Evolving Cooperative Control on Sparsely Distributed Tasks for UAV Teams Without Global Communication

Gregory J. Barlow   Choong K. Oh*   Stephen F. Smith

CMU-RI-TR-07-24

July 2007

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

*Choong K. Oh is with the U.S. Naval Research Laboratory, 4555 Overlook Avenue, SW, Washington, DC 20375

Abstract

For some tasks, the use of more than one robot may improve the speed, reliability, or flexibility of completion, but many other tasks can be completed only by multiple robots. This paper investigates controller design using multi-objective genetic programming for a multi-robot system to solve a highly constrained problem, where multiple unmanned aerial vehicles (UAVs) must monitor targets spread sparsely throughout a large area. UAVs have a small communication range, sensor information is limited and noisy, monitoring a target takes an indefinite amount of time, and evolved controllers must continue to perform well even as the number of UAVs and targets changes. An evolved task selection controller dynamically chooses a target for the UAV based on sensor information and communication. Controllers evolved using several communication schemes were compared on problem scenarios of varying size, and the results suggest that this approach can evolve effective controllers if communication is limited to the nearest other UAV.

Contents

1 Introduction

As the application of robotic systems to real-world problems increases, the use of multi-robot systems becomes more attractive. When a task can be completed more quickly by multiple agents than by a single agent, multi-robot systems can improve the speed and flexibility of task completion over single robot systems. More importantly, many tasks can only be solved by multiple robots. Some tasks, like lifting a large object, might require multiple robots working together. Other tasks might have physical or temporal constraints, such as requiring that two tasks be done simultaneously in different locations. Multi-robot systems range from centralized approaches to fully distributed approaches, with many approaches, like market-based coordination, falling somewhere in-between [1].

This paper investigates the design of a layered reactive controller for a system of multiple unmanned aerial vehicles (UAVs). Target radars are spread throughout a large area, and each task in the problem requires the proximity of at least one UAV and takes an indefinite amount of time, so the problem is one of task allocation. There are multiple types of radars, there is no prior knowledge about radars, and some radars can move, so task allocation must be dynamic. The number of UAVs and radars is not known a priori, so controllers must be adaptable. The UAV communication range is much smaller than the size of the environment, and sensor information about the radars is limited and noisy, making this a difficult problem to solve. In this paper, we use genetic programming to evolve high level controllers for task allocation.

Genetic programming (GP) [2] is a method of automated program creation using evolutionary computation. Given a measure of performance on a problem—a fitness function—evolution uses operators like crossover and mutation to create new solutions. Since more fit solutions are chosen with a higher likelihood, solutions tend to improve over time. GP creates solutions in the form of computer programs. Evolutionary techniques, including GP, are increasingly used in real-world applications, often producing results competitive with the best human efforts [3]. Evolutionary robotics [4], the application of evolutionary computation to robot applications, has yielded encouraging results in the design of robot controllers. For many evolutionary robotics problems, evolution is often able to create solutions a human would not have considered in order to find optimal or near-optimal solutions. Because of the difficulty of the problem considered here, evolving a controller using GP is an attractive alternative to designing a controller by hand.

Most multi-robot systems are hand-designed, but work has been done on evolving controllers for multi-robot systems. A popular multi-agent problem in the evolutionary computation literature is the predator-prey problem [5]. Controllers have been evolved using GP [6, 7], neural networks [8], and finite state machines [9]. In domains like the predator-prey problem, the numbers of agents and targets are often fixed, so one can use named sensing [7], where an agent communicates with a remote agent through a named channel specific to the remote agent. If the number of agents is not fixed, this is no longer feasible. For example, there may be a shortage of channels if the number of agents is larger than was expected when designing the system. An alternative is non-symbolic sensor-based communication [10, 11] where coordination is done through light or distance sensors. While this works well in some domains, an agent cannot easily share state information with this approach. Another alternative to named sensing is deictic sensing [7]. In this approach, communication channels are relative to the agent—e.g. *nearest agent*.

Experiments in the literature often assume global communication, where one agent can communicate with any other agent. This may significantly simplify the problem, but in many cases the assumption is not valid. The power to transmit over long distances may be beyond the capabilities of some robots, or the weight or size of long-distance communication equipment might be too great for some robots. While some researchers have addressed this problem in part by only communicating with nearest neighbors [7, 12] or using non-symbolic sensor-based communication [10, 11], most of the work using evolved controllers has ignored these limitations, either by assuming a small area of operation or the availability of long-distance communication.

