Improving the Expected Performance of Self-Organization in a Collective Adaptive System of Drones using Stochastic Multiplayer Games

Ian Riley
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, 74104, USA
ian-riley@utulsa.edu

Brett McKinney
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
brett-mckinney@utulsa.edu

Rose F. Gamble
Tandy School of Computer Science,
University of Tulsa,
Tulsa, Oklahoma, USA 74104
gamble@utulsa.edu

Abstract

The Internet-of-Things (IoT) domain will be one of the most important domains of research in the coming decades. Paradigms continue to emerge that can employ self-organization to capitalize on the sheer number and variety of devices in the market. In this paper, we combine the use of stochastic multiplayer games (SMGs) and negotiation within two collective adaptive systems of drones tasked with locating and surveilling intelligence caches. We assess the use of an ordinary least squares (OLS) regression model that is trained on the SMG's output. The SMG is augmented to incorporate the OLS model to evaluate integration configurations during negotiation. The augmented SMG is compared to the base SMG where drones always integrate. Our results show that the incorporation of the OLS model improves the expected performance of the drones while significantly reducing the number of failed surveillance tasks which result in the loss of drones.

1. Introduction

Paradigms continue to emerge [1,2] that capitalize on the unique benefits that Internet-of-Things (IoT) devices have to offer as well as the sheer number of devices available. One such paradigm is the use of swarm applications [2], which takes advantage of the number and relatively low manufacturing cost of IoT devices and drones. As the ability to implement collective systems improves, developers will be able to incorporate more complex functionality into these systems to achieve greater compliance with a pre-defined set of system requirements. These *collective adaptive systems* [3] can operate as a large swarm or as smaller, more specialized teams through the exercise of self-organization [2,4,5]. To permit dynamic team compositions, collective adaptive systems require the ability to define and evaluate a specification of self-organization, which we refer to as an *integration configuration*. Integration configurations specify the intended functionality that each member of the team will exercise, the inputs required for the intended functionality to be executed, and the expected performance of the intended functionality given its inputs. Devices can specify and agree upon an integration configuration via a mutually known negotiation protocol [5,6].

The use of automated, trust-based negotiation protocols [5,6,7] has been investigated for service composition [8] and mobile ad hoc networks (MANets) [9]. Some approaches [6] use static rules to establish trust between devices and then accept or reject integration depending on whether trust is established. This is inadequate when a device has other functional requirements that can be undermined by constraints imposed during negotiation and integration [3]. In prior work [5], we extend trust-based negotiation to devices [10] that can determine what conditions comply with or undermine their own requirements. We introduce the concept of *situational goals* as goals that can only be met via negotiation and integration.

In this paper, we extend prior work [3,5] to an intelligence, surveillance, and reconnaissance (ISR) scenario [12] involving two collective adaptive systems of drones. These systems must work together to recon a large map for intel caches and organize themselves into a two-drone team to attempt a surveillance task whenever an intel cache has been identified. To facilitate self-organization, we apply a negotiation protocol [5] to permit negotiation between collective systems via an intermediary. The negotiation protocol requires that each individual drone be capable of evaluating integration configurations at runtime. We demonstrate how stochastic modeling and regression can be combined to evaluate configurations at runtime.

A primary challenge to deploying IoT devices or drones within a safety-critical domain is that low power devices must accommodate foreseen and foreseeable sources of uncertainty [11] present in their application domain. This challenge can be met by incorporating stochastic modeling, which can model sources of uncertainty present in the system or environment as random variables [3]. We employ the use of a stochastic multiplayer game (SMG) [13,14,15] that models an individual drone and its environment as its players. SMGs are intended for use as a tool to evaluate behaviors and adaptations at design time.

An ordinary least squares (OLS) regression [16] model is trained on the output of the SMG for the ISR scenario. The trained model is utilized by the drones to evaluate integration configurations at runtime with little to no overhead. To demonstrate the efficacy of the OLS model, the SMG is augmented to use the OLS model to evaluate configurations during negotiation. The results demonstrate that the use of an OLS model to evaluate configurations improves a drone's expected performance with respect to their requirements while significantly reducing the expected number of failed surveillance tasks, and subsequently reducing the destruction of drones in the field and the loss of intel.

