Multi-agent System Architectural Aspects for Continuous Replanning

Carlos Joel Tavares[1], **Célia Ghedini Ralha**[1]

[1]Computer Science Department – Exact Sciences Institute – University of Brasília
Campus Darcy Ribeiro, 70.904-970 Brasília, Brazil

carlosjoel.tavares@gmail.com, ghedini@unb.br

Abstract. *Robots' coordination to achieve the system's goal is one of the challenges that complex Multi-Robot Systems (MRS) encounter. One could use automated planning (AP) to better face this challenge by diminishing problems and continually correcting the execution when failures occur. Some works in the literature try to fix this problem, but there are still few, and there's not much analysis between them. This work implements a Multi-Agent System (MAS) to simulate an MRS mission using a MAS architecture integrated with AP illustrated with space resource gathering robots. The results show the importance of the ability to plan recovery and research in complex space missions field.*

1. Introduction

Multi-robot Systems (MRS) are complex and need real-world environment execution [Klavins 2004], [Aziz et al. 2021]. Nevertheless, it is not always feasible to prepare for all environmental changes before the deployment of the system. Moreover, coordinating heterogeneous robots is a demanding task. One viable solution that creates the optimal plan recovers it when a failure occurs, and diminishes that demand is Automated Planning (AP). Therefore, the process of plan recovery becomes needed when systems run in dynamic environments [Schmitt et al. 2019], [Moreira and Ralha 2021b], [Moreira and Ralha 2022b], [da Silva and Ralha 2023].

The indispensable coordination of robots needed to achieve the system's goal is one of the many difficulties of MRS [Verma and Ranga 2021]. MRS' coordination, communication, and interaction between agents pose a problem similar to Multi-Agent Systems (MAS) [Wooldridge 2009], [Weiss 2016], [Salzman and Stern 2020]. While the formal definition of a planning problem, which includes a tuple composed of actions, prepositions, initial state, and goal, helps to mitigate this problem, in classical planning, dynamic environments are not the focus, and such are the MRS's real-world environments. While planning solutions focus on changes that derive from internal actions, dynamic-focused solutions also focus on exogenous actions and action failures. That inserts the need for plan recovery since that plan can become unfeasible many times. This complex scenario relates to Multi-Agent Planning (MAP), which involves the coordination of resources and activities of many agents [Komenda et al. 2016], [Moreira and Ralha 2021a], [Moreira and Ralha 2022a].

There are studies focused on problems of coordination aligned with planning [Cashmore et al. 2015], [González et al. 2020], [Bischoff et al. 2021], [Martín et al. 2021], [Lesire et al. 2022]. The proposed solutions in the studies create a centralized analysis of the environment to coordinate the plan. The studies of MRS analyze many aspects like goal decomposition, task allocation strategies, quality of attributes/plan adaptation using probabilistic and temporal planning, interactive coordination of heterogeneous robotic teams, relating architectures, frameworks, and robot operating systems. However, the recovery of plans in dynamic environments is seldom the focus. Even so, the literature discusses solutions to the adaptation of robotic missions [et al. 2021], [Carreno et al. 2022], design patterns of architectures to heterogeneous robots [Rodrigues et al. 2022] and planning agent architectures [Silva et al. 2020], [Magnaguagno et al. 2022].

Multi-robot planning in space robotics is crucial for enhancing efficiency, reliability, and productivity in space missions [Sun et al. 2023]. By enabling multiple robots to work collaboratively, tasks can be completed faster and more effectively, with specialized robots handling specific functions [Basmadji et al. 2020]. This approach provides redundancy, ensuring mission continuity even if one robot fails, and facilitates the execution of complex tasks that require precise coordination. This work's main contribution is the implementation of a MAS simulating a robotic space mission using the architecture defined in [da Silva 2024] with code available to promote open science (https://github.com/CJTS/city-planning).

The rest of the article includes: in Section 2, we show architectural aspects of MAS with AP; in Section 3, we display the experiments together with the used illustrative example; and lastly, in Section 4 the conclusions and future work.

2. Architectural Aspects

One aspect of MRS is the complex task-completing process which is viewed as decoupling complex tasks into simpler sub-tasks, forming a coalition of robots that will perform them, allocating to the coalitions, and using MAP to transform those tasks into sequential actions so the robots can perform [Rizk et al. 2019]. Figure 1 presents an adapted MRS workflow [Kiener and von Stryk 2010]. Note the workflow includes a human expert to decompose complex tasks into simpler sub-tasks based on the robots' capabilities available and the coalition formation of a set of agents. Then, the task (re-)allocation and robot planning and control steps are autonomously performed by the robot teams.