Some recent results have investigated more realistic multi-robot applications. Richards et al. [12] evolved GP controllers for multi-UAV collaborative search. Communication took place between nearest neighbors, so the size of UAV teams and search area could scale. However, the search area to be swept is known a priori, and global communication is assumed. Agogino and Tumer [10] evolved neural network controllers for a multi-rover task similar to the one considered here. A heterogeneous team of rovers tries to observe points of interest of different values within the environment. Points of interest were distributed relatively densely, and there were typically more points of interest than rovers, making the problem easier. Communication was sensor-based, global communication was assumed, the points of interest had fixed locations, and all sensors were noise-free. Tumer and Agogino [13] extended this work by adding sensor noise, allowing points of interest to move, and limiting the rovers to local communication.

2 Problem

In this paper, we look at a multi-robot domain that can be posed as a distributed task allocation problem. The robots are unmanned aerial vehicles (UAVs) operating in a large environment. UAVs have a limited communication radius and a limited time in the environment (mission time). The environment contains target radars, with a one-to-one correspondence between the number of radars and the number of tasks. Each task requires a UAV to perform some action on the radar, such as surveillance or jamming, which requires proximity to the radar and takes an indefinite length of time. Since the particular action taken by the UAV is independent of the problem of assigning UAVs to radars, we will refer to performing the chosen action—and being close enough to the radar to do so—as monitoring the radar. Each radar can be monitored by a single UAV, but it may be possible to improve performance by assigning multiple UAVs to monitor the same radar. Unlike tasks that can be accomplished by finite length visits to a location, such as instances of the multi-depot traveling salesman problem [14], we can see tasks in this problem as taking infinite time to solve.

UAVs sense two pieces of information about the incoming signal from each radar: the amplitude and the angle of arrival (AoA). The AoA measures the relative angle between the heading of the UAV and the source of incoming electromagnetic energy. This model assumes an electronic support measures (ESM) sensor capable of splitting all incoming electromagnetic energy into signals by radar and maintaining a history of this information, a valid assumption given the capabilities of current commercial offerings. In addition to the current sensory information, the UAV stores amplitude values for a fixed time window; the slope of these historical values is available to the UAV controller. Real sensors do not have perfect accuracy in detecting radar signals, so the simulation models an inaccurate sensor. Both the amplitude noise and AoA accuracy can be set in the simulation; in this research, controllers evolved with amplitude noise of $\pm 6dB$ and an AoA accuracy of $\pm 10°$. A radar is invisible when it is not emitting. A target radar may be classified using two attributes: when it is deployed and its mobility. In our simulations, radars fall into three distinct types: stationary, delayed, and mobile. Stationary radars have a fixed location and are deployed for the duration of the mission. Delayed radars also have a fixed location, but are not deployed until after the mission has begun. Mobile radars are also delayed, but change location several times during the course of the mission. Mobile radars do not emit while moving. If all radars are stationary, then this problem can be solved optimally prior to the mission using a centralized approach since all relevant information is known a priori. This becomes a distributed problem when delayed and mobile radars are present. All types of radars can emit either continuously, where the radar signal is constant while the radar is deployed, or intermittently, where the radar signal emits for some duration periodically. Radar locations are random and are not known a priori.

From the problem outline, it should be clear that to solve this problem, multiple UAVs are necessary. Since radar positions are not known a priori and UAVs have small communication ranges, this is a distributed task allocation problem. A single UAV can be assigned to only one radar at a time, so we need at least as many UAVs as there are radars for an optimal solution. An ideal solution to this problem would be able to dynamically assign UAVs to radars such that at least one UAV is monitoring every radar at all times. Since radars are distributed spatially, a given UAV must be within some distance of the radar to monitor it, making a perfectly ideal solution infeasible. We pose this as a maximization problem in the time each radar is monitored by at least one UAV. If all radars are of equal importance, performance can be measured as a sum of monitoring time for each radar. It is more likely, however, for radars to have different priorities. We prefer a solution that takes into account these priorities.

The constraints imposed by this problem make good controller design difficult. The sparseness of targets and short range of communication mean that UAVs have only small windows of time for communication and must make decisions with incomplete information. Because radars are long distances apart and monitoring a radar requires proximity to it, poor task allocation heavily degrades performance. Limited and noisy sensor information, radar movement, and the lack of a priori information about the number of UAVs and the number and type of radars all contribute toward making this a difficult problem of dynamic task allocation.

