Simulating Collaborative Robots in a Massive Multi-Agent Game Environment (SCRIMMAGE)

Kevin DeMarco[1], Eric Squires[2], Michael Day[3], and Charles Pippin[4]

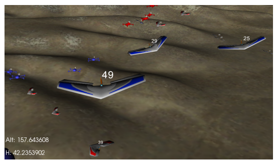
[1] Georgia Institute of Technology, Atlanta GA 30332, USA,
`kevin.demarco@gtri.gatech.edu`,
[2] Georgia Institute of Technology, Atlanta GA 30332, USA,
`eric.squires@gtri.gatech.edu`
[3] Georgia Institute of Technology, Atlanta GA 30332, USA,
`michael.day@gtri.gatech.edu`
[4] Georgia Institute of Technology, Atlanta GA 30332, USA,
`charles.pippin@gtri.gatech.edu`

Abstract. Testing mobile robotic systems in the field is a costly and risky task. Unfortunately, there is a gap between the existing simulation capabilities and those required to simulate large numbers of aerial vehicles. Many multi-agent robotics simulators have been restricted to the 2D plane, which limits their usefulness for aerial robotic platforms. While high-fidelity 3D robotics simulators exist, simulating large numbers of agents in these simulators can result in slower-than-real-time performance. SCRIMMAGE provides a 3D robotics environment that can simulate varying levels of collision detection, sensor modeling, communications modeling, and motion modeling fidelity due to its flexible plugin interface. This allows a robotics researcher to simulate hundreds of aircraft with low-fidelity motion models or tens of aircraft with high-fidelity motion models on single computer. SCRIMMAGE provides tools for batch simulation runs, varying initial conditions, and deployment to a cluster.

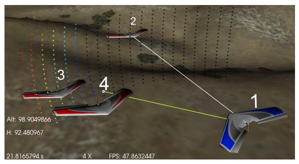
Keywords: simulation, swarm robotics, autonomy

1 INTRODUCTION

The high-cost of testing robotic systems in the field has led to the development of a number of simulators for autonomous systems. A well-designed simulator allows robotics researchers to debug and test their algorithms and data pipelines in a controlled environment before deploying to a physical system where the cost of failure is significantly higher. However, many robotics simulators have been developed primarily for ground-based systems, with a focus on high-fidelity multi-body physics models. This has made simulating large numbers (hundreds or thousands) of aerial vehicles infeasible due to the computational complexity of mesh collision detection and multi-body physics simulation. In many use cases for aerial robotics, precise collision detection and multi-body physics simulation is not required. Thus, assumptions can be made in the plugin design that decrease simulation time. To bridge this gap in simulation capabilities, the Simulating Collaborative Robots in Massive Multi-Agent Game Environment (SCRIMMAGE) was developed. SCRIMMAGE provides a flexible simulation environment for the experimentation and testing of novel mobile robotics algorithms. For example, a screenshot of a predator/prey simulation in SCRIMMAGE is shown in Fig. 1a [17]. The development of SCRIMMAGE was inspired by the Stage



(a) A screenshot of the SCRIMMAGE visualizer during a predator/prey simulation involving fixed-wing and quadrotor agents.



(b) A screenshot of a fixed-wing performing velocity obstacle-based collision avoidance [7].

Fig. 1: SCRIMMAGE screenshots.

robotics simulator [9]. Specifically, Stage's ability to efficiently simulate multiple robots and its plugin interface for robotic control, motion models, and autonomy communication. However, Stage is inherently a two-dimensional simulator, which limits its usefulness for aerial robotic platforms. SCRIMMAGE provides a three-dimensional robotics environment that can simulate varying levels of sensor, motion model, and network fidelity due to its flexible plugin interface. This allows a robotics researcher to simulate hundreds of aircraft with low-fidelity motion models or tens of aircraft with high-fidelity motion models on a standard consumer laptop. An example of using high-fidelity motion models from the JSBSim flight dynamics simulator to test a collision avoidance maneuver is shown in Fig. 1b. This example demonstrates some of the visualization tools that are available through SCRIMMAGE. Also, the fact that SCRIMMAGE's controller deterministically executes all plugins in a lockstep-like fashion means that the cost cloud can be visualized at each simulation time-step and analyzed meticulously by the autonomy developer.