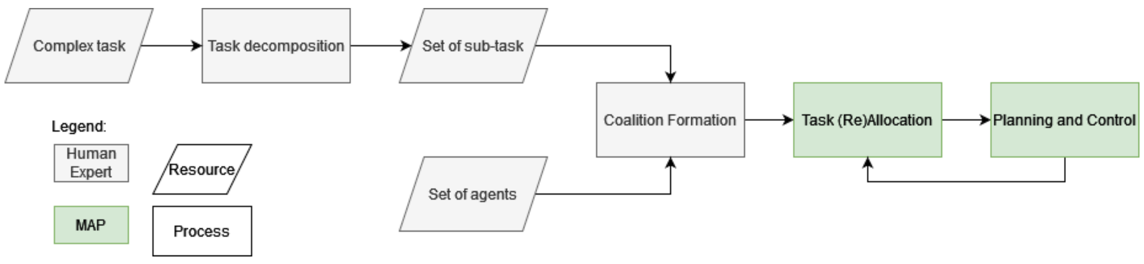


Figure 1. Adapted MRS workflow. Source: [Rizk et al. 2019].

The architecture used in this work is presented in [da Silva 2024], which integrates AP in MAS. The architecture's requirements involve the mission's decomposition into local plans to be executed by the agents. The *Coordinator* is responsible for receiving the mission request that contains the plans assigned to robot roles. Then, it coordinates the execution by assigning agents that fit the roles. It focuses on recovering the mission's plan when problems happen.

2.1. Design

As illustrated in Figure 2, the design phase necessitates a domain expert to outline the mission requirements. The *System Integrator* task is to develop the *Problem Domain* application component in planner syntax. This element is crucial to the architecture, as the effectiveness of the planning capabilities hinges on the formal definition of the problem and the domain. Goal reasoning functions may be incorporated later to address issues stemming from inadequate descriptions in this component.

The *Coordinator* and *Robot* runtime components exchange data on local plans and mission properties via messages. Establishing a communication protocol between these components is crucial.

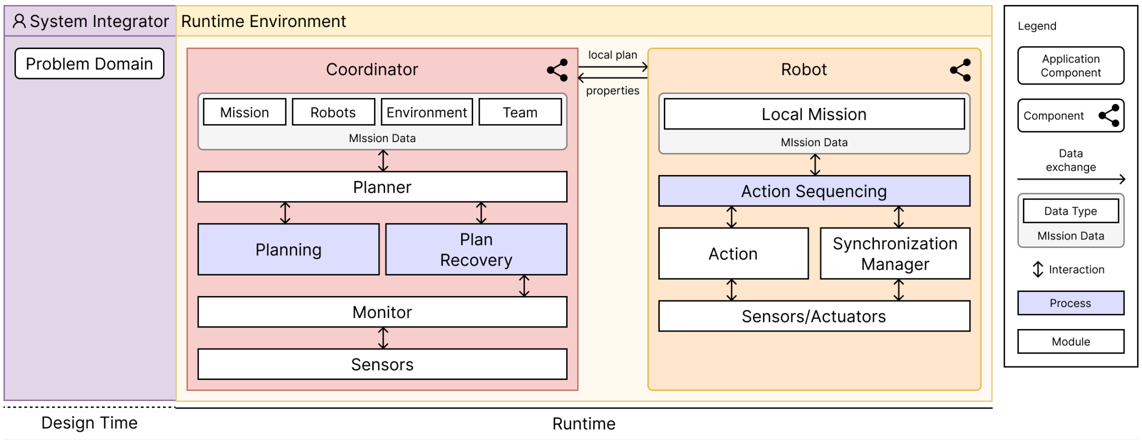


Figure 2. The high-level architecture. Source: [da Silva 2024].

The *Coordinator* oversees mission control, encompassing planning and plan recovery. This component stores the *Mission Data* required for mission execution, which includes the mission details, available robots, and the environment state. The *Coordinator Planner* module generates a mission plan through the *Planning* process using the automated planner, environment state, and problem domain. Subsequently, a planning execution cycle commences, where the *Coordinator* monitors the environment for unexpected changes via the *Monitor* and *Sensors* modules and receives feedback from the robots.