3 Approach

Our approach to this problem assumes a layered reactive controller. The control architecture of an individual UAV, shown in Figure 1, is divided into three layers. The *target selection controller*, the layer evolved in this work using GP, takes current sensor information, communication from other UAVs, and a small amount of internal state information as inputs and then outputs a target radar. The next layer, the *navigation controller*, takes as inputs the current sensor

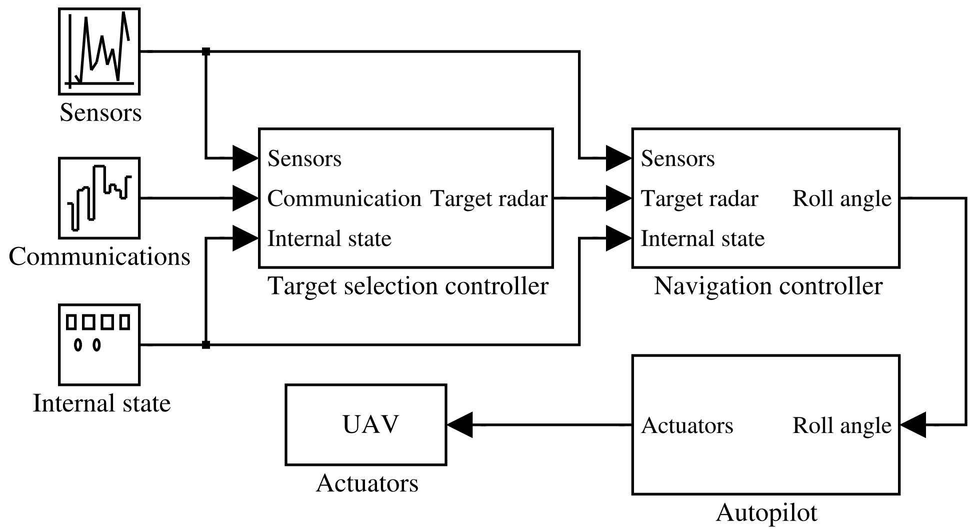

Figure 1: UAV controller architecture

information, the target radar from the target selection controller, and the current roll angle and outputs a desired roll angle. The navigation controller used in this work, described in [15–17], was also evolved using GP. The roll angle from the navigation layer is passed to the *autopilot* layer. The autopilot uses the desired roll angle to change the heading of the UAV. This layered technique results in a general controller model that can be applied to a wide variety of vehicle platforms; the evolved controllers are not designed for a specific UAV airframe or autopilot. The system is homogeneous; all UAVs use the same controller. For a specific scenario with a fixed number of UAVs and known radars, a heterogeneous system might perform better than a homogeneous system, since heterogeneity would allow for specialization, but in this problem, homogeneity allows easy variation in the number of UAVs and the number and types of target radars.

With limited local communication, a UAV only has the option of communicating with other UAVs in range. The number of UAVs in range changes, so one must have some scheme to decide how communication from a variable number of agents will be amalgamated. We investigate three communication schemes which fit the representation and controller structure: communication only with the closest other UAV; communication with all UAVs in range, where all communication is weighted equally; and communication with all UAVs in range, where the closer another UAV, the more heavily weighted the communication. The first communication scheme, *closest*, uses only communication from the nearest UAV in range. The genetic program is run once, using communication from the nearest UAV, and the output is set as the radar to track. The second scheme, *majority*, weighs communication from all UAVs in range equally. The genetic program is run once for each UAV in communication range, and the most common output is chosen as the radar to track (ties are broken arbitrarily). The third scheme, *weighted*, weighs communication from all UAVs in range by distance. The genetic program is run once for each UAV in communication range, and the output from each execution is weighted by the distance to the remote UAV, where closer UAVs have higher weights. The radar with the highest weighted sum is tracked.

We chose this approach based on the qualities of the problem, which required a solution that used small amounts of local communication, was scalable to larger groups of UAVs and target radars, and was flexible to different types of radars. Our approach does not require any high level world knowledge, using only a limited set of sensors and small amounts of local communication. Computation is completely distributed, allowing a single UAV to operate independent of other UAVs when necessary. While we used a fixed communication range of 5 nautical miles, this approach would work for other communication ranges. It is important to note, however, that if global communication is available, the performance of our approach will not be as good as a centralized or market-based approach.