2. Background

The use of automated, trust-based negotiation has been investigated in service composition both through internet-based web-services [8] and in MANets [9]. Unlike internet-based webservices, services employed over a MANet or service-oriented ad hoc network [17] can experience periods of unavailability as devices lose power, suffer intolerable network conditions, or move away from one another. Due to these sources of uncertainty [11], as well as others that threaten the reliability of an integration over an ad hoc network, it is necessary to justify the application of an integration configuration given the requirements of all participating parties [3,17]. To that end, researchers have investigated assessing configurations based on pre-defined metrics [6,17].

Wang et al. [17] develop CATrust to establish the trustworthiness of a device based on context information gathered from the environment. Dragoni et al. [6] define the SxT negotiation framework to establish trust between devices based on pre-defined security policies. Both these authors consider domains where it is necessary to establish trust between devices because they are unknown to one another and may not perform reliably. Other authors have investigated the reliability of devices based on a pre-established history [18,19] or reputation [20]. In our scenario, the conditions that determine the reliability of a drone's involvement in an integration change frequently and must be consistently reassessed. As such, the history or reputation of a drone's performance is irrelevant.

In prior work [5], we extend the SxT negotiation framework by augmenting the defined message types to use assessment procedures rather than pre-defined security policies. For our current scenario, drones are deployed by the same organization but are threatened by adversarial elements that aim to destroy the drones. As such, drones must assess the efficacy of an integration configuration given the risk of being destroyed should they be challenged by adversaries. We seek to incorporate stochastic modeling [3,13,14,15] into the negotiation process to determine if a configuration should be permitted given both the risk of involvement in an integration and the efficacy of the integrated parties with respect to their task.

Camara et al. [14] propose the use of SMGs [15] to model safety-critical applications that possess sources of uncertainty. In prior work [3], we investigate the use of SMGs to evaluate the relationship between design constraints and system performance. By utilizing an SMG, we can evaluate an integration configuration with respect to a drone's expected performance while including goals that are negatively impacted by integration.

Similar efforts frame the evaluation of an integration as a multi-objective optimization problem [21]. These problems typically require the application of exponential algorithms. Although these algorithms could be employed at design time, there is no guarantee that the results would cover all possible instances of the scenario since the conditions necessary to solve the multi-objective optimization problem consistently change and their values may not be fully known. Another approach is to use heuristics to reduce the complexity of the solution to linear time [20]. However, in a safety-critical domain, heuristics must adequately reflect device requirements, and we must consider that the only viable complexity may be constant time.

Researchers have investigated the use of regression models to evaluate a device's performance [22,23]. Li et al. [22] apply an autoregression model to historical observations to produce a model of device performance over time. Venkataraman et al. [23] propose a regression model to learn the weights associated with device quality-of-service attributes, where each attribute is assessed individually using Bayesian inference. In this paper, we demonstrate how OLS regression can be applied to the output of an SMG to produce a model that can evaluate an integration configuration at runtime.

3. Defining the stochastic multiplayer game

In this section, we introduce the scenario and the drone requirements. We define an SMG based on the scenario, into which negotiation is incorporated as an action.

3.1. Scenario

We describe a safety-critical scenario [12] involving two collective adaptive systems of drones that participate in an ISR mission. Both systems are deployed by the same organization and are assumed to possess the means required to authenticate all drones and communicate between one another using encrypted channels. The possibility that adversaries might intercept drone communications, identify that a drone is communicating, or act as a malicious imposter is considered outside the scope of this paper. The intended use of negotiation is not to establish trust per se but to select an integration configuration that can be reliably met by the negotiating parties and to improve the expected performance of the drones with respect to the scenario's grading criteria.

