2007)
10. Nicolescu, M., Mataric, M.: Natural methods for robot task learning: instructive demonstrations, generalization and practice. In: Proceedings of the Second International Joint Conference on Autonoumous Agents and Multiagent Systems, pp. 241–248. ACM (2003)
11. Polushin, I.G., Dashkovskiy, S.N.: A small gain framework for networked cooperative teleoperation. In: Proceedings of the 8th IFAC Symposium on Nonlinear Control Systems, pp. 90–95. Bologna, Italy (2010)
12. Agostini, A., Torras, C., Worgotter, F.: Efficient interactive decision-making framework for robotic applications. J. Artif. Intell. **247**, 187–212 (2017)