Balancing automated behavior and human control in multi-agent systems: a case study in RoboFlag

Philip Zigoris, Joran Siu, Oliver Wang, and Adam T. Hayes
Department of Computer Science
Cornell University, Ithaca, NY 14853 USA
\{zigoris,jss65,ow22\}@cornell.edu
\textsuperscript{2}Department of Computer Science
California Institute of Technology, Pasadena, CA 91125, USA
athayes@caltech.edu

Abstract
Many potential robotic applications require operation in highly dynamic environments and, in multi-agent systems, some degree of coordination between robots. Often it becomes necessary to include an element of human control in order to compensate for limitations in the automated reasoning available to the agents. The project focuses on how one might include a human operator in the feedback loop of multi-agent system, where the agent to human ratio is greater than one. Using the RoboFlag environment as a test-bed, we have implemented various levels of human interaction ranging from the most basic motion control to high-level behavior assignments. Evaluating our system in real time competition has shown that multiple levels of interaction are necessary for successful operation. Because the agent to human ratio is relatively high, it becomes difficult for the human to maintain low-level control of all the robots, but in critical or novel situations higher-level automated intelligence must be circumvented by the human.

1. Introduction
Artificial intelligence is in its early stages for dealing with highly dynamic environments, so it becomes necessary to include human control in the feedback loop. The appropriate way to do this depends on the nature and criticality of the system. For instance, the air force uses remote controlled planes that require three operators \textsuperscript{5}. In this situation the human controls most of the details of the plane’s behavior. Then consider a highly autonomous example from the academic realm such as Cornell’s RoboCup\textsuperscript{4}, which only provides an interface for starting and stopping the game.

The focus of our project is how to integrate human control into a multi-agent complex system where the robot to human ratio is high. This eliminates the possibility of controlling the robots using very low-level action primitives (i.e. ‘point camera to bearing theta’) and instead, the human-robot interface must depend on a higher-level behavioral specification. Furthermore, each agent may have access to only information about its local environment. Reasoning about such a system must account for the uncertainties that arise from this constraint.

The RoboFlag game is the test bed for our work\textsuperscript{3}. In this environment, two teams of six robots play Capture the Flag according to most of the common rules. Other features such as available fuel and obstacles (which deactivate the robots when touched) were added as well in order to provide a richer, more dynamic environment. Each robot has access to a limited circle of vision that provides information about nearby robots and obstacles. The robots can also receive messages from one another and from the human controller subject to limited bandwidth and packet-loss. Vision, GPS, and messaging
account for all that the robot knows about the environment. Similarly each has two means of responding in the environment: moving with \([x,y]\) velocity and sending a message to another robot or human controller. Of course one can build a higher level representation of the environment and, similarly, provide a higher level set of action primitives.

In designing a system for human-robot interaction, we viewed the problem in terms of five different components, which are illustrated below. The following sections discuss each component separately. Section 2 describes the behavioral primitives that form the entity side intelligence, and how these behavioral primitives are interfaced to some authoritative piece of software (the Operator Module). Section 3 describes the Operator Module. This includes discussion of high level feature detection and a general architecture for integrating information and human control. Section 4 describes the graphical interface to the operator. Issues such as user underload/overload and multi-sensory feedback are discussed. Section 5 describes how human intelligence and strategy were developed.

2. Entity Side Intelligence

The primary method for interacting with the robots is to tell them which behavior to activate. Certain modules were provided, including: simple trajectory generation, obstacle avoidance, friend/foe detection, and robot tracking. The following are the behaviors that were implemented:

1. **Shield**: in this behavior, the robot is responsible for shielding a teammate from any approaching enemies. The robot simply looks for the closest enemy within a certain radius of the robot, \(R\), it is assigned to protect. If an enemy is found then it positions itself between the enemy and \(R\).
2. **Follow**: Simply follows a specified teammate.
3. **Scout**: The robot is assigned a region in which to move about randomly. This behavior helps compensate for the locality constraint.
4. **GoToFlag**: Very simple: when the robot has the flag, it returns to base. When it does not, it goes to the flag.
5. **Drive**: This provided a way for the human operator to manual drive the robot around with the arrow keys. This behavior circumvented trajectory generation, allowing control of different components of the robot’s velocity. This did not prove very useful because of latency issues.
6. **Defense**: Maintain some specified position, intercepting any taggable enemy in vision.
7. **Move**: Go to a specified destination and stop. This represented the lowest level of control.