Once the *Local Mission* plan is established, the robots receive the plan from the *Coordinator* and initiate execution by carrying out actions sequentially through their *Action Sequencing* process. The *Robot Action* module executes each action using their *Sensors/Actuators* devices. Additionally, the *Synchronization Manager* module ensures coordination among actions performed by multiple robots. It's important to note that the plan is dynamically defined at runtime by the *Coordinator* component. In the event of a problem, the *Coordinator* reevaluates the original plan for redistribution to the robots, aiming to minimize disruptions in the MRS.

2.2. Execution Process

Figure 3 presents the execution workflow of the architecture. The initial step of the execution process is the initial trigger, determined by the system integrator and varies depending on the domain. Upon receiving the initial trigger, the *Coordinator* adjusts the initial state, defines the mission, and initiates preparations for creating the plan to accomplish it. Consequently, the *Coordinator* must comprehend, based on the problem domain, the required robot types to assemble the team.

Utilizing the information within the *Mission Data*, the *Planner* module initiates the *Planning* process to generate the optimal plan. In this work, we employed the HyperTensioN planner for this purpose [Magnaguagno et al. 2022]. However, alternative planners can also be considered, with careful evaluation of their capabilities beforehand. For a comprehensive comparison between planners, see [Georgievski and Aiello 2015].

In the sequence, the *Coordinator* divides the plan among the robots in the team and dispatches their respective local missions. Each robot initiates its task sequencing process to execute the plan. Concurrently, the *Coordinator* monitors the environment for any changes that could jeopardize the plan's feasibility. If an issue occurs, the *Coordinator* initiates the plan recovery process, which involves rectifying the current state of the environment and subsequently generating a new plan (replan) for the mission. The plan is successfully executed by the team of robots if no unexpected events arise.

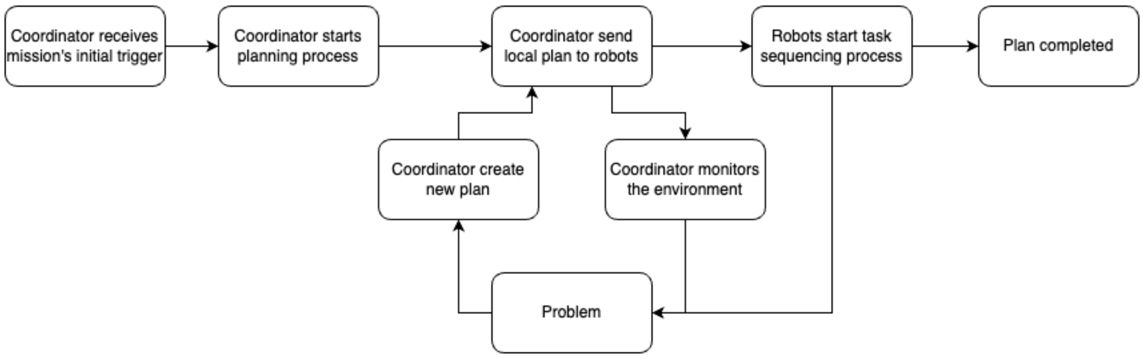

Coordinator receives mission's initial trigger
Coordinator starts planning process
Coordinator send local plan to robots
Robots start task sequencing process
Plan completed
Coordinator create new plan
Coordinator monitors the environment
Problem

Figure 3. The solution's execution process. Source: [da Silva 2024]

Plan Recovery

One of the reasons the architecture might need to execute a plan recovery process is when all actions yield the desired outcome, but, an unexpected event occurs in a detectable manner. An example of such an event is the robot's battery depletion. Since most planners do not incorporate numeric values in their domain modeling, typically, a boolean flag indicates the battery level (e.g., low). Most actions necessitate this flag to indicate sufficient battery power. During execution, if the flag indicates a low battery level, the system must replan to include a task for recharging the robot. This scenario is not necessarily an error but rather an event that could occur at any given time.

Another scenario requiring plan recovery arises when there's an issue with the domain modeling. For instance, if a car is supposed to wait for a crane to finish loading all boxes before unloading, but the modeling precondition erroneously allows unloading if no boxes are present, the car may prematurely vacate the docking area. Consequently, the initial plan, which only considered one car navigation, must be replanned to ensure the car returns for the remaining boxes. This situation underscores the necessity of plan recovery to rectify discrepancies in the domain modeling and ensure plan adherence.

The second scenario necessitating plan recovery arises from external events or agents interacting with the environment in a manner that unpredictably alters its state. For instance, consider a scenario where a door needs to remain open for a robot to pass through, but someone unexpectedly closes it. This unforeseen event disrupts the execution of the plan, prompting the need for plan recovery to address the changed environmental condition and revise the plan accordingly.