Table 1: Functions

Function	Arity	Description
If{Same,Diff}	4	If the first two arguments are the same/different, returns the third argument, else returns the fourth argument
IfUAVsInCommRange	2	If at least one other UAV is in communication range, returns the first argument, else returns the second argument
IfTracking	2	If the UAV is tracking a radar, returns the first argument, else returns the second argument
IfPosition{N,S,E,W}	2	Returns the first argument if the UAV is further in the given cardinal direction than the remote UAV, else returns the second argument
IfHeading{N,S,E,W}	2	Returns the first argument if the UAV's heading is closer to the given cardinal direction than the remote UAV's heading, else returns the second argument
{Local,Remote}AoA{Smaller,Larger}	2	Returns the radar with the smaller/larger angle of arrival
{Local,Remote}AoA{Left,Right}	2	Returns the radar with the angle of arrival further to the left/right
{Local,Remote}Amp{Smaller,Larger}	2	Returns the radar with the smaller/larger amplitude
{Local,Remote}Slope{Smaller,Larger}	2	Returns the radar with the smaller/larger slope
Priority{Smaller,Larger}	2	Returns the radar with the smaller/larger priority

4 Genetic Programming

Genetic programming is a method of automated programming that uses a genetic or evolutionary algorithm [2]. Starting from a measure of performance for a particular problem—a fitness function—GP creates a computer program to solve the problem. Like a genetic algorithm, a population of random solutions is generated, and each individual in the population is evaluated for fitness. Individuals are selected based on fitness to create new members of the population using genetic operations like crossover and mutation. Since individuals with higher fitness are more likely to be selected, the fitness of the population tends to improve toward optimal solutions over successive generations. In GP, each individual is a computer program, which can be represented as a tree or a symbolic expression similar to Lisp. Programs are composed of functions and terminals from a defined set of operations.

An evolved target selection controller takes as inputs local sensor information and information communicated to it by other UAVs and outputs a radar for the navigation controller to track. The choice of appropriate functions and terminals is essential to the success of GP-based solutions. Initially, we experimented with evolving navigation controllers with functions and terminals on the space of sensor values where control actions were side effects of the GP operators, similar to the operators in [16, 17] with added operations for communication. These representations that directly used sensor values and attempted to evolve both target selection and navigation performed poorly; one reason was that it is not straightforward how to combine information from multiple UAVs given no a priori information about the number of UAVs or the number of radars. Rather than operate on the space of sensor values and use side effects for control, our approach operates on the space of radars, where all argument and return types are radar identification numbers. At each time step, the UAV tracks the radar that is the output of the genetic program. To allow for a variable number of radars, the representation is deictic. Deixis is a process where expressions rely on context. In this representation, the output of functions and terminals depends on the position and orientation of the UAV. For example, depending on the orientation and location of the UAV, the terminal that outputs the radar with the smallest AoA could represent any radar.

The functions and terminals used by GP are shown in Tables 1 and 2. Many of the operators in the table are split between sensing relative to the UAV executing the genetic program (local) and sensing relative to a UAV in communication range (remote). The communication scheme, chosen from the schemes described above, determines the remote UAV. If no other UAV is in range, the local UAV is set to be the remote UAV. In all operators, ties are broken arbitrarily. Figure 2 shows an example controller. In this controller, if the local UAV and the remote UAV are tracking the same radar, a new radar to track is chosen at random. Otherwise, if the UAV is tracking a radar, it continues to do

Table 2: Terminals

Terminal	Arity	Description
{Local,Remote}TrackingCurrent	0	Returns the radar currently being tracked
{Local,Remote}TrackingLast	0	Returns the radar tracked prior to the radar returned by TrackingCurrent
{Local,Remote}AoA{Smallest,Largest}	0	Returns the radar with the smallest/largest angle of arrival
{Local,Remote}AoA{Left,Right}most	0	Returns the radar that, based on a sweep of the angle of arrival, is the furthest left/right
{Local,Remote}AoA{N,S,E,W}most	0	Returns the radar with the angle of arrival closest to the given cardinal direction
{Local,Remote}Amp{Smallest,Largest}	0	Returns the radar with the smallest/largest amplitude
{Local,Remote}Slope{Smallest,Largest}	0	Returns the radar with the smallest/largest slope
Priority{Smallest,Largest}	0	Returns the radar with the smallest/largest priority
RandomRadar	0	Selects a radar at random to return

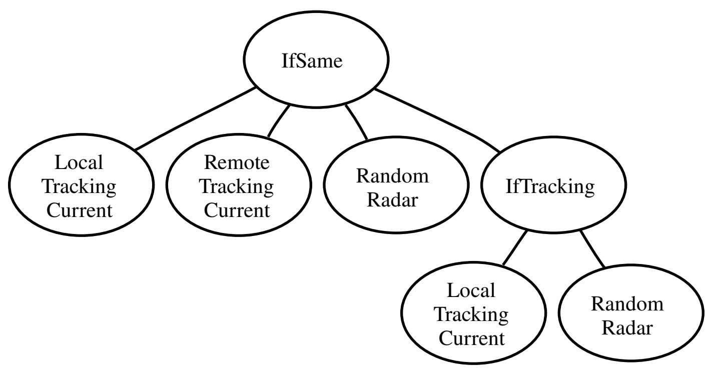

Figure 2: Example controller

so, else it chooses a radar to track at random.