Behaviors were defined according to a very simple abstraction: a behavior must define a message handling mechanism, an environment to action mapping, and some means of initializing the behavior’s parameters from a message (where messages are simply a collection of bytes that can be sent between entities). The first and third requirements...
provide the human (and other entities) with a means of remotely interacting with the behavior, albeit a very general ‘means’. For instance, the move behavior’s message handler allows for the operator to specify a new destination. Similarly, follow allows an operator to change the ID of the robot to follow.

It is important to note that the provided modules (e.g. obstacle avoidance) were not entirely reliable and this error propagated up into the higher-level behaviors described above. This became very apparent in actual game-play as it was often necessary to subsume control of the robot and redirect them towards a specific destination. Situations such as this would typically arise from anomalies in the obstacle avoidance algorithm. Also, interception algorithms would occasionally fail and similarly necessitate human intervention. For reasons such as these, very simple behaviors like defense, goToFlag, and move were the most frequently activated behaviors.

Built into the RoboFlag system is a simple messaging system that allows the operator and all the robots to communicate with one another. A message is simple an array of bytes. The first four bytes are interpreted as the Message_Type. There is no constraint on how the remaining bytes, the MessageBody, are to be interpreted, which means that one could potentially design a very high level grammar for interpreting these bytes. But this proved not to be necessary.

The system used for the final competition relied on messaging to simply pass small amounts of data between entities and operators. Each message type had one associated data structure that, when a message of that type was received, the MessageBody could be interpreted. For instance, a message of type Destination would indicate the first $f$ bytes represented an x-coordinate and the second $f$ bytes represented a y-coordinate, where $f$ is the number of bytes needed to represent a float. Messages are passed to the active behavior’s message handler and an appropriate response is generated. Also assumed is a top-level message handler which handles BehaviorChange messages in order that behavioral transitions can be supplied by the operator.

3. Operator-Side Intelligence

The client-side intelligence is a system that resembles an overarching automated ‘coach’ unit, which coordinates the group behaviors of sets of robots. By analyzing data collected from each robot entity, this ‘coach’ artificial intelligence is responsible for relaying commands between the human interface, prioritizing various strategic options, and executing the best plays or commands to the individual robot entities. As a result, the actions and knowledge of the players on the field are influenced by the commands from what is agreed to be a higher authority.

This client intelligence system utilizes the blackboard architecture\[6\] upon which various analytical and control modules are built. These modules can be assigned different and smaller goals that contribute towards solving the overall problem. The fundamental concept behind the blackboard architecture is that there is a place where information is gathered and shared by different users.\[7\] Each of these small groups of agents, while working on their own analysis, maintains full access to all areas of the blackboard. Various modules can run in parallel, making this setup a very flexible, dynamic, extensible architecture for development.

The implementation of this client-side intelligence consists of a variety of modules (agents of the blackboard) that are responsible for information gathering, global
data analysis, and task distribution. Information gathering modules involve collecting the local data received from each robot entity, and performing primitive analysis on such data, such as tracking, and velocity calculations. These modules present this raw data in a usable format for the other modules connected to the architecture.

The global data analysis modules form the core of the artificial intelligence system, whereby the primitive data are used to generate higher-level reasoning and conclusions. One such module creates an influence map, which uses positional and velocity information to calculate an influence of each object exerts on a map of the field. Friendly robots exert a positive influence on the map, while obstacles and enemy robots exert a negative influence. By performing gradient analysis on the influence map, enemy robots can be assigned into groups, and areas of the field can be classified as under friendly or enemy influence. These higher-level observations are then used to generate tasks and environmental information for the individual robots. For example, in our implementation, the influence map analysis determines whether a group of enemy robots poses as an offensive threat. This information is used to generate attack alerts and tasks, with which the client intelligence will allocate available robots to intercept such attackers, if possible. The final allocation of such tasks will send an assigned robot into a robot-level attack behavior.

Each higher-level behavior (aside from the simple behaviors that are available on the robot entities themselves) is defined in terms of a series of tasks. A task is something that performs some behavior and has an associated cost function. The cost function is used to prioritize the various tasks. Given that at any moment in time, there may be more tasks than available robots, or more robots than available tasks, a robust algorithm of task distribution is required. Our current implementation uses the Stable Marriage Algorithm, which is a technique that ensures the elements of two disjoint lists (tasks and robots in this case), are paired up in a stable fashion [7]. This allows us to assign the best available robot to accomplish the higher prioritized tasks.