Regardless of the reason for replanning, the initial trigger is when the robot interacts with the environment. This interaction marks the commencement of the replan cycle, which can be visualized as a reactive replan process, as depicted in Figure 4 using a UML sequence diagram [Rumbaugh et al. 2004], [Object Management Group (OMG) 2015]. The reactive replan process transpires when the *Coordinator* receives the status of the environment after the robot attempts to act, and a problem emerges.

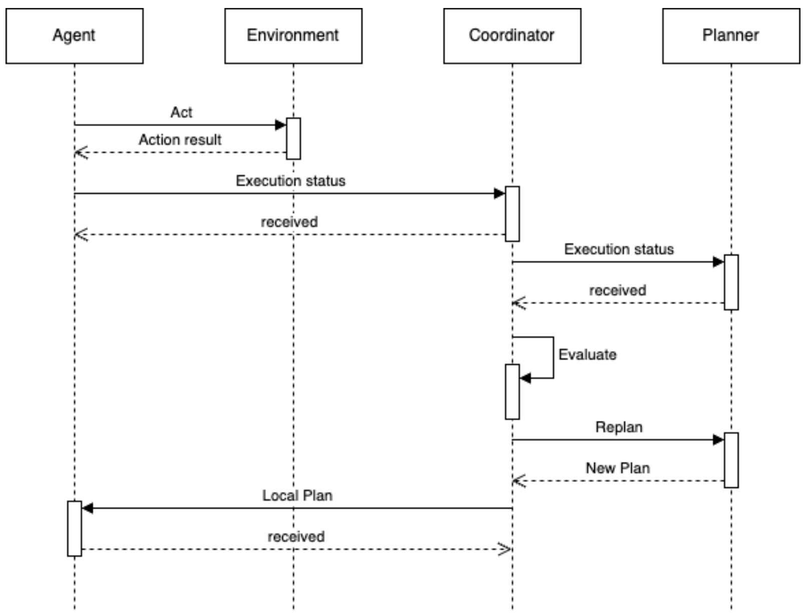


Figure 4. The architecture reactive replan sequence diagram.

When the environment status confirms the action is successful, the plan's execution can proceed uninterrupted. However, if the robot encounters difficulty completing the action, the *Coordinator* must ascertain the reason for the failure. The robot communicates the problem through an execution status message. Subsequently, the *Coordinator* updates the environment state model and requests a new plan from the planner to address the encountered issue. This iterative process enables the architecture to respond dynamically to environment changes and ensure plan adherence.

Dealing with false action results can indeed pose significant challenges. These issues arise when sensors erroneously indicate that an action was successful when it was not, or conversely, signal a problem when there isn't one. Resolving false action results typically involves two approaches, each with its complexities.

The first approach entails the *Coordinator* validating all actions to ensure their accuracy. However, this method can incur additional hardware costs, such as employing sensors to verify all action effects. Alternatively, the second approach needs a sophisticated heuristic, such as a backtracking algorithm that stores execution traces. This method enables the *Coordinator* to backtrack to the point where the problem occurred when the action's effects are needed.

At the current stage, the proposed architecture assumes that agents cooperate with veracity behavior when exchanging messages. Thus, addressing false action results was left as an avenue for future exploration. Such exploration underscores the need for further research to develop robust mechanisms for identifying and mitigating false action results within the architecture.

Figure 5 depicts a proactive replan process, where the *Coordinator* actively monitors the environment, preemptively identifying potential problems. In this process, the *Coordinator* maps the environment states, evaluates them to detect issues, updates the planner model as necessary, requests a new plan, and distributes the updated local plans to the agents for execution.

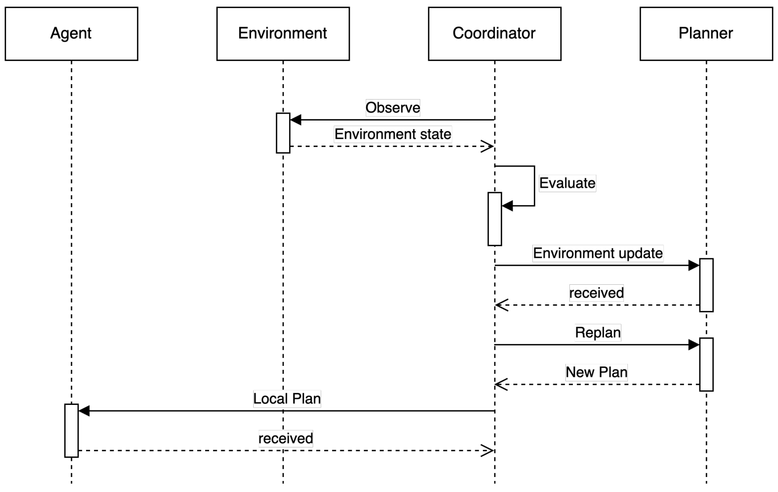


Figure 5. The architecture proactive replan sequence diagram.

