

An Agent Based Framework for Modeling UAV's

Nathan Huff, Ahmed Kamel, Kendall Nygard
Computer Science Department
North Dakota State University
Fargo, ND 58105
Nathan.Huff@ndsu.nodak.edu

Abstract

An agent based system is used to model unmanned aerial vehicle missions utilizing various control models. The system uses a multi level approach to modeling a UAV in order to support approaches to mission planning that are different in their approach. This is accomplished using an agent system through which different levels of control can communicate. The communication takes place through defined interfaces allowing modules to be interchanged with minimal changes to the system.

1 Introduction

Our research group is working on different control and planning systems for unmanned aerial vehicles. As a part of this study we designed a system that could be used to compare the effectiveness of the different approaches side by side. Conceptually this seems like it should be simple, however it becomes difficult because we employ several approaches to mission planning that work in very different ways. For example one of our approaches is a bottom control using swarm behaviors. This approach follows a very hands on and reactive style to mission execution with very little in the way of collective planning among the UAV's on the mission. Another approach is a linear programming approach to planning that tries to create an optimal set of assignments for all UAV's on the mission and therefore assumes that all the UAV's have the same information and will all come to the same conclusion of what the mission is. In short, we have to accommodate models that have output on a very low level such as speed up and turn right 5 degrees to models that have outputs such as UAV 5 attack target 6. Also the subgroups involved are using different languages to implement their planning systems. The major languages in use right now are java and C/C++.

As well as the many different frameworks that have to be supported it is also desirable for the system to be able to be modified as the simulator is running so that the effect of disruptions in communications and unknown targets and threats can be added during a simulation run. This allows for the observation of the decay of the model as the real situation and the models perception of the situation start to diverge. In the end, to address all of these issues we decided to use an agent based approach to the problem. This lets us easily add, remove, and substitute parts of the system without rewriting all the other parts of the simulator.

2 Related Work

The original idea of using an agent based system came from one of the other subgroups on our team. The swarm group had implemented an agent based simulator for their research which is described in a previously published paper [1]. We looked at trying to expand their simulator into a more generic simulator that could be used with the other groups in our team. However, after examining their simulator, we decided that it was too heavily geared toward the reactive bottom-up approach that they were researching, and it would be very difficult to integrate the planning mechanisms written in other languages so while some of the ideas of the system were used our system is pretty much a complete rewrite.

3 Design

The major design of the simulator was driven by the need to support multiple planning approaches into the same simulator while changing the simulator as little as possible while doing it. We also wanted a simulator that would allow us to easily change the parameters of the simulation such as communication and sensor

reliability on the fly in the simulator to get a better idea on how the planning approaches handle problems that would be encountered in a real world application of the algorithms. There was also a strong desire that the simulator be able to be used for demonstration purposes so we had to be able to visualize the simulation as it happens, and it would be desirable for the simulation to run in as close to “realtime” as possible.

These requirements led us to an agent based design. There are three main visible types of agents in the system. These are the UAV’s the control of which is the main topic of our teams research, the sensors and transmitters which are how the UAV’s see their surroundings and communicate with each other. Finally, we have targets which give the UAV’s something to do. There are also several supporting agents which bind the whole system together. These agents are described in section 4.

3.1 UAV’s

The UAV’s are probably the most interesting and important agents in the system. The actual design of a UAV is broken down into at least two separate agents. There is a physical UAV agent and at least one logical UAV agent. This split is what allows us to support the different planning approaches without large changes to the simulator. It gives a clean break between the physical control of the UAV and the reasoning that drives the action of the UAV.

3.1.1 Physical Agents

The lowest level of a UAV agent is the physical agent. This agent is mainly concerned with the physical interaction with the environment. It knows for instance how fast the UAV is going, what the heading of the UAV is, what the UAV’s current position is, and how much fuel the UAV has left. It also interacts with the sensors and transmitters.

The basic purpose of the physical UAV is to provide a consistent interface for its logical counterpart to interact with the rest of the system. This is the easiest way to hook the logical agents into the system because no matter what planning system is controlling the UAV the physics of a UAV remains constant. It also provides a nice abstraction layer for the sensors and transmitters on the UAV. For the most part models need to send messages and receive data. It does not matter if the transmitter is a radio or a laser based line of site transmitter, same goes for sensors. It is however important to the simulator what type of transmissions is occurring. This way we can have

the physical agent handle deciding how a message is sent and which sensor is detecting what. The physical agent also checks for physical interactions such as the UAV crashing into the ground or another UAV.

The interaction of the logical agent with the physical agent takes the form of commands such as set course and set speed. The logical agent can also query the physical agent for information such as the current position, the amount of fuel left, and the current time. The physical agent can also notify the logical agent of events that occur such as a sensor detecting a target or a receiver getting a message from another UAV. As long as the logical agents use these defined messages any logical agent can be put into the system without having to change any of the rest of the system.

