A Distributed Monte-Carlo Simulation Toolbox for MOOS-IvP

Kevin Becker[1] Michael Benjamin[1]

***Index Terms*—Monte-Carlo, distributed simulations, marine robotics, MOOS-IvP, autonomy certification, verification, autonomy testing, behavior-based autonomy, multi-objective optimization**

I. Introduction

Multi-objective optimization generally is a method for reconciling competing utility functions over a common decision space. It has been applied as a method of action selection for choosing decisions in behavior-based robotic systems [1] when multiple competing goals are simultaneously vying for influence in the direction, speed, depth or altitude of a vehicle, [2]–[5]. Such systems involve many possible settings in both (a) the relative priority weights of objective functions and (b) the configuration parameters of the behaviors producing the objective functions. Furthermore, the state of the physical world, driving the behavior output also has endless variations. This is even more pronounced when a robot is working in the midst of several collaborating, neutral, or adversarial robots. Even with massive simulation resources, it would be impossible to exhaustively test all permutations.

However, thoughtfully constructed sets of simulations can sample the space of possibilities to search for edge-case anomalies. Automated simulations and analysis can also automate the direction of follow-on permutations to simulate. Monte-Carlo simulation with full pipeline automation, can have an enormous impact both on algorithm development as well as building confidence through verification of capabilities. A mission pipeline includes both the pre- and post-mission stages surrounding the autonomous mission itself. Monte-MOOS was developed to adhere to standards and conventions used by MOOS-IvP autonomy and its supporting utilities. The approach presented here can be generalized to other autonomy systems.

II. The need for Automated Simulations in MOOS-IvP

A. Infeasibility of Exhaustive Real World Testing

1) Certification of Multi-Objective Optimization: The number of parameters required for multi-objective optimization results in a high-dimensional search problem, where a user is attempting to maximize some performance metric while meeting certain criteria.

[1] Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA kevin00@mit.edu, mikerb@mit.edu

A collision avoidance scenario can have many parameters and test scenarios, as shown in Figure 1. The number of required tests increases exponentially with the number of configuration settings, demonstrating the need for automated testing in simulation.

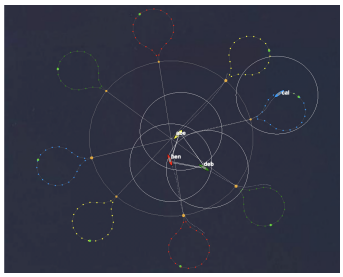

Fig. 1: The *jousting* mission is an experiment where autonomous vehicles are driven through the middle of a circle to test collision avoidance algorithms. This mission has many parameter variations. These include vehicle count, behavior configurations, and starting conditions. It is a common test for verifying collision avoidance behaviors which can be tested using Monte-MOOS.

III. Monte-MOOS Features

A. External Code Management

This toolbox was designed to work with the marine-robotics focused middleware MOOS-IvP [6]. This adaptable and expandable middleware comes with a template for expanding the codebase [7]. By leveraging this template, Monte-MOOS has the ability to download, compile, and run code from a verified user. The software also has the built-in functionality to remove any external codebases that have not been used in the past 30 days.

B. Distributed Simulations, Centralized Results

Monte-MOOS uses a host-client architecture, where the host determines what should be run and by which computers. The host also acts as a centralized database, storing how many times to run each job, instructions for how to run each job, and the results from each run. Distributed simulations allow for multiple computers to contribute to the results.

By allowing for distributed simulations, the client computers may be located anywhere in the world with internet connection. The distributed architecture also makes the cluster robust to the loss of a client due to maintenance or bugs as shown in Figure 2.

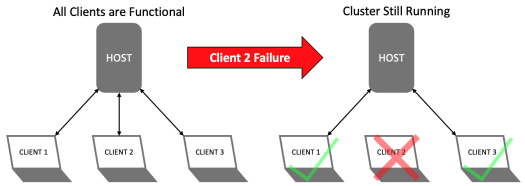

Fig. 2: The distributed architecture makes the simulation cluster robust to clients dropping.