This work extends beyond traditional plan recovery by addressing planning in situations where unexpected events occur with reactive (Figure 4) or proactive (Figure 5) replan, leading to changes in the environmental state. By encompassing both aspects, this architecture offers a comprehensive approach to handling dynamic and unpredictable scenarios within MRS.

3. Experiments

The main goal of the experiments is to describe and validate the strategy of the architecture's plan recovery process [da Silva 2024]. Experiments using MRS and a healthcare case already were made [da Silva and Ralha 2023]. However, the architecture focuses on working with multiple domains. Thus, the necessity of testing with other technologies and domains is vital. This work uses the Space Resource Gathering (SRG) on planet exploration illustration example.

3.1. Space Resource Gathering

Motion and path planning is already a common area of study in space robotics, as seen in works of [Basmadji et al. 2020], [Sun et al. 2023]. However, planet exploration is also a possible focus, and with the advancement of technology, we demand more research. On that note, to expand the research of planet exploration, we describe this illustrative example of *SRG* where we idealize that the cost of sending robots to the planets is lower and the possibility of sending a heterogeneous group of robots to perform various tasks is a possibility.

The *SRG* includes three robot types. One robot can map the environment looking for resources (*Scout*), another can collect the found resources (*Gatherer*), and the last has the skill of removing obstacles (*Remover*) that can appear during the plan execution. The mission has two types of main plans, one of mapping the environment and the other of collecting the resources. In the last part, should any obstacles arise, the plan will fail, and the *Coordinator* needs to replan accordingly.

Figure 6 shows a simple example of a map to illustrate how the simulation would work. The map has four parts: Base, Location 1, Location 2, and Location 3. In the figure, while the *Gatherer* is collecting resources in Location 1, the *Remover* is removing obstacles in Location 2, and the *Scout* searches for resources in Location 3.

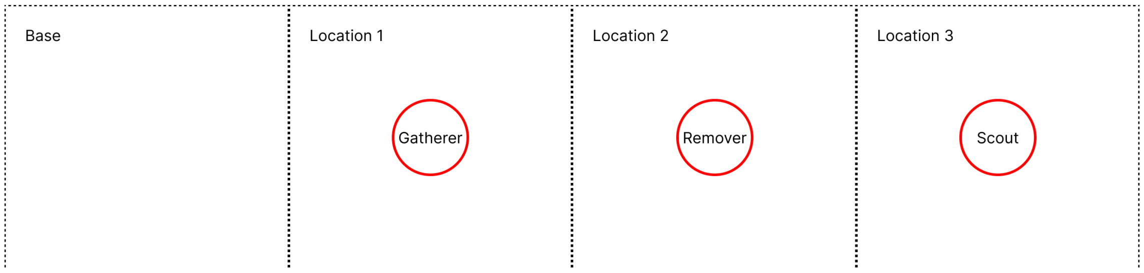


Figure 6. Example of a simulation map.

Listing 1 shows an example of a plan generated by the HyperTensioN where no replan is needed. The *Scout* searches for a resource, and then the *Gatherer* collects it. In Listing 2, a new part of the plan is needed after the *Gatherer* tries to collect the resource, but an external event happens, and an obstacle is in the way, so the *Remover* need to remove it before the *Gatherer* can finish its local plan.

Listing 1. Example of a plan without replan.

```
move_scout(scout base location)
map(scout resource enemy location)
move_scout(scout location base)
move(gatherer base location)
pickup(gatherer resource)
move(gatherer location base)
drop(gatherer resource)
```

Listing 2. Example of a plan with replan.

```
move_scout(scout base location)
map(scout resource enemy location)
move_scout(scout location base)
move_remover(remover base location)
```

```
remove_obstacle(remover obstacle location)
move_remover(remover location base)
move(gatherer base location)
pickup(gatherer resource)
move(gatherer location base)
drop(gatherer resource)
```

3.2. Experimental Setup

The experiments use Ruby (v2.6.10) [Matsumoto 2022] as the principal language. A process creates a thread for each agent (*Coordinator* with planner and robots). The communication between the agents uses a web socket with the TCP protocol. The messages exchange uses a JSON format with two attributes, "action" with a string representing the message purpose and a "value" with the necessary data for the agent act. Figure 7 shows the execution process of the experiment.