Three fitness functions measure the performance of a group of UAVs. In order to measure performance, we make a distinction between the time a UAV spends tracking a radar (once assigned to a radar, the UAV moves toward the radar and begins to circle it) and the time the UAV spends close enough to monitor the radar (in this work, 2 nautical miles). The first fitness function, $f_{monitor}$, measures the percentage of the time the target radars are not monitored by at least one UAV. Designed to measure the performance of the UAV team on only the main goal of monitoring the target radars, this fitness function used alone tends to suffer from the bootstrap problem, and due to the noise in the system can lead to undesirable results. The second fitness function, f_{track}, alleviates the bootstrap problem by measuring the percentage of time that each radar is not tracked by at least one UAV. Often, when evolving robot controllers, the best controllers exhibit some unwanted behaviors that have only minor negative effects on fitness. In an optimal controller, evolution would weed out these behaviors, but when evolving controllers in a noisy environment, evolving a perfectly optimal controller is often not feasible. In this environment, one such behavior is a tendency for a pair of UAVs to repeatedly swap targets mid-flight. This has very little effect on $f_{monitor}$ and no effect on f_{track}, but the crossing zig-zag pattern this behavior creates could lead to collisions. To help eliminate this behavior, the third fitness function, f_{switch}, measures the percentage of time UAVs switch from tracking one radar in order to track another. These three fitness functions are measured over the course of each simulation. At time t, let $target^u_t$ be the target radar of UAV u, and let the binary variable $switched^u_r$ be true if $target^u_t \neq target^u_{t-1}$.

For evaluation, at time t, let d^r_t be the minimum distance from radar r to a UAV, x^r_t and y^r_t its location in space, and $priority^r_t$ its priority (the more important the target, the higher the priority value). For the following binary variables,

Table 3: Genetic programming parameters

Population Size	1000	Maximum Initial Depth	5
Crossover Rate	0.9	Maximum Depth	25
Mutation Rate	0.05	Generations	80
Tournament Size	2	Trials per Evaluation	20

let $deployed_t^r$ be true if radar r is emitting at time t, let $monitored_t^r$ be true if $d_t^r < range$ (where $range$ is the monitoring range of the UAV) and radar r is emitting at time t, let $moved_t^r$ be true if $\left(x_t^r \neq x_{t-1}^r\right) \cup \left(y_t^r \neq y_{t-1}^r\right)$, and let $tracked_t^r$ be true if $\exists u$ such that $target_t^u = r$ at time t. For each binary variable, let $t_{variable}^r$ be its sum over time. For example

$$t_{deployed}^r = \sum_{t=1}^{T} deployed_t^r \tag{1}$$

The $moved$ variable is used to calculate the travel time to a radar as distance — initial distance plus distance increments when the radar moves — divided by UAV velocity. The variable t_{travel}^r eliminates the bias from different travel times to each radar.

$$t_{travel}^r = \frac{d_0^r + \sum_{t=1}^{T} d_t^r \cdot moved_t^r}{v} \tag{2}$$

The first fitness function is the average over all radars of the percentage of time that each radar is unmonitored weighted by the radar priority.

$$f_{monitor} = \frac{1}{R} \sum_{r=1}^{R} priority^r \frac{t_{deployed}^r - t_{travel}^r - t_{monitored}^r}{t_{deployed}^r - t_{travel}^r} \tag{3}$$

The second fitness function is the average over all radars of the percentage of time that each radar is not being tracked.

$$f_{track} = \frac{1}{R} \sum_{r=1}^{R} \frac{t_{deployed}^r - t_{tracked}^r}{t_{deployed}^r} \tag{4}$$

The third fitness function is the average over all UAVs of the percentage of time that each UAV switches targets.

$$f_{switch} = \frac{1}{U} \sum_{u=1}^{U} \frac{t_{switched}^r}{T} \tag{5}$$

The genetic programming system attempts to minimize all three fitness functions.

Target selection controllers were evolved using multi-objective GP with non-dominated sorting, crowding distance assignment to each solution, and elitism using an implementation of NSGA-II [18] for GP. Evolution was generational, with crossover and mutation similar to those outlined in [2]. The parameters used by GP to evolve controllers are shown in Table 3. Tournament selection was used. Initial trees were randomly generated using ramped half and half initialization. All computation was done on a Beowulf cluster parallel computer with ninety-two 2.4 GHz Pentium 4 processors.