If a large number of simulation runs is desired, SCRIMMAGE provides tools for cluster deployment or batch runs on a local machine. Because the space of pertinent input variables can grow exponentially for a given simulation, SCRIMMAGE allows users to easily use Latin Hypercube Sampling to create a set of mission files based on desired input ranges [8, 15]. While SCRIMMAGE has built-in logging capability for visualization, state information, and entity interactions, it also provides the capability to add custom data capture through a novel metrics plugin interface. A metric plugin can subscribe to a topic of interest, log the data, and calculate a score both for individuals and teams. SCRIMMAGE will then save log information to a summary csv file and calculate an overall score by aggregating the results of individual metrics plugins. After the simulation runs are completed, SCRIMMAGE provides tools that can aggregate the results from the simulation runs and provide statistical results. If a robotics developer requires advanced statistical processing, the developer can leverage SCRIMMAGE's Python bindings, which allow the developer to directly parse the trajectories from simulation runs.

2 RELATED WORK

The authors required a simulator that provided (1) scalability to a large number of robots for swarm applications, (2) support for the aerial domain, (3) a minimal-effort path for simulator code to arrive on a physical system, and (4) an open source license. Several simulators were considered for this task, and SCRIMMAGE leverages lessons learned from the advantages and disadvantages presented in the following existing simulation environments.

The Player/Stage Project provides an architecture to control and simulate multiple ground-based robotic systems. However, its lack of 3D motion models makes it unusable for aerial robotics research. While Stage version 3 has improved the degree to which swarm robotics can be scaled [25], it does not natively support aerial robot flight dynamics models because it focuses on ground robots. Similar to Stage in fidelity of simulation is the NetLogo simulator [22]. While NetLogo can be used to simulate 100's of interacting agents, NetLogo models are not written in C++, but NetLogo's own programming language. This could make deployment to physical robots cumbersome.

A simulator related to Stage, Gazebo, provides dynamic 3D multi-robotic worlds that can be rich in complexity [13]. Gazebo's development has been tightly integrated with the Robot Operating System (ROS), which has led to its widespread acceptance by the robotics community. Gazebo is capable of working with SolidWorks to provide closed kinematic chains to highly complex CAD models [2]. However, the Gazebo simulator does not scale well when simulating hundreds of robotic systems on a single machine due to the high-cost of precise collision detection. Gazebo depends on a collision detection package such as the Bullet Collision Detection library or the Open Dynamics Engine (ODE) in order to operate. A major advantage of SCRIMMAGE is that it can leverage a full collision detection library or it can utilize lower-fidelity collision detection methods in order to meet the researcher's needs. Similar to Gazebo in fidelity of simulation is the V-REP robot simulator [21]. V-REP provides an open source license for academic use, but a non open source license for companies, research institutes, and non-profit organizations. The authors of this paper cannot use V-REP's open source version since they work at a research institute.

A need for a large number of aerial robots acting together in a virtual environment led SCRIMMAGE developers to look to multi-agent based simulators. Simulators such as MASON, FLAME, and Repast focus on agents interacting together with minimal physics modeling or networking in order to simulate large numbers of agents [14, 12, 6]. MASON, which is written in Java, seeks to be lightweight and scalable on many architectures and provides some limited 3D visualization. FLAME and Repast are highly scalable in that they promote parallelizable models that can be run in high-performance computing environments. All of these simulators; however, are general purpose for many different domains and not specific to robotics.

A relatively new 3D multi-robot simulator, ARGoS, was designed to simulate large heterogeneous swarms of robots [18]. ARGoS' novel feature is its ability to allocate different physics engines for different spatial areas of the simulation environment. While ARGoS has a mini-quadrotor robot model, the examples provided are mostly of indoor ground-based systems that perform manipulation tasks. The motivation for developing SCRIMMAGE was to simulate large numbers of outdoor autonomous aerial systems.

