**Framework Application**

In Chapter ?, two decoupled multi-agent problem algorithms were introduced and their inner workings explained. In Chapter ??, a flexible and general framework for the implementation of algorithms was presented. This chapter combines the concepts of the preceding chapters, presenting a guide process through which two different algorithms are adapted to function on equivalent playing fields through the use of the state machine approach.

To begin, the algorithms are decomposed into their underlying behaviors such that they cohere with the state machine diagrams presented in Chapter ?. With increasing granularity, routines present in the algorithms are assigned to states in the governing state machine until the algorithm is fully represented. From this position it becomes trivial to extract a strategy for practical implementation, as is done in the design of FleetBench. The use of this process, shown partly in this chapter and partly in the Appendix, demonstrates the advantages of approaching MAPF and MAPD solution implementation in this manner.

This chapter assumes that the Simulation Definition state is already completed by a user configuring certain options. The actual implementation of the definition state experiences a great degree of freedom in terms of what options are available in the simulation and must be developed as needed. As an example, the implementation used in FleetBench at time of writing accepts the following configuration choices:

**Table of options available in definition phase, separated by type**

**WHCA\***

To begin fitting the decoupled MAPF algorithm WHCA\* into the state machine model it is worth noting that WHCA\* offers no particular strategy for task assignment, as the MAPF problem assumes that all agents have a task assigned before the problem should be solved. As a result, any implementation of the WHCA\* algorithm in an MAPD context will need a generic strategy for task assignment. Here, as in the implementation given in Appendix ?, the generic routine is named GenerateTask and simply selects the first available task from the task set.

Critically, WHCA\* is also incomplete in the MAPD case. It fails to consistently avoid collisions during its runtime. This problem arises when agents finish their current plans, and thus reservations, while another agent is attempting to reach the same goal location. If the first agent to arrive finds itself trapped, it will be unable to move away while simultaneously not being able to remain in place. In order to avoid an immediate end of the simulation via the crashed state, it is necessary to develop a generic collision resolver. Even in the MAPF case, a one-agent width corridor of sufficient depth (exceeding the window size) will prevent progress from being made as neither agent will find an escape from its current position, resulting in an infinite stall. FleetBench approaches this problem using a collision resolver which prioritizes trapped agents and forces a replanning of agent motions until the problem is resolved.

The following algorithm comparison shows the translated version of WHCA\*, where lines of the original algorithm have been replaced with named routines for ease of reference.

**Two equivalent expressions of the WHCA\* algorithm typeset here**

Of course, the bulk of the logic employed in WHCA\* is for pathfinding, making it relatively easy to fit into the state machine model. For completeness, the algorithm is expanded to include the routines used in FleetBench for task selection and collision resolution, explained in Appendix ?. Algorithm ? shows the manner in which each line of WHCA\* is rewritten into named routines.

**Algorithm table showing the actions taken, and their named routines**

The state machine representation of WHCA\* is therefore given by figure ?:

**Figure showing the state machine version of the algorithm, with routine names**

**TPTS**

TPTS, being designed for the MAPD problem which better analogizes real-world applications in continuously active cooperative robotics applications, is more involved. The algorithm has a strategy for both task selection and task exchanging, offering an improvement in efficiency by minimizing unnecessary travel time. It also offers conditional guarantees regarding completeness of the algorithm. If the system map conforms to the definitions presented in [1], then it should be impossible for a collision to occur. Verifying whether those conditions are met is a task for the Simulation Definition state. In the case that a system map is not well-formed a conflict resolution system must be in place to prevent the simulation from entering the crashed state.

Once again, a line-by-line process of re-composing the algorithm into named routines is presented here, both in algorithmic form in Algorithm ? and in the resulting state machine form, shown in Figure ?.

**Algorithm table show the actions and their named routine versions**

**Figure showing the state machine version of the algorithm, with routine names**

[1] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, “Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks.” arXiv, May 30, 2017. Accessed: Nov. 05, 2023. [Online]. Available: http://arxiv.org/abs/1705.10868