C. Minimal Dependencies

Reducing the number of dependencies makes the code easier to install, which improves the usability. Therefore, the only dependencies for Monte-MOOS are bash, cmake, gcc, and MOOS-IvP. The programming language Python 3 and the libraries Matplotlib, Numpy, and Pandas were also included in the dependency list. This allows users to incorporate these well-known utilities in their post-simulation data processing without worrying if clients have the ability to run the post-processing scripts.

IV. Utilizing the Front-seat Back-seat Paradigm

A. About the Front-seat Back-seat Paradigm

There is a concept used in marine robotics research called the front-seat-backseat paradigm [8]. Marine robots often contain proprietary code from a manufacturer, which makes it difficult to interface with these vehicles or load custom software to a given platform. This paradigm is where the vehicle computer handles the control algorithm and sensors, while the autonomy algorithm is on a separate computer, commanding desired heading and speed values. This paradigm enables users to develop an autonomy stack on their own computer, called the backseat, and send instructions to the autonomous vehicle's computer, known as the front-seat computer.

B. Benefits of the Front-seat Back-seat Paradigm

This paradigm also enables multiple researchers to share the same set of robots more easily, since users can keep a back-seat computer with them and test code in simulation on those computers directly. It also prevents issues with operating systems, different versions of libraries, and more. Additionally, since these backseat computers are more affordable than a new robot, it is feasible to have extra backseat computers that can be swapped out in minutes if one breaks, or if another researcher wants to test their code on the same platform. It has also been successfully implemented in the MIT 2.680 course, where students are given their own backseat autonomy computers [8] as shown in Figure 3b.

C. Back-seat Computers as Clients

Testing a specific pairing of code and hardware makes a system more robust, since any memory issues or processor speed issues will be exposed. Monte-MOOS is lightweight enough where it can run on the Payload Autonomy Boxes, or back-seat computers. This helps further verify the code's reliability and robustness. These computers are abundant in research labs where the paradigm is used, such that adding them to the simulation cluster will notably increase the size of a given Monte-MOOS cluster.

(a) Three *Clearpath Robotics* Heron USVs

(b) Backseat computer

Fig. 3: The *Clearpath Robotics* Heron USVs are used for testing jousting variations that are found to be difficult through testing in Monte-MOOS. The backseat computer used by the MIT marine autonomy class (Payload Autonomy Box) consists of a Raspberry Pi 4 computer in a watertight box with connectors.

V. Conclusions & Future Work

Monte-MOOS is a toolbox for running distributed simulations in the MOOS middleware. It is important for certification of autonomy stacks with many variables, such as software which relies upon multi-objective optimization, particularly for expensive platforms. The toolbox leverages the front-seat back-seat paradigm by using these computers to run simulations.

Monte-MOOS is currently being used for marine collision regulations research. A large Monte-MOOS simulation cluster consisting of one hundred Payload Autonomy Boxes is preparing to be deployed.

Acknowledgment

We would like to thank Mikala Molina and Tyler Paine for helping stress-test the Monte-MOOS code and tutorials.

References

[1] R. Brooks, "A robust layered control system for a mobile robot," vol. 2, no. 1, pp. 14–23.
[2] J. Riekki, *Reactive Task Execution of a Mobile Robot*. PhD thesis, Oulu University, 1999.
[3] J. K. Rosenblatt, "Damn: A distributed architecture for mobile navigation," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2, pp. 339–360, 1997.
[4] P. Pirjanian, *Multiple Objective Action Selection and Behavior Fusion*. PhD thesis, Aalborg University, 1998.
[5] M. R. Benjamin, *Interval Programming: A Multi-Objective Optimization Model for Autonomous Vehicle Control*. PhD thesis, Brown University, Providence, RI, May 2002.
[6] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested autonomy for unmanned marine vehicles with moos-ivp," *Journal of Field Robotics*, vol. 27, no. 6, pp. 834–875, 2010.
[7] "moos-ivp/moos-ivp-extend." original-date: 2021-12-07.
[8] M. Benjamin, "MIT 2.680: Introduction to the PABLO Payload Autonomy Computer." https://oceanai.mit.edu/ivpman/pmwiki/pmwiki.php?n=Lab.ClassPabloIntro, 2024. Accessed: Jan. 01, 2024.