An Agent-based Simulation for Modeling Intelligent Munitions

KARL ALTENBURG, JOSEPH SCHLECHT, KENDALL E. NYGARD
Department of Computer Science and Operations Research
North Dakota State University
Fargo, ND 58105
USA

Abstract: An agent-based emergent intelligence methodology is utilized to assemble agents into teams and assign distributed tasks to them in a dynamic environment. The application concerns the management of military missions that utilize intelligent munitions that are capable of searching, detecting, identifying, and attacking targets in a battlefield. The simulator follows a reactive, behavior-based design philosophy. Agents are implemented as asynchronous threads that employ polymorphic signal-based communication. Results to date indicate that the simulation framework is effective in evaluating ways for emergent teams to quickly solve dynamic teaming and task allocation problems.

Key-Words: Autonomous agents, Simulation, Intelligent munitions, Unmanned Air Vehicles

1 Introduction

Intelligent, unmanned, and autonomous flying munitions are of high interest to the military for their ability to search, detect, identify, and precisely destroy enemy targets. Although small and low in cost, such weapons can now be equipped with multiple sophisticated sensors, advanced wireless communications systems, and avionics that provide agile flight control and navigation. The concept of operation involves releasing large numbers of such systems in groups, allowing them to autonomously carry out a mission. They are one specific type of Unmanned Air Vehicle (UAV) among many in use or under development today. Coordination of multiple resources in the battlefield is a major issue. The traditional tenets of pre-planning, centralized control and decentralized execution must be re-examined in a modern engagement of the armed forces [5]. For example, a mission in which many preplanned targets are struck in a short time, must have adaptive mission planning capabilities to address emerging threats and targets remaining. Military mission planners today rely on a sophisticated and constant virtual forward presence in the form of UAVs for reconnaissance. The success of such UAVs, like the Predator, has prompted a heightened interest in the potential of pilotless air forces [8]. In the work we report here, we address the need to coordinate multiple autonomous munitions with an adaptive approach. The core problem is as follows: given a set of deployed intelligent munitions, a suite of available targets, and dynamic attrition and new battlefield information; form teams of munitions and assign them tasks in a manner that maximizes mission effectiveness. Although centralized control using information gathered before mission deployment has been effective in the case of a small number of resources, this is likely to be inadequate when there are many intelligent munitions in dynamic environments. We address the problem with an approach inspired by the observed ability of biological systems such as bees, wolves, and birds to form swarms, packs, and flocks that exhibit globally intelligent emergent behaviors. Variations of this approach for modern warfare are being investigated and are steadily gaining credibility [2]. In this paper, we present the design and implementation of an agent-based UAV simulation framework for experimenting with this approach.

The requirements of such a multi-agent system present several design challenges. Some of the requirements include the explicit modeling of cooperation and communication choices, and the recognition and exploitation of emergent phenomena. The resultant mission controls must also map back into the physical and control capabilities of the intelligent munitions. Combinatorial optimization models have also been developed for adaptive mission planning, but are very computationally intensive. The emergent intelligent systems that we are developing can be used to provide lower bounds on the performance of those models, providing a benchmarking method. The system is also intended to provide for the incorporation of existing centralized, optimal planning methods.

2 Related Work

The simulation agents are inspired by graphical Turtles used to explore predator and prey interactions [1,8]. These interactions depend on models of sensations such as sight and smell. Multiple Turtle-like agents with limited sensory abilities were used by Reynolds to model flocking in his Boids system [10]. StarLogo is a massively parallel extension of Turtle graphics in which thousands or tens of thousands of Turtles interact as independent processes [9]. Like our system, StarLogo was designed to explore and exploit emergent phenomena. Our system involves many Turtle-like agents with varied, but realistic, sensory and behavioral capabilities.

Behavioral design may follow bottom-up or top-down approaches. At the individual agent level, our work is inspired by reactive, behavior-based approaches [4]. At the group-level, we follow the basic design philosophy ideas of Mataric, in which a small set of basic, group-level interaction serve as building blocks for system-wide behavior [6]. These basic interaction include *Avoidance*, *Attraction*, *Following*, *Dispersion*, *Aggregation*, *Homing*, and *Flocking*.

3 Simulation Design

The ultimate goals of the system are to provide: 1) a means of demonstrating realistic battle scenarios, and 2) an end-to-end working system for experimentation with alternative methods for managing swarms of intelligent munitions.