3.1.2 Logical Agents

Logical UAV agents are a wide collection of different agents. Some logical agent are in fact a collection of several agents. To explain why it is often necessary to use several agents, it is easiest to refer to a diagram that our team uses to refer to the levels of control in a UAV (Figure 1). The physical agent resides right around the bottom level. The planning strategies that we use range from the top level down to the second to last level. At the higher levels the solution to problems tend to take on the form of UAV to task assignments where a task is a high level description like strike target A. These assignments do not give information such as the path to get to the task. Also some tasks have an ordering dependency where task A must be done before task B. This is also not always taken into account when giving directions to the physical agents. The way to overcome this problem is to use several agents in a pipeline to frame the solution into a form that the physical agent can use. For simplicity, for the rest of this section the we will refer to the collection of agents as the logical agent.

The logical agent is what decides the behavior of the UAV. Logical agents work as an event-driven system. Events such as target detected, waypoint reached, or a message from another UAV are sent by the physical UAV agent and depending on the planning strategy can generate various actions or no action at all. An example might be if a target is detected that the UAV did not know about before, it could do several different things. It could either strike the target immediately, store the target’s location for later, or send a message to other UAV’s about the target. The behavior of all of the logical modules ,however, is beyond the scope of this paper.

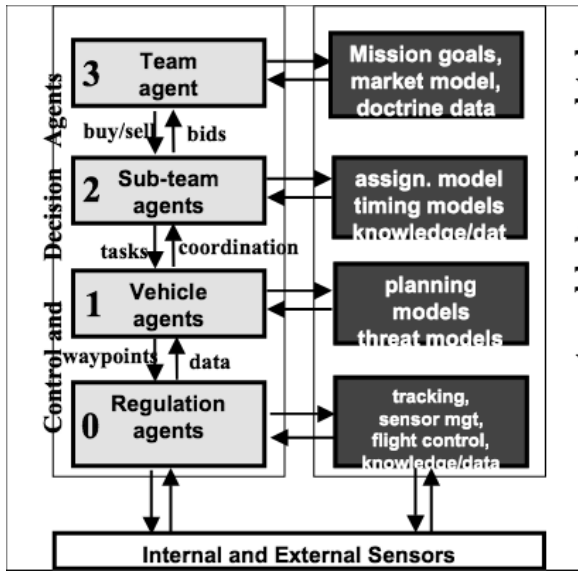


Figure 1: UAV Control Layers

3.2 Sensors/Transmitters

One of the harder design decisions was whether or not to make sensors and transmitters full fledged agents or to try and make them part of the physical agent. We framed them as agents since there are several types of sensors and transmitters and different UAV's would need to carry different sensors. This lets us easily assign sensors and transmitters to UAV's and it lets us easily set parameters on them such as power and reliability.

Sensors can be divided into two main categories, active and passive. Active sensors are used to detect targets and watch other UAV's. These are sensors such as radar or video cameras. They are almost always directional and have a maximum range. Passive sensors on the other hand are things like radio receivers and are usually used for communication between the UAV's. These do not have a range associated with them. Instead the range is associated with the transmitter. They can be non directional such as a radio receiver or directional such as when line of site communication is used. Both types of sensors can be viewed as event generators for the UAV's.

Transmitters are the main way that the UAV communicates with other UAV's on the mission. Transmitters have properties associated with them that are similar to the passive sensors since they are used in conjunction with them. As mentioned above, transmitters have a transmitting power associated with them. This power is how the range of a transmission is calculated.

We use the inverse square rule for the drop off of the transmission to calculate the actual range of transmission. Any UAV within range of the transmission that has the correct receiver will then receive the transmission.

3.3 Targets

The main purpose of targets in the system is to provide tasks to the UAV's. Targets have two main attributes one is type and the other is status. Targets can cover a broad range of actual real world objects from tanks to SAM sites. This type is used by the planning system to assign a value to a target. It can also be used to choose an attack strategy. For example if the target is a SAM site it is probably a good idea to try and strike it from multiple sides since some of your UAV's might get shot down during the attack. When a UAV goes to strike a target it is usually a three pass process. First a UAV classifies a target getting it's type. Then a UAV strikes a target. Finally a UAV does a battle damage assessment to see what the status of the target is. Status is either alive or dead. If the target is still alive then another strike needs to be done.

In the current simulator the design of target agents is very simplistic. Targets sit in one spot and pretty much wait to get destroyed. The design of the simulator however allows for complex targets that have their own planners to try and escape destruction. It is also an eventual goal to have offensive targets such as the SAM sites listed above. This is in the future work section of the paper.