SCRIMMAGE seeks to bridge the gap between a system such as Gazebo that provides a rich set of high-fidelity models ready-made for a robotics context and a multi-agent system that can model large numbers of aerial robots in a single world. Also, SCRIMMAGE provides a way to substitute new models of varying fidelity if users need to simulate a system that has novel needs. Furthermore, SCRIMMAGE provides interfaces to other systems that are commonly used in the robotics community, such as the Robot Operating System (ROS), the Mission Oriented Operating Suite (MOOS), and OpenAI Gym [19, 16, 5]. The SCRIMMAGE ROS Autonomy plugin matches the ROS topics and data types that are used by the Stage simulator.

3 DESIGN GUIDELINES

SCRIMMAGE was developed following design guidelines that closely follow those of modern open source software development as well as some novel guidelines for robotics simulators. The first guideline is that SCRIMMAGE simulations must be deterministic. Each SCRIMMAGE world is defined in an XML file, which is now referred to as a SCRIMMAGE mission file. This mission file contains robot descriptions, terrain data, obstacle locations, physics engines, etc. The mission file also contains a `seed` tag that is used to seed SCRIMMAGE's pseudorandom number generators. SCRIMMAGE uses these pseudorandom number generators to add noise to sensors, add noise to motion models, and randomly place entities based on distributions. Plugins can use these same pseudorandom number generators for decision, control, and noise generation as well. Since all software components within SCRIMMAGE generate pseudorandom numbers using the same initial seed value, given the same mission file, the SCRIMMAGE simulation will be deterministic.

SCRIMMAGE provides the ability to run a simulation in lockstep to facilitate plugin development. ROS provides a useful framework for passing messages between software components for robotic applications. However, one of the difficulties with developing with ROS is the inability to "pause" or run a debugger in a ROS node because other asynchronous nodes in the system will continue to process and publish data even though the node that the developer is debugging has paused. Thus, a second design guideline for SCRIMMAGE was to ensure that the entire simulation could be paused and stepped in a lockstep fashion. This allows a plugin developer to run SCRIMMAGE in a debugger, such as gdb, or print debug information to the terminal without affecting the simulation results. Part of the lockstep implementation includes not using any form of TCP/UDP network messaging in SCRIMMAGE or its plugins. However, SCRIMMAGE does provide a simple shared-memory publish/subscribe system that does not use the system's networking layer to pass messages between plugins. Additionally, SCRIMMAGE can provide the simulation clock to ROS, which allows it to "step" ROS nodes.

A third design guideline for SCRIMMAGE is that motion model plugin developers should be allowed to describe their robot motion models using explicit state transition equations of the form, $\dot{x} = f(x, u)$. This is different from how motion models are described in multi-body physics simulators like ODE in Gazebo, where a model is used to describe the transformations between the robot's base link, wheels, sensors, etc. While the transformation-based model description is useful for robot manipulation tasks, it is not the canonical form used by control engineers and mobile robot planners. This can lead to inconsistencies between a planner's expected model of the state transition equations and the actual motion model's state transition equations. This is not an issue when using planners in SCRIMMAGE because the motion model used during simulation can be directly used during planning, as shown in Fig. 2. In this case, an autonomy plugin cre-

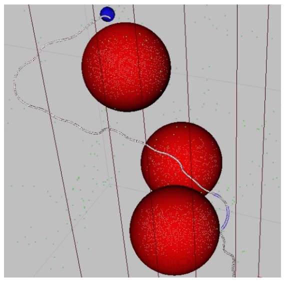


Fig. 2: The resulting trajectory of a motion planner shown in white. The blue sphere is the goal location, the red spheres are obstacles, and the rectangular meshes are obstacles.

ates an instance of the motion model plugin and uses it during the Open Motion Planning Library's propagation step [23]. Since SCRIMMAGE guarantees determinism and lockstep execution, it can guarantee than the planned path that was generated with a motion model is the exact path that is followed by the same motion model.

SCRIMMAGE makes use of C++14 and open source packages such as Boost, Eigen, GeographicLib, JSBSim, and CMake. SCRIMMAGE is hosted publicly on GitHub[5] and is distributed under the GNU Lesser General Public License v3.0. It is the intention of the SCRIMMAGE authors for all code changes made to the core SCRIMMAGE project to be made available to the original project. However, third-party plugin developers can designate their own licenses for their individual plugins. This allows individuals to retain the intellectual property rights for their SCRIMMAGE plugins.