The design of the system was divided into several components: 1) the agents, 2) the agent's environment, and 3) the mechanisms for communication. Communication was incorporated into the agent sensory requirements, and provides a fundamental way to support inter-agent cooperation.

In designing the system, one approach would be top-down, working from developed battle scenarios down to the agent and environment requirements. An alternative approach would be bottom-up, developing design mechanisms to model basic behaviors of the agents in such a way that the battle scenarios could be handled indirectly. Our approach is fundamentally bottom-up, but does have some top-down aspects, following something of a hybrid model.

3.1 Agents

With a bottom-up orientation, several low-level abilities were specified. These abilities can be combined to form composite abilities. At the lowest level, agents are able move by incremental changes in direction and velocity. In our initial experiments, agents are provided with the following discrete alternative available speeds: i) a pre-specified normal (cruise) speed, ii) a pursuit speed, nominally 25% faster than normal, and iii) a stationary speed, to represent loitering in a fixed location (or hovering in a helicopter-like UAV). The model supports an arbitrary number of discrete speed steps. Discrete choices provide modeling simplicity and yet allow a full range of realistic alternatives.

A *Homing* and *Aggregation* behavior is fundamental to the simulation model. When carrying out this behavior, upon detecting the appropriate signal, agents move towards that signal. The reverse behavior, *Fleeing* and *Dispersion,* is also fundamental. When carrying out this behavior, upon detecting the specified signal, agents will move away from that signal. In all cases, the specific Agent behavior is the reaction that it takes upon detecting a signal. It is important the reactions be state-dependent. For example, an agent active in survival or the execution of a critical mission task should ignore more trivial signals. However, if no important tasks are pending at the moment, responding to nominal signals is appropriate behavior. The behaviors of an agent are modulated through either an external signal or internal events (i.e., counter or timer), and based upon the state of the agent at that time. Available internal states include *wander* (move in the absence of a goal), *fight* (attack a target), *flee* (retreat as quickly as possible), *follow*(traverse a path determined by a leader), or *lost* (move in the absence of locational information) .

Following the general philosophy of reactive, behavior-based control, sensors are coupled to actuators through behavioral modules. Each behavioral module has sensor inputs, internal state, process logic, and actuator output. These behavioral modules are coupled to the sensors and actuators so that these resources are appropriately shared. This tight coupling between sensation and action is characteristic of a reactive, behavior-based agent control philosophy [4]. Agents are supported with a hierarchy of behaviors that follow a subsumption-like architecture. Alternatively, a summed vector approach could have been used [3]. The lowest module provides the most minimal level of control; the next layer is astride that layer and overrides it under the proper external or internal signal. A summed vector approach would have required that actuator inputs be weighted.

Given the state-based nature of the agents, behavioral modeling based on state transition diagrams is appropriate and intuitive. Figure 1 shows a state transition diagram for an agent that wanders until it detects an attracting alarm signal. It then enters the *attracted* state and homes in on that signal as long as it is sensed. When no signal is sensed, the agent returns to the *wander* state. However, if an agent detects the proximity of another like agent, the agent enters the *avoid* state and is repulsed away from the other agent.

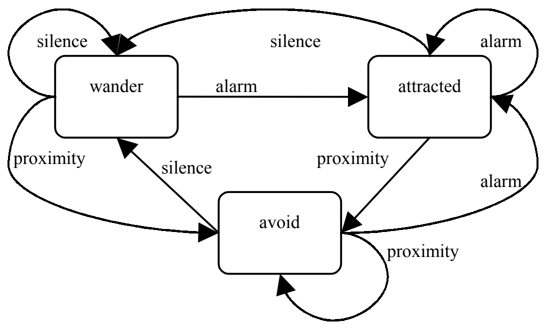

Fig. 1. State transition diagram for agent behavioral modeling.

In some cases signals cause complete state transitions, but the same signal may be handled differently in different states. During *searching*, a *proximity* signal generated by one agent may cause another agent to avoid the sender by a substantial distance. This permits better dispersion in an area. However, when *homing* in on a target, the degree to which a friendly agent is avoided is lessened. This permits more agents to get close to the target. In some cases, a signal in one context causes a *homing* reaction while in another context causes *dispersion*. For example, if an agent is leaving its home base on a search mission, and home base transmits a “home base” signal, the agent follows a disperse behavior to better search away from the base. However, when the agent is low on fuel, it will follow a homing behavior on the base signal. Lastly, differential attention to different signal is permitted through a prioritization scheme.