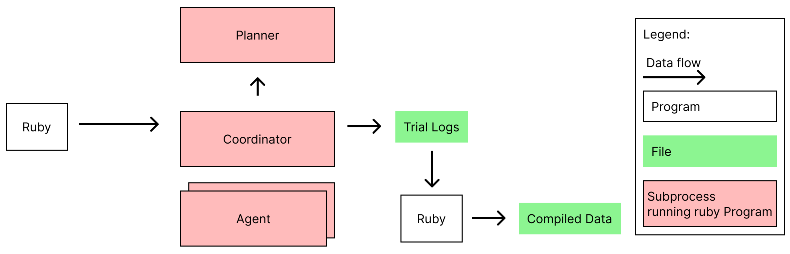


Figure 7. Experimental process.

The experiment execution took into account the following aspects:

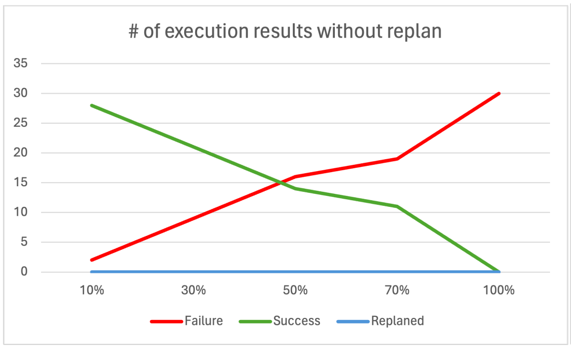
- objective - to evaluate how effectively the architecture mitigates issues when operating in a simulated dynamic environment.
- problem types - Objects preventing resource collection.
- case study - the SRG using five experimental scenarios related to the evaluated problem.
- validation strategy - we conducted five scenarios, each repeated 30 times for statistical significance, to assess the *Coordinators*' ability to complete the plan. Plan results were analyzed at the end of each execution, focusing on the percentage of uncompleted missions related to the evaluated problem. The scenarios featured a door-closed event with probabilities of 10%, 30%, 50%, 70%, and 100%.
- results evaluation metric - plan completion and time required to complete the plan. We compared the experimental results with a baseline case where the *Coordinator* cannot replan.

3.3. Results

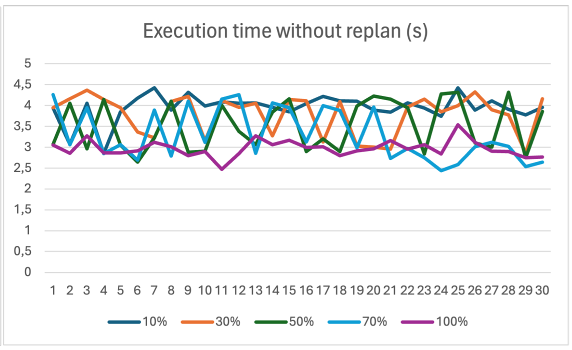
In this section, we analyze the experimental results to assess the effectiveness of the proposed architecture in dynamic environments, specifically focusing on the plan recovery process illustrated with space resource-gathering robots.

3.3.1. Baseline Analysis

The baseline scenario serves as a control to compare the effectiveness of the plan recovery mechanism. In the baseline, the architecture cannot replan in response to unexpected events. The results are depicted in Figures 8a and 8b.



(a) # of execution results without replan.



(b) Execution time without replan (s).

Figure 8. Charts of mission's results without replan.

Figure 8a shows the number of mission completions without replan capability across various probabilities of obstacles (10%, 30%, 50%, 70%, and 100%). As expected, the number of failures increases proportionally with the obstacle occurrence probability. This result demonstrates the limitation of static planning in dynamic environments, where an increased likelihood of external disruptions significantly impacts mission success rates.

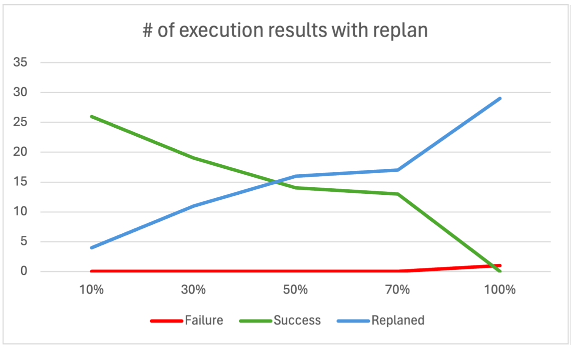
Figure 8b presents the execution time for missions without replanning. Despite variations in the probability of obstacles, the execution times remain relatively consistent. This result relates to the simplified plan nature used in the baseline scenario, where missions either succeed without disruption or fail early due to obstacles.