4 PLUGIN ARCHITECTURE

To facilitate flexibility in simulation fidelity, SCRIMMAGE implements several plugin interfaces. A SCRIMMAGE entity is an agent in the simulation that is composed of one or more plugins. Each entity can have multiple sensor, autonomy, and controller plugins, but only a single motion model plugin. The flow of information through an entity's plugin interfaces is shown in Fig. 3. An auton-

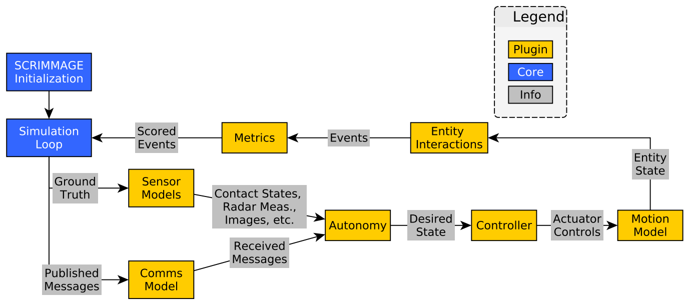


Fig. 3: Plugin order of execution in SCRIMMAGE.

omy plugin queries the entity's sensors and/or the environment's ground truth to make a decision. The types of information that an autonomy plugin could reason over might be the positions of other robots, obstacle detections, map data, etc. The autonomy plugin generates a desired state, which includes position, velocity, and/or orientation information. The purpose of the controller plugin is to consume the desired state and produce actuator commands for the motion model plugin. Finally, the motion model plugin takes the entity's current state and the control outputs from the controller plugin to generate a new state for the entity in global coordinates. SCRIMMAGE also provides a novel communications network plugin interface. The purpose of this plugin interface is to simulate the effects of the communications network on message transmission between entities. SCRIMMAGE provides three default network plugins: the

[5] The SCRIMMAGE source code is publicly available at https://github.com/gtri/scrimmage.

`LocalNetwork` for simulating messages being sent between processes on the same physical platform, the `SphereNetwork` for simulating messages being sent across a radio network, and the `GlobalNetwork` for the transmission of simulation-specific messages that will always be delivered. Without a dedicated network plugin interface, other simulators require the researcher to setup intermediate message topics in order to simulate the effects of the network on message delivery.

Collision detection and interaction between entities is abstracted through the entity interaction plugin interface. An entity interaction plugin can wrap a collision detection library, such as Bullet, or it can perform simple collision detection based on the ranges between entities, such as with SCRIMMAGE's `SimpleCollision` plugin. A complete tutorial for creating SCRIMMAGE plugins is available on the official SCRIMMAGE website. [6]

5 INTEGRATIONS

While SCRIMMAGE can be used in isolation to develop and test novel autonomy and controller algorithms, SCRIMMAGE can interface with other software packages used by the robotics community. SCRIMMAGE has utilities and plugins that interface with ROS, MOOS, and OpenAI Gym [19, 16, 5].

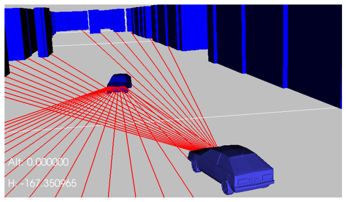
5.1 ROS

There are two methods of interfacing SCRIMMAGE with ROS. The first method uses SCRIMMAGE utilities to abstractly construct publishers, subscribers, and services, such that the plugin uses SCRIMMAGE's API when being simulated in SCRIMMAGE or the plugin uses ROS' API when being executed in a ROS node. The second method emulates the Stage simulator's ROS API, which allows SCRIMMAGE to be a drop-in replacement for Stage when working with the ROS 2D navigation stack.

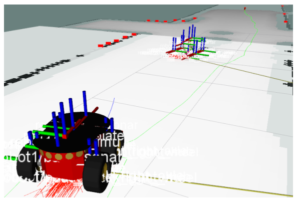
After writing and verifying algorithms in simulation it is a simple step to integrate with ROS as fundamental concepts in the latter map directly to the former. In particular, ROS nodes map to SCRIMMAGE plugins, ROS services are functions provided by one plugin and called by another, and ROS publishers/subscribers map to SCRIMMAGE network devices. Similar to the ROS interface, data is loosely coupled in the form of a publish-subscribe system.