The need to detect a signal mandate a need for sensors. Each sensor is characterized in several ways as described below in the section on Communication. The potential for sensor access is equal among all behavioral modules (layers). Behavioral modules have direct input into another layer, like a virtual sensor. A sensor-controller-actuator registry is utilized to support an explicit mapping among these agent resources.

The arbitration of signals (or behaviors) is a central issue in the design of reactive agents. The question concerns how to represent multiple, behavior transitioning signals, that are evaluated simultaneously. For example, how should a signal to *home* combined with a signal to *avoid* collision be represented? We employ internal state timers to handle this situation. Alternatively, all triggered behaviors could be allowed to fire simultaneously, which is sufficient if there is adequate separation of signals and behavior in an agent’s state-space. However, the possibility of inducing contradictive behavior led us to avoid this design choice.

Two additional system requirements are the ability to dynamically create and destroy agents and for one agent to contain another. Supporting agents with or within agents would require allow agents to be geographically co-located. This is needed to model such things as a surface-to-air missile (SAM) launcher and its missile, a UAV that launches its own missile, or a cargo aircraft that offloads dozens of UAVs. Allowing agents to be dynamically added or removed from the list of active agents is useful in modeling a pop-up threat or the self destruction of a UAV.

3.2 Environment

The agents exist within a specific environment, which is also the medium through which signals are carried. For each signal, the environment determines its strength at any particular location, just as the physical properties of our own atmosphere or hydrosphere determines the characteristics of signal propagation traveling through. We follow a paradigm in which one imagines that an agent sends a message to the environment, providing the information that it generated a particular signal at a particular location. The environment is then responsible for manipulating that signal according to the model physics of the environment. We base the attenuation of signal strength on an inverse-square function.

The environment also maintains a record of the physical elements within the environment. Thus, Agents can “ask” if they have collided with something or use a sensor to ask if there is something nearby. All physical elements generate a nominal signal of their presence within the environment. As with our environment, no two physical elements may occupy the same space.

3.3 Communication

Agents do not communicate directly, but rather through the environment. Likewise, they act through the environment. We model communication as an elaboration of sensation and action. Communication is characterized by the existing elements of signals, sensors, and behaviors.

Signals come in many forms. We basically model a discrete set of distinct signals, say signals A, B, C, and D. Each of these signals is characterized by its point of origin and strength. The point of origin is the location of the sender (transmitter) and located in x-y coordinates. The strength of the signal is a relative value, scaled from 1 to 100 and mapping into typical minimum and maximum transmitter capacities. For a commercial AM radio, for example, the 1 represents 1 Watt, and 100 represents 100,000 Watts. A fourth characteristic of a signal is directionality. Many signals are omnidirectional (e.g., radio broadcasts), radiating in all directions with nearly equal strength. Other signals, such as a laser beam, have a very narrow angle through which they travel. Two additional parameters model the relative angle range through which a signal is potentially detectable. An omnidirectional signal has an angle range of 0 to 359 degrees. A narrow signal transmitting to the south might have a range of 170 to 190 degrees. A data structure that captures these four characteristics, signal type, strength, origin, and directionality, is sent to the environment.

Sensors are the sensory "organs" for the agent. They can be a simple radio antenna or a directional laser receiver. For each signal type, an agent needs a particular sensor to detect that signal. Whether a signal could reach an agent is determined by the environment. An important design decision concerns whether the environment "pushes" signals to agents or agents "pull" them from the environment. We utilize a pull approach, the agents actively collect signals from the environment. Essentially each sensor "asks" the environment questions like "what's out there?" or "Do I hear any signals of type B?" The environment then performs the appropriate calculations to answer the question posed by the agent.. The alternative would be a design in which the environment "tells" all nearby agents of a signal. This alternative is more realistic in terms of the operation of a real system. However, such a design would often push signals that are being ignored by the agent, wasting computational effort in the simulation.

As with the signals, sensors has several characteristics. These include modality or medium (sound, light, chemical, radio, ultrasonic, etc.), discrete or continuous, and directionality (ranging from omnidirectional to narrow-focus unidirectionality). The directionality of the sensor allow the modeling of situations where detection of a signal is possible only when approaching from a particular direction. Or, more simply, sensors may be shadowed by the agent itself, much as a person's ears are shadowed by one's head and outer ear. This allows for bearing on a signal based on differential reception – that is, orienting until a signal is either strongest straight ahead, or equal on both sides and getting stronger in the direction of travel.