3.3.2. Replan Capability Analysis

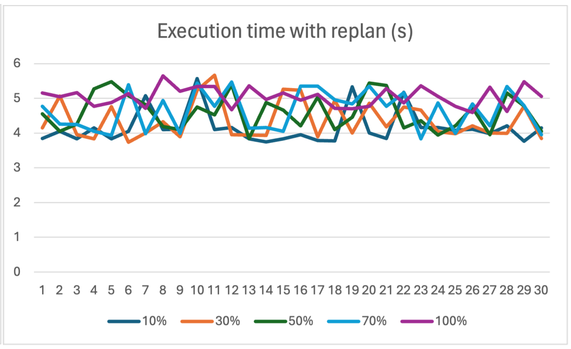
The replan scenarios incorporate the architecture's ability to recover plans dynamically in response to obstacles. Figure 9a and Figure 9b present the results.

Figure 9a depicts the number of mission completions with replan capability across varying probabilities of obstacles considering success when no problem occurs. Unlike the baseline, the architecture's ability to replan significantly improves mission success rates, even as the probability of obstacles increases. This result demonstrates the effectiveness of the plan recovery mechanism in mitigating the impact of dynamic environmental changes.

Figure 9b shows the execution time for missions with replanning. Similar to the baseline, the execution times are consistent across different probabilities of obstacles. However, note that the architecture's replan capability adds a slight overhead to the execution time due to the replanning process. Despite this, the improvement in mission success rates outweighs the marginal increase in execution time.



(a) # of execution results with replan.



(b) Execution time with replan (s).

Figure 9. Charts of mission's results with replan.

3.3.3. Comparative Analysis

The advantages of the replan capability are evident in Figures 8 and 9. The architecture's ability to dynamically recover plans allows it to maintain high mission success rates, even with frequent obstacles. The slight increase in execution time is a reasonable trade-off for the enhanced robustness and reliability of the replan mechanism.

Overall, the results validate the effectiveness of the proposed architecture in handling dynamic environments through its replan capability. The ability to dynamically adjust plans in response to unexpected events significantly enhances mission success rates, demonstrating the potential of this approach for real-world applications in multi-robot systems operating in unpredictable conditions.

4. Conclusion

The experiments validate the proposed MAS architecture in [da Silva 2024] integrating AP for multi-agent coordination in dynamic environments. The *SRG* example illustrates the architecture's capabilities in a simulated space robotics scenario, managing task decomposition, coalition formation, and task allocation adapted to dynamic changes, ensuring mission continuity. The central *Coordinator* oversees planning and execution, presenting a reliable strategy for managing heterogeneous robotic teams in the studied scenario. Comparing the scenarios, it is evident that this functionality enhances the system's ability to complete missions successfully, even when the probability of encountering obstacles increases.

Future work focuses on enhancing the architecture's capabilities with false action results using sophisticated heuristic algorithms. Further, performing experiments in other domains to validate the architecture's versatility, experiments with increasing numbers of robots, integrating with more advanced planning algorithms to enhance communication protocols, and finding a suitable benchmark for multi-agent planning and comparing with some existing work will be salient to future research.

References

Aziz, H., Chan, H., Cseh, A., Li, B., Ramezani, F., and Wang, C. (2021). Multi-robot task allocation-complexity and approximation. In *Proc. of 20th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 133–141.

Basmadji, F., Seweryn, K., and Sasiadek, J. (2020). Space robot motion planning in the presence of nonconserved linear & angular momenta. *Multibody System Dynamics*, 50.

Bischoff, E., Teufel, J., Inga, J., and Hohmann, S. (2021). Towards interactive coordination of heterogeneous robotic teams – introduction of a reoptimization framework. In *Proc. of IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*, pages 1380–1386.