SCRIMMAGE also has the ability to emulate Stage's interface to ROS through the `ROSAutonomy` plugin. The `ROSAutonomy` plugin publishes sensor information to ROS over the `odom` and `base_scan` topics and it accepts commanded velocities over the `cmd_vel` topic. SCRIMMAGE can convert the same maps used by Stage to 3D wall barriers and uses the Bullet Physics Engine's ray tracing feature to simulate a LIDAR sensor. A screenshot of two simulated ground robots in SCRIMMAGE being controlled by the ROS 2D navigation stack is shown in Fig. 4a.

[6] The SCRIMMAGE website is located at http://www.scrimmagesim.org.



(a) A screenshot of the SCRIMMAGE 3D viewer while interfacing with the ROS 2D navigation stack. The red lines represent LIDAR detections.



(b) A screenshot of the ROS 2D navigation stack running in RVIZ.

Fig. 4: SCRIMMAGE / ROS Screenshots.

5.2 MOOS

The Mission Oriented Operating Suite (MOOS) is a publish-and-subscribe system for robotic platforms and MOOS-IvP (MOOS Interval Programming) is an autonomy behavior fusion engine that is commonly used on autonomous maritime vessels [16, 4]. SCRIMMAGE's `MOOSAutonomy` plugin converts the SCRIMMAGE entities into the appropriate MOOS data structure and publishes them to the MOOS community. MOOS-IvP then generates a desired heading and a desired speed and publishes them to the `MOOSAutonomy` plugin.

5.3 OpenAI Gym

SCRIMMAGE provides an OpenAI Gym environment to support evaluation of reinforcement learning algorithms [24]. The user can select from already created environments by changing a few lines in a mission file or they can create their own. The gym environment interacts with scrimmage through pybind11's [11] embedded python library. Users can customize their environment by specifying action and observation spaces to be continuous, discrete, or combinations thereof. Further, as SCRIMMAGE is designed to be a multi-agent simulator, users can run environments with multiple learning agents.

6 PREDATOR/PREY SIMULATION USE CASE

To demonstrate the development and analysis processes when working with SCRIMMAGE, a simulation of two groups of predators trying to capture prey in a swarm was developed. In this experiment, the two groups of predators will be competing against each other in a game to determine which predator group can capture the most prey. The two predator groups will be different in how each group selects prey to pursue for capture. SCRIMMAGE will be used to organize the game experiment, simulate 1000 runs of the games, compute game scores, and aggregate the game results in a batch process. In previous work, the authors attempted to simulate 10's of aircraft in a similar scenario using Gazebo and were unsuccessful in running the simulation faster-than-real-time.

6.1 SCRIMMAGE Mission Setup

A SCRIMMAGE simulation, or mission, is defined in an XML file, which contains the information necessary to run the simulation. The XML file defines parameters needed to control the timing of the simulation such as the simulation step size, real-time warp factor, simulation end time, simulation end conditions, etc. The XML file also specifies the types of entities that will be instantiated during the simulation and the plugins that will be associated with each entity. Since SCRIMMAGE was designed with swarm-based simulations in mind, an entity can be a single robot or a single XML `entity` block can spawn a large number of agents. For the Predator/Prey simulation, there will be three swarm entities: a swarm of prey (Team 1), a swarm of predators (Team 2), and a second swarm of predators (Team 3). The motion model for each agent in the simulation will be a modified unicycle model. The prey will execute a modified Boids flocking behavior [20], while the predators will execute two different greedy capture behaviors. All behaviors were implemented as SCRIMMAGE autonomy plugins. A predator "captures" a prey by publishing a `CaptureEntity` message when it is within a distance threshold of the prey. The prey are only acting on the ground truth state information of other entities in the simulation. After a predator publishes a `CaptureEntity` message, the `SimpleCapture` entity interaction plugin receives the message, computes the distance between the predator and the prey it is trying to catch, and it determines whether the capture is successful. If the capture is successful, the `SimpleCapture` entity interaction plugin has the authority to modify the state of the prey, modify the health of the prey, or remove the captured prey from the simulation. The `SimpleCapture` plugin will also publish a `NonTeamCapture` message, which will be received by the `SimpleCaptureMetrics` metrics plugin, so that the team score can be calculated.