4 Implementation

The implementation of the system is done as a distributed agent system using SRI's Open Agent Architecture(OAA) [2]. All of our agents are written as separate programs that talk to each other through the OAA. Except for some of the logical UAV agents which are written in C/C++ all agents in the simulator are currently written in java. We chose to use java because it provides a good object oriented system with a large number of built in objects allowing us to concentrate on writing agents and not on support code. It also offers the benefit of being platform independent which is a benefit since our team develops on a mix of windows and Linux. Figure 2 shows the agents that are described below.



Figure 2: Agent Architecture

4.1 Open Agent Architecture

The open agent architecture [2] is a distributed agent framework that the simulator is built upon. The main part of the OAA is called the facilitator agent. The facilitator agent allows other agents to register with it and advertise their capabilities. It also allows agents to request other agents to solve problems. Messages are passed using a format called the inter-agent communication language which is based on the style of prolog. For example to request an agent add two numbers, the request would look like `add(7,8,X)` and the response would be `add(7,8,15)`.

While several agent architectures exist (such as the Java Agent Framework [3] and RETSINA [4]) we decided to go with OAA for several reasons. The most important reason was the multi language support that OAA provides. Most of the agent architectures are focused on one language and that would mean that we would have to completely rewrite some of our planning systems. The OAA on the other hand supplies libraries for several languages including C and java which are the ones we were interested in. Secondly the OAA is a distributed agent system. The messaging system uses TCP sockets for communication which means agents can run on different computers. This is important to us because we want to be able to simulate problems with many UAV's and some of our planning systems are fairly processor intensive. So if we want to run a simulation at a reasonable speed we will need to distribute the load among several machines. Finally, the OAA provides a generic way for agents to register capabilities so it is possible to pull out an agent and replace it with another one. As long as the agents register the same capabilities with the system none of the other agents will know the difference.

4.2 Communication

Communication between agents is accomplished by going through the OAA facilitator agent. The first step of the process is that an agent creates a list of the possible services that it can provide and associates a callback function with them. Then it publishes that list to the facilitator agent. Whenever another agent requests that service the facilitator will lookup which agent or agents have published the ability to provide that service and then collects the answers provided by all agents providing that service and returns them as a list to the requesting agent.

It is also possible to specify parameters with requests such as the maximum number of replies and the address of the agent that should be asked for an answer. We use the later parameter in our simulator for communication between the logical and physical UAV agents. Because all physical agents are instances of the same agent and there are usually several instances of the same logical agent running. If we did not use the address parameter all physical agents would respond, and vice-versa. The exchange of addresses is accomplished during agent startup. The logical agent is started first. It requests it's address from the facilitator and then starts a physical agent and provides it with it's own address. During startup the physical agent uses that address to send it's address back to the logical agent through the facilitator agent. After the handshake, all messages can then be directed to the appropriate physical or logical agent.

4.3 Supporting Agents

In addition to the visible agents in the system there are several agents that work behind the scenes to make the simulation run. These agents include the simulation clock, the communication agent, a reporter agent, and several reportie agents.

4.3.1 Clock

The first and most important supporting agent is the simulation clock. The simulation is a discrete simulation so it is the clock that tells the other agents that another time period has started and they need to perform their actions for that time period. It does this by sending messages to the other agents. For instance it sends a `run_agent_cycle` request to all of the physical UAV agents which causes them to move. The main activity of the clock is a loop that increments the time counter tells the UAV's to move and then tells the reporter to give a report of what happened during that

time period to all of the reporties. The clock exports services to start the clock, stop the clock, get the current time, and calculate the ratio of simulation time to real time. The ratio of simulation time to real time is used as a crude performance measure of the simulator.

4.3.2 Communication Agent

The communication agent is used during communication from one UAV to others. The main function is to calculate which UAV's would be in range to receive the message. So when a UAV wants to send a message to other UAV's it first sends it to the Communication agent which then based on the current position of the UAV, the strength, and properties of the transmitter decides which other UAV's are in range. It then delivers the message to those UAV's.

The purpose of this extra layer is to provide a central place that we can effect if we want to study the effect of communication (such as making message delivery less reliable) on the models. It also gives us a centralized place to record all of the messages sent by the UAV's and supply the list to the reporter.

4.3.3 Reporter

The reporter agent is used to collect data from the simulator and distill it into a report that is then sent to reportie agents. The UAV's, targets, and communication agent all send a report to the reporter during their execution. The reporter takes these separate reports and creates a collective report that it then sends to the reporties. We decided to use a reporter agent because we had several other reportie agents that needed similar information. The reportie agents could then parse out the information they need. Otherwise we would need to send different reports to different reporties which would be very difficult to maintain.

4.3.4 Reporties

Reporties are the counterpart agents to the reporter. The reportie agents are used for output and record keeping. Currently, we have two reportie agents that we normally use. The first is a record keeper which archives the reports into a file. This allows us to go back and analyze what happened during the simulation. The second is a graphical output of the current state of the simulation. It shows the position of the UAV's and targets in the simulator.