Carreno, Y., Ng, J. H. A., Petillot, Y., and Petrick, R. (2022). Planning, execution, and adaptation for multi-robot systems using probabilistic and temporal planning. In *Proc. of 21st Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 217–225.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carreraa, A., Palomeras, N., Hurtós, N., and Carrerasa, M. (2015). ROSPlan: Planning in the robot operating system. In *Proc. of 35th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, page 333–341.

da Silva, C. J. T. (2024). A multi-robot system architecture with multi-agent planning. Computer Science Department, University of Brasilia, Campus Darcy Ribeiro - Asa Norte, Brasília - DF, 70910-900, Brazil.

da Silva, C. J. T. and Ralha, C. G. (2023). Multi-robot system architecture focusing on plan recovery for dynamic environments. In *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1668–1673.

et al., M. (2021). RoboMAX: Robotic mission adaptation exemplars. In *Proc. of Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 245–251.

Georgievski, I. and Aiello, M. (2015). Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156.

González, J. C., García-Olaya, A., and Fernández, F. (2020). Multi-layered multi-robot control architecture for the robocup logistics league. In *Proc. of IEEE Int. Conf. on Autonomous Robot Systems and Competitions*, pages 120–125.

Kiener, J. and von Stryk, O. (2010). Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots. *Robotics and Autonomous Systems*, 58(7):921–929. Advances in Autonomous Robots for Service and Entertainment.

Klavins, E. (2004). *Communication Complexity of Multi-robot Systems*, pages 275–291. Springer, Berlin, Heidelberg.

Komenda, A., Stolba, M., and Kovacs, D. L. (2016). The international competition of distributed and multiagent planners (CoDMAP). *AI Magazine*, 37(3):109–115.

Lesire, C., Bailon-Ruiz, R., Barbier, M., and Grand, C. (2022). A hierarchical deliberative architecture framework based on goal decomposition. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 9865–9870.

Magnaguagno, M. C., Meneguzzi, F., and De Silva, L. (2022). HyperTensioN: A three-stage compiler for planning. In *Proc. of 30th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 1–4.

Martín, F., Clavero, J. G., Matellán, V., and Rodríguez, F. J. (2021). PlanSys2: A planning system framework for ROS2. In *Proc. of IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, page 9742–9749.

Matsumoto, Y. 2022). Ruby. `https://www.ruby-lang.org/en/`. Accessed: 2024-06-03.

Moreira, L. H. and Ralha, C. G. (2021a). Evaluation of decision-making strategies for robots in intralogistics problems using multi-agent planning. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1272–1279.

Moreira, L. H. and Ralha, C. G. (2021b). Plan recovery process in multi-agent dynamic environments. In Gusikhin, O., Nijmeijer, H., and Madani, K., editors, *Proc. of 18th Int. Conf. on Informatics in Control, Automation and Robotics (ICINCO)*, pages 187–194.

Moreira, L. H. and Ralha, C. G. (2022a). An efficient lightweight coordination model to multi-agent planning. *Knowledge and Information Systems*, 64:415–439.

Moreira, L. H. and Ralha, C. G. (2022b). Method for evaluating plan recovery strategies in dynamic multi-agent environments. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–25.

Object Management Group (OMG) (2015). Meta-Object Facility (MOF) Specification, version 2.5. OMG Document Number formal/2015-03-01.

Rizk, Y., Awad, M., and Tunstel, E. W. (2019). Cooperative heterogeneous multi-robot systems: A survey. *ACM Comput. Surv.*, 52(2).

Rodrigues, G., Caldas, R., Araujo, G., de Moraes, V., Rodrigues, G., and Pelliccione, P. (2022). An architecture for mission coordination of heterogeneous robots. *Journal of Systems and Software*, 191(111363).

Rumbaugh, J., Jacobson, I., and Booch, G. (2004). *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education.

Salzman, O. and Stern, R. (2020). Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proc. of 19th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 1711–1715.

Schmitt, P. S., Wirnshofer, F., Wurm, K. M., Wichert, G. v., and Burgard, W. (2019). Modeling and planning manipulation in dynamic environments. In *Proc. of Int. Conf. on Robotics and Automation (ICRA)*, pages 176–182.

Silva, L. d., Meneguzzi, F., and Logan, B. (2020). BDI Agent Architectures: A Survey. In *Proc. of 29th Int. Joint Conf. on Artificial Intelligence, (IJCAI)*, pages 4914–4921.

Sun, Y., Wu, J., and Liu, T. (2023). Joint task allocation and path planning for space robot. *IEEE Access*, 11:42314–42323.

Verma, J. K. and Ranga, V. (2021). Multi-robot coordination analysis, taxonomy, challenges and future scope. *Journal of Intelligent & Robotic Systems*, 102(1).

Weiss, G. (2016). *Multiagent Systems*. The MIT Press, 2nd edition.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.