5 Experiments

We evolved controllers with a single, general scenario—five UAVs and four radars—for all three communication schemes: *closest*, *majority*, and *weighted*. We performed ten evolutionary runs for each scheme, and for each evaluation, all UAVs used the same controller and communication scheme. Two radars were stationary: one with normal priority, the other with high priority. The third radar was a delayed radar with normal priority, and the fourth radar was a mobile radar with high priority. All four radars emitted intermittently for a normally distributed random duration with a mean of 5 minutes and a normally distributed random period with a mean of 10 minutes. All controllers were evolved using the fitness functions and GP parameters outlined in Section 3. While f_{track} and f_{switch} were important fitness functions for overcoming the bootstrap problem and controlling behavior, $f_{monitor}$ is a true measure of the fitness of a controller, so we chose the best controller for each communication scheme from 10 evolutionary runs using only this fitness function.

To evaluate this approach, we compared the best controllers from each communication scheme over a variety of scenarios. Since an exhaustive study of all possible scenarios was not feasible, we selected realistic scenarios for the simulation area of forty nautical miles by forty nautical miles. We evaluated these controllers on four scenarios: three UAVs and three radars, where all radars were stationary and emitted continuously (this scenario has a perfect solution when radar assignments are determined a priori); five UAVs and four radars, where the radars are the same types as used to evolve the controllers; ten UAVs and four radars, with the same radar types as before; and ten UAVs and eight radars, with the same radar types as before, just twice as many of each. The density of radars and UAVs was the most important consideration in choosing a realistic scenario, since the problem of five UAVs and four radars in 1600 square nautical miles is effectively the same at fifty UAVs and forty radars in ten times the area, as long as the radars are distributed randomly.

When information about all radars is not known a priori, a centralized approach produces poor solutions; since the communication range of a UAV is much smaller than the area where the tasks are distributed, a situation where all UAVs could communicate with one another occurs infrequently. On the other end of the spectrum, a fully distributed approach with no communication would also tend to produce poor solutions. Under certain initial configurations of UAVs and radars, it is possible to perform well without communication, but in general, communication is necessary in order to get the best distribution of UAVs to radars. In recent literature, multi-depot traveling salesman problems using real robots have been successfully solved with market-based approaches [1, 14]. These approaches benefit from free and global communication and knowledge of the environment for use in planning paths and estimating bids on tasks. While it would be possible to use a market-based approach on this problem, the small communication range and lack of knowledge about radar locations for planning purposes would make it difficult to achieve good performance.

Since the specific characteristics of this problem made these competing controller methodologies unsuitable, we compared the evolved controllers to a baseline randomized controller. In the *random* controller, each UAV initially chooses a radar to track uniformly from all known radars (initially, only stationary radars). At each time step, a UAV polls all other UAVs in range to see which ones are tracking the same radar; let n be the number of UAVs tracking this radar. The UAV knows the number of other UAVs, u, and the number of deployed radars, r. Ideally, the number of UAVs monitoring each radar is $\frac{u}{r}$. At each time step, if $n > \frac{u}{r}$, then the UAV picks a new radar to track randomly with probability $\frac{n-\frac{u}{r}}{n}$, since the number of UAVs tracking this radar that should be tracking other radars is $n - \frac{u}{r}$. This controller performs reasonably well given that selection of which radar to track is random.

Each combination of communication scheme and scenario was evaluated 1000 times, where radar positions were random for each evaluation. Figure 3 shows the average percentage of time that a radar is unmonitored (with 95% confidence intervals) for the *closest*, *majority*, *weighted*, and *random* communication schemes on each of the four scenarios. This measure is equivalent to $f_{monitor}$ without weighting for radar priority. In all scenarios, the *closest* controller performed best. The *majority* and *weighted* controllers had similar performance, but always performed worse than the *random* controller. The best *closest* controller also required very little communication, as not all communication functions and terminals appear in the evolved program. Of the 43 nodes in the program tree, the only communication operations were the *RemoteAoALargest*, *RemoteTrackingCurrent*, and *RemoteTrackingLast* terminals, several functions requiring heading and position, and the *RemoteSlope*$\{$*Smaller, Larger*$\}$ functions, requiring the communication of only $r + 6$ variables, where r is the number of radars (the values of the three terminals, heading, latitude, longitude, and slope values for all radars). Interestingly, the *AoASmallest* terminal was not used at all by the best controller.