6.2 Entity Motion Models

SCRIMMAGE provides a number of motion models that can be used with most autonomy plugins: simple car, simple aircraft, double integrator, single integrator, high-fidelity JSBSim model, etc. The Boids flocking model demonstrates the desired performance when using motion models that are similar to unicycle dynamics. However, since we wish to demonstrate SCRIMMAGE's 3D capabilities, we extended the 2D unicycle model to a 3D version by adding pitch to the unicycle state, as shown in (1).

$$\begin{aligned}
\dot{x} &= u_s \cos(\psi)\cos(\theta) \\
\dot{y} &= u_s \cos(\psi)\sin(\theta) \\
\dot{z} &= u_s \sin(\psi) \\
\dot{\theta} &= u_\omega \\
\dot{\psi} &= u_\phi
\end{aligned} \tag{1}$$

The 3D unicycle state consists of a 3D position, (x, y, z), its heading, θ, and its pitch, ψ. Inputs to the motion model are the forward speed, u_s, yaw rate, u_ω, and pitch rate, u_ϕ. Not shown in (1) are the limits on pitch rate and yaw rate.

6.3 Predator Autonomy Model

The predator behavior can be decomposed into two primary tasks: selecting the prey to capture and constructing a trajectory to capture the prey. In differential game theory, it has been shown that a pure pursuit strategy in pursuer/evader games is the optimal strategy for the pursuer [10]. Thus, the pure pursuit strategy was used to construct a trajectory to capture the prey. To compute the control inputs required to achieve pure pursuit, the predator first computes the desired velocity vector normalized by its maximum speed, s_{max}, pointing from its own position, p_{own}, to the prey's position, p_{prey}.

$$\mathbf{v} = s_{max} \frac{p_{prey} - p_{own}}{\|p_{prey} - p_{own}\|} \tag{2}$$

The desired heading, θ_d, and desired pitch, ψ_d, can be computed with (3).

$$\begin{aligned} \theta_d &= \tan^{-1}\left(\frac{\mathbf{v}_y}{\mathbf{v}_x}\right) \\ \psi_d &= \tan^{-1}\left(\frac{\mathbf{v}_z}{\sqrt{\mathbf{v}_x^2 + \mathbf{v}_y^2}}\right) \end{aligned} \tag{3}$$

Finally, these desired orientations can be converted into control inputs by setting the desired speed to the maximum speed and implementing simple proportional controllers.

How a predator should select the prey is less obvious than how it should construct its capture trajectory. A greedy strategy will first be employed, where the predator selects the prey based on a distance metric. The predator will select the prey that is closest to it in 3D space. How often the predator chooses its next prey also has to be determined. Should the predator select the closest prey at every decision time-step or should it select a prey one time and pursue the prey until it has been captured? This is the question that this SCRIMMAGE simulation will answer in a statistical fashion. One team of predators (Team 2) will perform a one-time selection of its prey and the other group of predators (Team 3) will be allowed to select the closest prey at every decision time-step. The predator's autonomy plugin was parameterized with a boolean variable called `allow_prey_switching` that switches the selection behavior of the predator, which facilitates running the same plugin with different parameters in the same simulation. It is hypothesized that the predator team that is allowed to switch prey will capture more prey because the predators will be able to opportunistically catch prey that happen to fly nearby.