4 Simulation Implementation

The simulation was implemented with the Java Development Kit version 1.3. It is highly object-oriented and utilizes the concurrency features provided by Java Threads. What follows is an outline of the major classes and their interactions within the simulation.

4.1 Agents

Each agent in the simulation is represented as an object extending, or inheriting from, the Java *Thread* class. By doing this, not only is the simulation taking advantage of concurrent execution, but it is also modeling the autonomous agents more realistically; the agent threads, like UAVs, run asynchronously.

The agent class is abstract; meaning that concrete objects cannot be created from it. To instantiate an agent, a subclass is created that implements a *run* method, which is the main execution loop for an agent thread. Within an agent's *run* method are calls to various methods that modulate its behavior. For example, an agent subclass may contain a method which enables it to flee from a specified location. The decision as to which behavioral method is chosen for execution is modulated by signals it receives or internal events.

In addition to the properties that the abstract agent class inherits from the Java *Thread* class, it also contains methods that enable it to move in within the environment. These basic methods of movement are be aggregated by concrete agent subclasses into their behavioral methods mentioned above.

4.2 Environment

An environment class was created to provide a substrate within which the agent objects exist and communicate. It contains registration methods to add and remove agents from an *environment* object. A linked-list data structure contains references to all of the *agent* objects contained within an environment. More specifically, the list contains references to *registered-agent* objects. The *registered-agent* class is an aggregation of an *agent* and a collection of references to signals that have been transmitted past the agent. The need for this collection is driven by communication modeling, described in the next section.

4.3 Communication

The core of the simulation framework supports communication between agents via the environment in which they are registered. This communication is accomplished by transmitting and receiving *signal* objects. The *signal* class encapsulates the origin location, angle of origin, angular range, strength and message.

To transmit *signal* objects to the environment, the agent contains a *transmitter*. Similarly, *receiver*s are utilized for an agent to receive signals from the environment. To implement these capabilities, each *agent* class contains a collection for *receiver*s and a collection for *transmitter*s. Methods are also available to dynamically add and remove both *transmitter*s and *receiver*s to their respective collection.

As with the *agent* class, the *signal*, *receiver* and *transmitter* classes are abstract. This choice was governed by the need for *agent*s to have different types of *receiver*s and *transmitter*s with different properties (such as angular range). Since all types of *receiver*s need a method to receive signals, it is made abstract. Similarly, with *transmitter*s the method to transmit signals is made abstract. For each pair of *transmitter* and *receiver* concrete subclasses, there is also a corresponding concrete subclass of *signal*, which implements methods for returning references to the class required for receiving and transmitting it. The net result of such a class structure is the powerful ability for polymorphic processing of signals by the agents and their environment.

When an agent wishes to transmit a signal, it passes the information necessary to create a signal (except the angular range, which is only known by the transmitter) to an appropriate transmitter. The transmitter then creates a *signal* object and passes it to the environment. Next, the environment analyzes the *signal* which was passed to it and decides which *registered-agent*(s) it would reach. For each *registered-agent* the *signal* is to reach, the environment places a reference to the *signal* in the *registered-agent*'s signal collection.

The process is reversed for signal reception; however, the active role is still played by the agent. The agents are constantly polling the environment for any new signals. If an agent has any new signals (e.g., *the registered-agent*'s signal collection contains *signal*s), it receives them *en masse* from the environment. Once received, the *agent* iterates over the *signal*s and selects an appropriate *receiver* for actual reception. If the agent does not contain the appropriate *receiver*, the *signal* is discarded.

An argument could be made that it might be more appropriate for the environment to do the work of checking if the *agent* has the appropriate *receiver*. However, since the *agent*s are running as concurrent *thread*s and access to the environment is synchronized, it is more efficient for the *agent* to do this work.