4.4 UAV's

As stated earlier in the paper a UAV is composed of multiple agents. The physical agent is mostly responsible for the physical interactions of the UAV. During a time period the UAV turns toward it's next waypoint, moves, checks to make sure it did not hit anything, updates it's fuel status, checks it's sensors, transmits any pending messages and if anything interesting happens like finding a target reports it to the logical portion of the UAV.

The implementation of the logical UAV varies with the model that is being simulated so we will concentrate on the interaction with the physical agent. The main method of controlling the physical agent is through the use of waypoints. The logical agent is responsible for generating a list of waypoints and sending it to the physical agent to follow. A waypoint can be used for navigation or a task can be specified at that position. The logical agent can also set the speed of the UAV within the constraints of the UAV type. Finally the logical UAV can tell the physical agent to have a transmitter send a message. Since the logical agent is event-driven there are messages generated from the physical agent. Currently these consist of waypoint reached, target found, message received, and task completed. It is also possible for the logical agent to ask for the current time, the fuel status, and the current position, and the current speed of the UAV. It is possible that we may have to add other messages as other planning models are developed, but we are trying to keep the interaction as simple as possible.

Sensors and transmitters are also an integral part of the UAV. These agents are "attached" to the physical UAV. The area that a sensor can see is defined by the location of the physical agent that it is attached to, the offset in degrees from the nose of the physical agent, and the range of the sensor. These attributes are used to define a cone shaped section of the world that the sensor can see. When a sensor is checked by the UAV it will report all other agents that are within that cone. Transmitters can be omni-directional or directional and sends that information with the message to the communication agents which then delivers it to the other UAV's.

5 Future Work

We currently have a minimal template for the simulator which allows for future expansion. One of the first things that needs to be worked on is the three dimensional aspect of the simulator. Currently, the simula-

tor is in effect a two dimensional simulator. We have designed the simulator with the idea of it working in three dimensions, but right now the height of most objects is not used. The adding of a third dimension can have quite a few aspects such as in collision detection. There are also some very complex things that can be done with three dimensions such as terrain being able to hide things such as targets or possibly UAV's from being detected. This however requires quite a bit of calculation to figure out so we have to figure out if the realism that it contributes would be worth adding or if it will to be a complexity we would rather avoid.

Another area that bares work is in trying to improve the simulator's performance. One of the drawbacks of the loose coupling of an agent based system is that all of the communication between the agents can slow down the system. There are a couple of ways that we are looking at to increase the simulator's performance. The easiest way is to increase the number of computers that the simulation is running on. This only scales so far however. Another area that we are looking into is using a feature in the OAA that lets you bypass the facilitator agent and send messages directly between agents. This reduces some of the communication and takes some of the load off of the facilitator agent. The best candidate for this is the agents that make up a UAV.

One of the most interesting avenues of future work is in the area of the targets. The current targets are static. However, the simulator offers the ability to expand them in several ways. One of the first thing we would like to look at is implementing targets that can move, possibly implementing their own AI that can try and avoid or fool the UAV's. This would offer a lot more realism. Another interesting area in targets is to have targets communicate with each other. This way if one target saw a UAV it could communicate with other targets and have those targets react accordingly. Finally, we would also like to implement offensive targets. Radar and SAM sites being the best example. If a UAV where to fly over a SAM site it is very probable that it could be destroyed.

6 Conclusions

We created a simulator that could simulate a wide range of planning approaches while changing a minimum of the simulator. We chose to use an agent based approach. In order to support multiple planning approaches, we broke the representation of a UAV into multiple agents. This allowed us to keep most of the simulator intact while plugging in different planning

modules. So far, it appears that our approach is working as we had hoped. We are currently in the process of testing our simulation module over different planning approaches.

References

- [1] Altenburg, Karl; Joseph Schlecht; and Kendall E. Nygard. *An Agent-based Simulation for Modeling Intelligent Munitions*. Proceedings of the Second WSEAS Int. Conf. on Simulation, Modeling and Optimization, Skiathos, Greece, 2002.
- [2] Cheyer, Adam and Martin, David. *The Open Agent Architecture*. Journal of Autonomous Agents and Multi-Agent Systems, vol. 4 , no. 1, pp. 143-148, March 2001.
- [3] Vincent, R.; Horling, B.; Lesser, V. *An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator*. Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems., Volume 1887. January, 2001.
- [4] Sycara, K., Paolucci, M., van Velsen, M. and Giampapa, J. *The RETSINA MAS Infrastructure*. To appear in the special joint issue of Autonomous Agents and MAS, Volume 7, Nos. 1 and 2, July, 2003.