6.4 Prey Autonomy Model

The prey's behavior is a combination of the Boids flocking model and motor schemas [1]. Similar to the traditional Boids model, each prey agent implements separation, alignment, and cohesion behaviors. The prey agents also implement go-to-waypoint and avoid non-team agent behaviors. The behaviors for avoiding other prey and predators both use the motor schemas formulation for collision avoidance. The avoidance behavior is parameterized by a sphere of influence, S, a minimum distance, d_{min}, the distance between the prey and the other entity, d, and a behavior gain or weight, w_o. The direction of the avoidance vector extends from the other entity towards the prey's own position. The magnitude of the avoidance vector, $\mathbf{O}$, is defined in (4).

$$\|\mathbf{O}\| = \begin{cases} 0 & d > S \\ \frac{S-d}{S-d_{min}} * w_o & d_{min} < d \leq S \\ \infty & d \leq d_{min} \end{cases} \tag{4}$$

The collision avoidance behavior that keeps the prey away from predators has a larger value for d_{min} than the behavior that keeps prey away from other prey. However, both behaviors have the same values for S. Likewise, the weight for the predator collision avoidance behavior is larger than the other behavior weights. The SCRIMMAGE plugin interface provides the plugin developer with an R-tree data structure for fast computation of nearest neighbors, which simulates communicating state information based on a maximum range [3]. The generation of the R-tree is more computationally expensive than a single nearest neighbors search, but queries on the R-tree are more efficient than a nearest neighbors search. The SCRIMMAGE simulation controller computes a single R-tree instance for all agents at each time-step, which is more efficient than each agent executing its own nearest neighbors search.

6.5 Predator/Prey Results

The predator/prey simulation was run 1000 times. The maximum simulation time for each simulation run was 500 seconds. There were 100 prey, five predators in Team 2, and five predators in Team 3. The predators that did not switch which prey they were trying to capture after the initial selection process (Team 2) outperformed the predators that could switch prey at each decision time-step (Team 3). Team 2 captured more prey in 593 out of the 1000 runs. Team 3 captured more prey in 361 runs. In 46 simulation runs, the two teams tied in the number of captured prey. A screenshot of the SCRIMMAGE visualizer during one of the predator/prey scenarios is shown in Fig. 5. The flight paths of the prey in the front of the flock are being disrupted by two predators that have penetrated the flock. However, the prey at the back of the flock are remaining in formation because the predators have not yet penetrated their spheres of influence. The number of prey in the predator/prey simulation was also varied to test how the time to execute the simulation varies as the number of entities

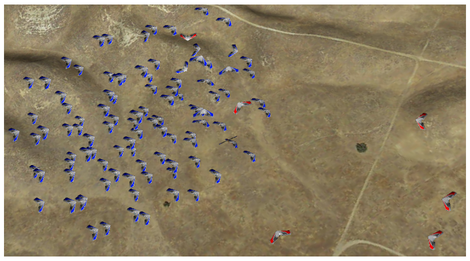


Fig. 5: A screenshot of the SCRIMMAGE visualizer during the predator/prey simulation. The prey are represented by blue aircraft and the predators are represented as red aircraft.

varies. Table 1 shows the ratio of simulation to actual time for running the predator / prey simulation as a function of the number of total agents in the simulation for three motion model plugins of increasing fidelity.

Entity Count	Single Integrator	Unicycle	JSBSim
200	49.6	41.3	1.1
500	13.3	12.2	0.4
800	6.9	5.9	0.2
1100	4.3	3.5	0.2
1400	2.9	2.4	0.1
1700	2.1	1.7	0.1
2000	1.5	1.3	0.1

Table 1: The ratio of simulation time to wall time in a predator-prey scenario for varying levels of motion model fidelity and number of entities. Data was computed using an i7-4790 CPU running at 3.6 GHz using a single thread. Wall time is how long it takes a computer to run a program.

6.6 Predator/Prey Discussion

The hypothesis that the predator team that was allowed to switch pursued prey at each decision time-step would capture more prey than the predator team that was not allowed to switch was incorrect. By reviewing the playbacks of the simulation runs it was obvious that the predators that were allowed to switch were often distracted by nearby prey as they were closing in on a prey that would have been caught. In future work, the prey selection criteria could be improved to move past a distance metric to capture the dynamics of the moving entities. However, this predator/prey simulation in SCRIMMAGE provides a useful tutorial for how to use SCRIMMAGE to conduct robot behavior experiments in a probabilistic fashion.

The results in Table 1 indicate that using a unicycle model does not significantly reduce run-time while moving to a six degrees-of-freedom model with JSBSim significantly reduces run-time. These results highlight the importance of being able to scale the fidelity of motion model plugins to allow the researcher to focus on the aspect of the simulation that is of most concern.