5 Current Results

The simulator serves as is framework to launch further research in UAV cooperative control. We are applying the simulator to evaluate control of multiple intelligent munitions, but the framework is general enough to support other types of UAVs with alternative capabilities, provided that they are agent-based and designed to develop emergent intelligent behavior. It is possible to evaluate configurations with an arbitrary number of autonomous agents in environments with varying characteristics and behaviors. One emergent pattern that has been simulated concerns synchronized simultaneous strike team attacks. One aspect of such an attack is the establishment of appropriate physical separation. In general, this is a problem of N-point, equal-angular distribution, where N represents the number of directions from which agents are approaching. For example, in a 3-point synchronized attack, agents should distribute themselves about a target separated by 120 degrees. Using simple rules of repulsion from like agents as well as attraction to identified targets, this problem can be solved with emergent intelligence (see Fig. 2).

We are also investigating the interplay between the relative size/strength and type of signals. For example, we are investigating issues like how increasing the radius of a repulsive signal effects the agent distribution about an attractive target.

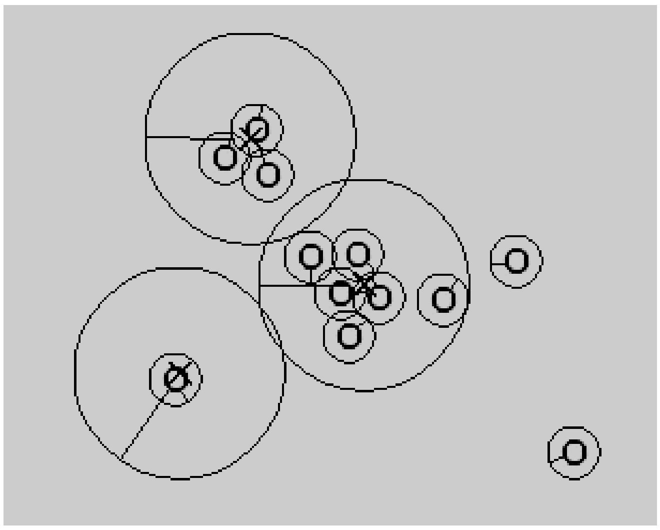
Fig. 2. UAV (small circles) and mobile target (Xs) agents. Around each agent is a circle depicting signal range and a line indicating the direction of movement.

6 Future Work

The agent framework offers considerable potential for further investigations. One overarching project goal involves evaluating mechanisms for combining pre-planned missions developed by off-line optimizers with reactive behavior emergent intelligence capabilities. In real mission planning, a balance between highly optimized pre-plans and reaction is likely to be effective.

We are also developing a language for describing scenarios, such as the geographic area of interest; the type, number, and location of threats, targets and friendly agents; and strategic and tactical objectives. The current framework will support the implementation and validation of the scenario-description language.

7 Conclusion

A simulation framework for investigation of multiagent systems was designed, developed, and implemented. This framework is now poised to help answer questions of whether realistically bounded agents can be provided with behaviors that exploit emergence to derive team-base solutions to distributed tasks in dynamic, hostile environments. Early results show that the use of reactive behaviors and simple external signals can induce spatial coordination for multi-point attacks. The system provides a specific control system for multiple UAVs. The system also provides lower bounds on performance metrics for more traditional control systems based on optimization models, enabling benchmarking of performance.

References

[1] Abelson H, diSessa AA. 1980. Turtle Geometry: The Computer as a Medium for Exploring Mathematics. MIT Press: Cambridge, MA, 477 p.

[2] Adams TK. 2000. The real military revolution. Parameters: US Army War College Quarterly. Autumn: 54-65.

[3] Arkin RC. 1992. Cooperation without communication: multiagent schema-based robot navigation. Journal of Robotic Systems. 9(3):351-364.

[4] Brooks RA. 1986. A layered control system for a mobile robot. IEEE Journal of Robotics and Automation. 2(1):14-23.

[5] Grant R. 2002. The ware nobody expected. Air Force Magazine. 85(4, April):34-40.

[6] Mataric M. 1993. Synthesizing group behaviors. pp 1-10. In: 1993 IJCAI Workshop Series: Dynamically Interacting Robots. 139 p.

[7] Murphy E. 2001. Last of a dying breed? IEEE Spectrum. 38(12, December): 17.

[8] Papert S. 1980. Mindstorms: Children, Computers, and Powerful Ideas. Basic Books: New York, 230 p.

[9] Resnick M. 1994. Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. MIT Press: Cambridge, MA, 163 p.

[10] Reynolds CW. 1987. Flock, herds, and schools: A distributed behavioral model. Computer Graphics 21(4):265-280.

This research was sponsored by the U. S. Air Force Office of Scientific Research