3.1.1. Game specification. The target scenario is implemented in Java and has been made available to us through the RASSim Docker [12]. A collection of 200 drones has been tasked with conducting an ISR mission. The mission is to survey a target amount of intel (measured in intel units) within a one-hour period. The available intel units are partitioned across a set of 65 intel caches. Caches are stagnant, but 15 of the 65 caches will not be available at the start of the scenario. Rather, these 15 caches will randomly 'pop up' at some point during the mission. These 15 'pop up' caches are each worth 70 units. 25 of the other caches are worth 20 units and the remaining 25 caches are worth 10 units. To collect intel units from an intel cache, drones must work together in a specialized team to perform a surveillance task. The number and location of intel caches, the distribution of intel units, and the details of the 'pop up' caches are all unknown to the drones prior to the start of the mission.

The scenario also includes adversaries that protect the caches. Adversaries are mobile but must remain within a set distance of the cache they are protecting. Only adversaries can destroy drones. Drones must work in teams to both inhibit adversaries and survey intel simultaneously. Adversaries are slightly less capable than the drones. They have a smaller radius of vision, a smaller max speed, and are unable to attack drones while being inhibited. A drone can only inhibit one adversary at a time. As the number of adversaries at an intel cache increases, so too must the number of drones assigned to the surveillance task. This can be handled using multiple instances of negotiation that operate in parallel. For this paper, we use one adversary per intel cache. The number and distribution of adversaries are unknown to the drones.

Drones begin the scenario at homebase, which is located at just outside the corner of the map. The map is a 25x25 square grid that represents a 256 km^2 area. Prior to the scenario, the 200 drones are partitioned into surveillance drones and wingmen. Surveillance drones have an intel sensor that can survey intel at a rate of one intel unit per second. Only surveillance drones can survey intel. Wingmen have electronic countermeasures that can inhibit adversaries. At the start of the scenario, drones recon the map searching for intel caches. All drones have visual sensors to identify intel caches, measure the size of a cache in terms of intel units, and identify adversaries.

When a drone discovers an intel cache, they report the location and size of the cache to homebase. Unless destroyed, all drones can communicate with homebase and vice versa. After reporting an intel cache, a surveillance task is generated. Surveillance tasks require one surveillance drone to survey the intel and one wingman to inhibit the adversary. The drone that discovers the intel cache is the first team member. The second team member is assigned from the opposite type of drone scattered throughout the map. If a surveillance task is successful, then all the cache's intel units are collected, and the drones resume reconnaissance. Otherwise, both drones are destroyed by the adversary and their intel with them. The mission ends when the drones have surveyed the target number of intel units or when the mission timer expires. The mission timer is one hour, and we allocate 5 seconds per turn, so the total length of the mission is 720 turns.

In this paper, we use a partition of 100 surveillance drones and 100 wingmen. Each drone is assigned quadrants to recon. Quadrants are selected using a quad-tree construction defined in prior work [12] and are assigned to drones alternating between surveillance drones and wingmen. We ensure that every surveillance drone has at least one neighboring quadrant that contains a wingman and vice versa. It does not guarantee that a surveillance task will be completed by the closest drones but provides a consistent distribution of drones at a variety of distances throughout the map. This simplifies the process of modeling the distance required for a surveillance drone or wingman to reach the location of a surveillance task.

3.1.2. Drone requirements. Once the mission ends, drones are graded based on three criteria. Table I provides the relative weights and descriptions of the criteria. The relative weight is the maximum number of points awarded to the drones in each criterion, and the description states how points are awarded. According to the grading criteria, drones are awarded a maximum of 30 points based on the percentage of intel caches that they identify regardless of whether they survey its intel. They are awarded a maximum of 25 points based on the percentage of intel units that they survey, and a maximum of 15 points based on the percentage of drones that survive. Drones only survive if they are not destroyed by an adversary and return to homebase before the mission expires. Each criterion awards partial credit linearly with respect to how well the drones perform. For example, if the drones only identify half of the intel caches, then they are awarded 15 out of the 30 points for the first criterion.

Table I: Grading criteria for the ISR scenario.