During development, the client-side intelligence was plagued by numerous problems, which rendered the system to be less effective. Such issues involved latency within the system and error propagation in the behaviors of the entities themselves. In a real-time dynamic system such as RoboFlag, delays in execution of commands due to latency often led to the failure of an assigned task.

4. Client-to-Human Interface

We attempted to tackle the problem of multiple robot control by establishing variable levels of artificial intelligence and incorporating human overseers to guide overall robot actions. Using this method, we hoped to be able to incorporate both the benefits of artificial intelligence (controlling large numbers of robots simultaneously) and humans (observing overall strategies and plays based on field positions). The way that the human controlled the robots was governed by a behavior paradigm that involved the selection of behaviors that in turn dictate the actions that the robot should take. Either the artificial intelligence or the human can do this kind of behavior selection. Therefore, the level of control divided between the AI and the human is completely flexible within this architecture.

One important aspect of this information network is the user interface. It is crucial that all (and only) pertinent information be displayed in a way that is easy to understand,
avoiding both operator overload and underload. The client program must display a large amount of information, including the states of all the friendly robots, their fuel level, the behaviors to select from, and of course, the field. In addition to this basic visual display, a warning system existed to attract the attention of the human to important events. For example, when a new opposing robot enters into view, that part of the screen flashes, drawing the attention of the human to that area. It is a common problem that while controlling large amounts of robotic entities, the operator is faced with too much information being displayed at one time. Therefore, to lower the visual load, a system of aural alerts was implemented so that the human will hear a locating sound effect accompanying the visual flash. This warning system plays in 3d-space relative to the location of the alert on the field. Therefore, when the users hear a warning beep in the right headphone, they know to look towards that part of the field for an incoming enemy. However, in the system that we were using, the size of the field was not large enough to necessitate this kind of alert, the presence of aural effects was actually more distracting for the operator. Most games were played without sound but perhaps with more training this method of information presentation would have become more useful.

Interfacing humans to multiple agents has been likened to various real-time strategy games\cite{9}. Some of the more popular titles that have attempted to tackle this problem are the Warcraft series by Blizzard\cite{1} and the Command and Conquer series by Westwood Entertainment\cite{2}. In these games, users are required to operate large numbers of individual troops in a way that produces the most desirable cooperative behaviors. In addition, these games display the field information in the same top-down style as the Roboflag simulator. Due to these similarities, many of the nuances of control that the games implement in their user interfaces were included in the Roboflag interface. For example, visible regions of the field are drawn in a lighter color than those areas that are out of sight of your robots.

Also, the selection and movement of the robot entities are very similar to those used in the game’s UIs. Dragging the left mouse button over robots will select them, and then clicking the right mouse button assigns a destination for them to move to. Behavior selection is in the form of a list box on the side of the field display that when clicked on will send the chosen behavior to all of the robots currently selected.

In addition to this method, the most commonly used behaviors have hotkeys attached to them to speed up the human’s ability to control units, and these were used almost exclusively during the final tournament.
5. Human-Side Intelligence

Two important modules were used for human-side strategy development: logging and playback. These modules connected to the AI and could access the data on the blackboard. Simply put, the logging module writes field information to a file and the playback module re-presents the data from the file. This playback module has features that let the user fast forward or rewind to different parts of the log and view segments at various speeds. Over the course of the project this feature was incredibly helpful in the development of strategies for our team. Because of the complexity of the game, many times there would be too many things happening during the course of the games for the humans to follow, so problems would occur unnoticed by the operators. With the playback feature, the users were able to sift through the games and realize where mistakes were being made. For example, in the matches leading up to the final competition, 3 or 4 robots would routinely go inactive (get tagged by an obstacle) early on in the match. Using the playback, we discovered that this was caused by the faulty stationary obstacle avoidance code. So during the final matches, special attention was paid to robots undergoing stationary avoidance, and this was one of the main reasons that we were able to win. It was also determined through analyzing plays that it would be beneficial to leave a re-fueled man in reserve back in our home base. This strategy worked very well, and helped our refueling situation greatly. This playback and logging system was one of our greater advantages over the opposing team.

Conclusion

The development of our system was very much focused on actual game play. It became clear fairly early that developing a robust, natural user interface was much more critical to the system’s success in actual competition than the development of robust artificial intelligence. Tightly coupled with this was a need for operational skill and training.

References

5. airforce