Why do the best *majority* and *weighted* controllers perform so poorly, when we might expect them to outperform the *closest* controllers? First, these controllers only have an advantage over the *closest* controllers when more than one other UAV is within communication range. Given the sparse distribution of radars in the environment, this rarely happens, and it happens most often at the beginning of a simulation, when tracking assignments are largely arbitrary. In these situations, communication can often be more confusing than advantageous, especially if it leads to constant switching between radars to track. A look at the genetic programs for some of the best controllers for each communication scheme reveal the source of this discrepancy in performance. The best *closest* controllers take the form shown in Figure 4a, while all of the best *majority* and *weighted* controllers take the form shown in Figure 4b where (...) represents sub-trees for choosing a new radar to track (which vary, and are too complex to show here in full). The *closest* controller first checks to see if the local and remote UAV are tracking the same radar, and if so, chooses a new radar (which might be the same radar), otherwise, it checks to see if the UAV is tracking a radar, and if so, continues

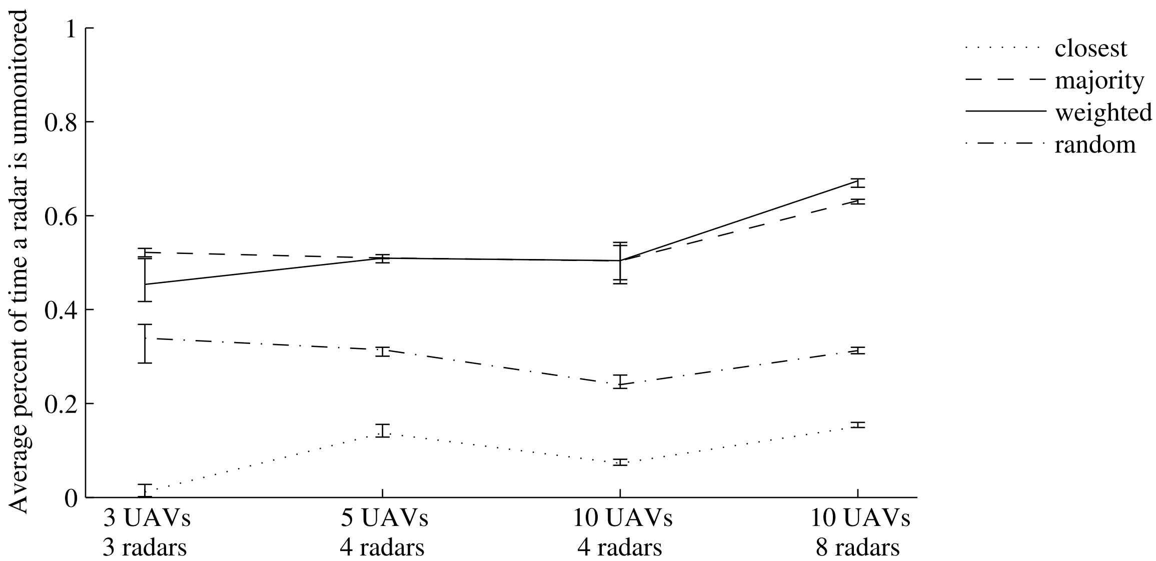

Figure 3: Average percentage of time that a radar is unmonitored (with 95% confidence intervals) for *closest*, *majority*, *weighted*, and *random* communication schemes.