Relative Weight	Description
30	% of identified intel caches
25	% of intel surveyed
15	% of drones that survive

The three criteria for the scenario are encoded into a set of requirements for the drones. The first requirement states that drones should maximize the number of intel caches that they identify, which is the first criterion of the scenario. The second requirement states that drones should maximize the number of intel units that they survey, which is the second criterion of the scenario. The third requirement, a safety requirement, states that drones should return to homebase by the end of the mission, which is the third criterion of the scenario. The last requirement is a situational goal that states that all drones should participate in surveillance tasks. This requirement indirectly stipulates that each drone should report intel caches and negotiate since a drone can only participate in surveillance tasks if it negotiates and is assigned to the task. By stating this requirement as a situational goal, a drone may terminate or reject negotiation if participating in a surveillance task would be more detrimental to its other requirements than it is beneficial to participate in the task. For example, a surveillance task could award too few intel units at too high a risk to the drone's survival.

DRONE.1: The drone should MAXIMIZE the number of intel caches that it identifies.
DRONE.2: The drone should MAXIMIZE the number of intel units that it surveys.
DRONE.3: The drone should return to homebase before the mission timer expires.
Situational DRONE.4: The drone should participate in surveillance tasks.

3.2. Negotiation protocol

In this paper, we continue to employ our extension of the SxT negotiation protocol [5]. We adapt the protocol implementation to support an individual drone that negotiates with a collective adaptive system. The negotiation protocol is defined by a set of message types that can be exchanged between two entities to initiate a negotiation, propose and counter propose integration configurations, and terminate negotiation by accepting a configuration or resolutely rejecting all previously proposed configurations. To extend this protocol to negotiations that include a collective adaptive system, we designate a leader from each collective that operates as an intermediary between a drone and the opposite collective. For our scenario, the homebase acts as the intermediary for both the collective system of surveillance drones and the collective system of wingmen. Each time a message is received from a drone, the homebase broadcasts the message to each member of the opposite collective, compiles their responses, and selects the collective's most preferred integration configuration. It then engages in negotiation with the individual drone and contacts members of the opposite collective as needed.

3.3. Stochastic multiplayer game

An SMG [13,15] is a type of stochastic model that models a scenario using a select number of players who alternate turns. During their turn, a player takes a single action before yielding their turn to the next player. The first player represents a single surveillance drone. The second player represents the environment. Due to sources of uncertainty in the environment, even though the drone may perform some action, such as looking for intel caches, it cannot guarantee that the action will be successful. The purpose of the environment player is to capture the sources of uncertainty present in the scenario through its actions. As such, while the drone player represents the drone, the environment player represents all other entities present in the scenario, including the intel caches, adversaries, wingmen, and the homebase.

The surveillance drone interacts with wingmen during negotiation via homebase which passes messages back and forth between the surveillance drones and the wingmen. Possible integration configurations are generated using random variables which determine how long the surveillance drone must wait for a wingman to reach the intel cache and how likely the surveillance task is to succeed without the drones being destroyed. This type of construction is particularly useful for scenarios where the environment player can be cooperative or competitive or are significantly influenced by stochastic processes [14]. The ISR scenario presented in this paper fits that description well. Although it exhibits competitive elements since adversaries attempt to destroy drones, the scenario does not treat adversaries as rational actors.

3.3.1. Definition. We define an SMG $\mathcal{G} = \langle \Pi, S, A, (S_{drone}, S_{env}), \Delta, AP, \chi, r \rangle$, where

- $\Pi = \{drone, env\}$ is the set of players: drone, representing a single surveillance drone, and env, representing the intel caches, adversaries, wingmen, and homebase,
- $S = S_{drone} \cup S_{env}$ is defined by the finite, non-empty sets of states of S_{drone} (controlled by player drone) and S_{env} (controlled by player env) with ($S_{drone} \cap S_{env} = \emptyset$),
- $A = A_{drone} \cup A_{env}$ is defined by the finite, non-empty set of actions of A_{drone} (available to player drone) and A_{env} (available to player env),
- $\Delta : S \times A \rightarrow \mathcal{D}(S)$ is a (partial) transition function denoting a probability distribution set S,
- AP is the set of predicates that are predicated over state variables and are used to label states,
- $\chi : S \rightarrow 2^{AP}$ is a labeling function that assigns atomic predicates to each state, and
- $r : S \rightarrow \mathbb{Q}_{\geq 0}$ is a reward function that maps a non-negative reward to each state.