7 CONCLUSION

A motivating example for the use of SCRIMMAGE in simulating large numbers of autonomous agents was presented in the form of a predator/prey game. SCRIMMAGE was originally developed due to the lack of open source, computationally efficient, multi-agent, aerial robot simulators. SCRIMMAGE was also designed with batch processing in mind; thus, its plugin architecture can compute metrics and aggregate results across large batches of simulations. This allows SCRIMMAGE to be used as a platform for experimentation, comparing robot behaviors, and comparing swarm strategies. SCRIMMAGE provides plugin interfaces for ROS, MOOS, and OpenAI. In future work, SCRIMMAGE will be leveraged for multi-agent deep learning due to its low computational overhead and its OpenAI interface. SCRIMMAGE will continue to improve as it is used on more research programs and its open source nature allows the robotics community to provide feedback.

References

1. Arkin, R.: Motor schema based navigation for a mobile robot: An approach to programming by behavior. In: Robotics and Automation. Proceedings. 1987 IEEE International Conference on. vol. 4, pp. 264–271. IEEE (1987)
2. Bailey, M., Gebis, K., Zefran, M.: Simulation of Closed Kinematic Chains in Realistic Environments Using Gazebo. Springer International Publishing, Cham (2016)
3. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. In: Acm Sigmod Record. vol. 19, pp. 322–331. Acm (1990)
4. Benjamin, M.R., Leonard, J.J., Schmidt, H., Newman, P.M.: An overview of moos-ivp and a brief users guide to the ivp helm autonomy software (2009)
5. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI gym. arXiv preprint arXiv:1606.01540 (2016)
6. Collier, N.: Repast: An extensible framework for agent simulation. The University of Chicagos Social Science Research 36, 2003 (2003)
7. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research 17(7), 760–772 (1998)
8. Forrester, A., Keane, A., et al.: Engineering design via surrogate modelling: a practical guide. John Wiley & Sons (2008)
9. Gerkey, B., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th international conference on advanced robotics. vol. 1, pp. 317–323 (2003)
10. Isaacs, R.: Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization, vol. 1. Dover Publications, Inc. (1965)

11. Jakob, W., Rhinelander, J., Moldovan, D.: pybind11 – seamless operability between c++11 and python (2017), https://github.com/pybind/pybind11
12. Kiran, M., Richmond, P., Holcombe, M., Chin, L.S., Worth, D., Greenough, C.: Flame: simulating large populations of agents on parallel hardware architectures. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 1633–1636. International Foundation for Autonomous Agents and Multiagent Systems (2010)
13. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. vol. 3, pp. 2149–2154. IEEE (2004)
14. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multiagent simulation environment. Simulation 81(7), 517–527 (2005)
15. Martinez, J.M., Collette, Y., Baudin, M., Christopoulou, M., Baudin, M., et al.: pyDOE: The experimental design package for Python (2009–), https://pythonhosted.org/pyDOE/, [Online; accessed 2017-09-07]
16. Newman, P., MOOS, A.: A mission oriented operating suite. Tech. rep., Technical Report OE2007-07. MIT Department of Ocean Engineering (2003)
17. Nishimura, S.I., Ikegami, T.: Emergence of collective strategies in a prey-predator game model. Artificial Life 3(4), 243–260 (1997)
18. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., et al.: ARGoS: a modular, multi-engine simulator for heterogeneous swarm robotics. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. pp. 5027–5034. IEEE (2011)
19. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe (2009)
20. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. ACM SIGGRAPH computer graphics 21(4), 25–34 (1987)
21. Rohmer, E., Singh, S.P., Freese, M.: V-rep: A versatile and scalable robot simulation framework. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. pp. 1321–1326. IEEE (2013)
22. Sklar, E.: Netlogo, a multi-agent simulation environment (2007)
23. Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robotics & Automation Magazine 19(4), 72–82 (December 2012), http://ompl.kavrakilab.org
24. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
25. Vaughan, R.: Massively multi-robot simulation in stage. Swarm intelligence 2(2), 189–208 (2008)