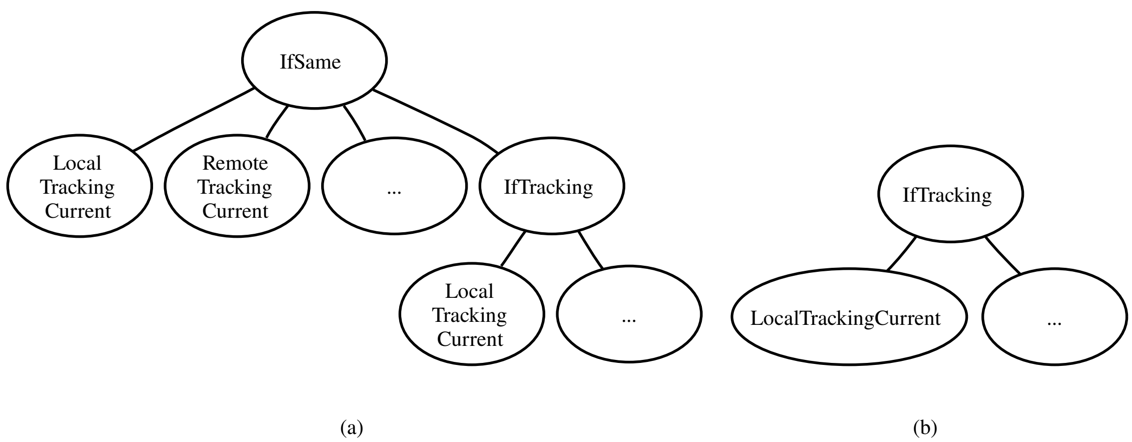

Figure 4: Evolved controller structures

to track that radar, otherwise, it chooses a new radar. This structure allows the UAV to choose a new radar to track in the case of redundancy, something that is necessary for tracking delayed and mobile radars, since these radars are not immediately visible. The best *majority* and *weighted* controllers evolved much less complex structures which only allow the UAV to track stationary radars. Evolution stalls on this simple structure, and seems unable to jump to a more complex structure, as it was able to do with the *closest* controllers. One reason for this may be that avoiding confusion by using this simpler structure brought higher fitness than using a more complex structure. It is also possible that the combination of the representation and these communication schemes is not conducive to evolving good controllers.

6 Conclusions

Based on these experiments, this approach can evolve effective controllers if communication is restricted to the closest other UAV in range. For UAVs controlled by the best *closest* controller, the average percentage of time spent unable to monitor radars was never worse than 16%, while the percentage for the *random* controller, the second best controller on all scenarios, was never better than 25%. Given the limited capabilities of a multi-robot system with such small windows of opportunity for collaboration, these results suggest that our approach using the *closest* communication scheme is a good solution to the problem.

Evolved controllers performed well for a variety of scenarios, responding well with changes in the number of UAVs, number of radars, and types of radars. For this particular class of multi-robot problem, where tasks have indefinite length, only local communication is available, and tasks are distributed sparsely throughout the environment, this work successfully evolved GP controllers with good fitness that require very little communication bandwidth. While this approach is tailored to this type of problem, and would not be suitable for all multi-robot problems, we feel this approach could be successful for other problems of this type.

Acknowledgments

Gregory J. Barlow is supported by a National Defense Science and Engineering Graduate fellowship. The U.S. Naval Research Laboratory (Code 5730) provided computation time on their Beowulf cluster for this work.

References

[1] M. Bernadine Dias and Anthony Stentz. A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2279–2284, 2003.

[2] John Koza. *Genetic Programming*. MIT Press, 1992.

[3] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[4] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics*. MIT Press, Cambridge, MA, 2000.

[5] M. Benda, B. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA, August 1986.

[6] Thomas Haynes, Sandip Sen, Dale Schoenefeld, and Roger Wainwright. Evolving a team. In *Working Notes of the AAAI-95 Fall Symposium on Genetic Programming*, pages 23–30. AAAI Press, 1995.

[7] Sean Luke and Lee Spector. Evolving teamwork and coordination with genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 150–156, Stanford University, CA, July 1996. MIT Press.

[8] Chern Han Yong and Risto Miikkulainen. Cooperative coevolution of multi-agent systems. Technical Report AI-01-287, The University of Texas at Austin Department of Computer Sciences, 2001.

[9] Kam-Chuen Jim and C. Lee Giles. Talking helps: Evolving communicating agents for the predator-prey pursuit problem. *Artificial Life*, 6(3):237–254, 2000.

[10] Adrian Agogino and Kagan Tumer. Efficient evaluation functions for multi-rover systems. In *Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, pages 1–11, Seattle, WA, 2004.

[11] Matt Quinn. Evolving communication without dedicated communication channels. In *Proceedings of the European Conference on Artificial Life*, pages 357–366, Prague, September 2001.

[12] Marc D. Richards, Darrell Whitley, J. Ross Beveridge, Todd Mytkowicz, Duong Nguyen, and David Rome. Evolving cooperative strategies for UAV teams. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.

[13] Kagan Tumer and Adrian Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, pages 591–598, Washington, DC, 2005.

[14] Robert Zlot and Anthony Stentz. Market-based multirobot coordination using task abstraction. In *Proceedings of the International Conference on Field and Service Robotics*, 2003.

[15] Gregory J. Barlow and Choong K. Oh. Robustness analysis of genetic programming controllers for unmanned aerial vehicles. In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, Seattle, WA, July 2006.

[16] Gregory J. Barlow, Choong K. Oh, and Edward Grant. Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems*, Singapore, December 2004.

[17] Choong K. Oh and Gregory J. Barlow. Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, pages 1538–1545, Portland, OR, June 2004.

[18] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.