We include an additional set of random variables as input parameters that define stochastic processes for the selection of actions by player env. These random variables include cache stagnation rate, cache detection rate, communication rate, likelihood of task success, intel units, and wait time. Cache stagnation rate determines the likelihood that an intel cache remains at its current location at the end of the turn. For our ISR scenario, intel caches are stagnant, but 15 of the 65 intel caches can randomly appear at any time over the course of the one-hour mission. This mechanism which permits caches to randomly pop up is similar to having intel caches that avoid a drone's detection by changing their location or hiding, as long as the drone has an imperfect (less than 1.0) detection rate.

The cache detection rate determines the likelihood that a drone discovers an intel cache at its location. The scenario does not employ impediments to cache detection, such as obstructions or caches with avoidance behavior. However, to model a scenario where 15 of the 65 intel caches randomly appear on the map, we must use an imperfect cache detection rate. Otherwise, a location that has been observed by a drone would have a zero likelihood of containing an intel cache, which can never be the case. The communication rate determines the likelihood that a drone can report the location of an intel cache. This random variable determines whether negotiation between drones can take place or whether the negotiation would be terminated, or interrupted, by environmental factors such as loss of connectivity or adversarial interference.

Once negotiation takes place, the integration configuration must include a likelihood of task success, intel units, and a wait time, which are taken from the inputs to the SMG. The likelihood of task success is a random variable that determines whether a surveillance task will be successful. If the task is successful, then the drones collect the cache's full intel units. Intel units is a random variable that determines the value of an intel cache. According to the scenario description, caches can be worth 10, 20, or 70 intel units but intel units are not uniformly distributed across the available intel caches. Finally, wait time is a random variable that indicates the number of turns a drone requires to reach the intel cache's location so that the surveillance task can begin.

3.3.2. Drone actions. Player drone takes the first turn. During their turn, player drone selects one of its available actions. Each action has a corresponding guard - a predicate over state variables to determine whether an action can be selected from a given state. Guards are specified as first-order predicates alongside their respective action. The predicate term $(t = drone \rightarrow t' = env)$ is the notation for *yield*. It states that "if the current turn owner t is player drone, then the next turn owner t' is player env." This notation restricts each player to a single action per turn.

The state variable $turns$ is the current turn in the game. Since both players are assumed to be operating simultaneously in the real world, $turns$ is incremented at the end of player env's turn. The variable $maxturns$ is the maximum number of turns that the game allows, which is 720 (i.e., one hour). All actions are restricted by the term $(turns \leq maxturns)$, which represents the terminal condition. The scenario is over once a player cannot perform any actions, which should occur only once the scenario's maximum number of turns have been played. The variable $cooldown$ is used to specify how long it takes for an action to complete. Player drone can only do nothing while on $cooldown$, and it is decremented at the beginning of player drone's turn. Actions are put into effect at the end of their cooldown, i.e., when $cooldown$ is 1. The variable HB is the location of the homebase and is set to $(-1, -1)$. The variable loc is the location that player drone intends to move to. Player drone always moves to the location that has the greatest likelihood of containing an intel cache. The function $dist(HB, loc)$ returns the number of turns required for player drone located at position loc to return to homebase.

There are several other state variables that are used as flags to determine whether an action is permitted (true) or restricted (false). The state variable $recon$ is set to true if player drone is performing a recon action while the state variable $interference$ is set to true if player env interferes with player drone's reconnaissance. The state variable neg is set to true if player drone is negotiating with homebase, and the state variable *conf* is set to true if player drone has been sent an integration configuration. The state variable *execute* is set to true when player drone executes a surveillance task.

$[\boldsymbol{a}^{drone}_{nothing}]$
Do nothing; yield to player env. Caches cannot be detected during this action.
<u>Guard</u>: $\text{turns} \leq \text{maxturns} \wedge (\text{t} = \text{drone} \rightarrow \text{t}' = \text{env})$.

$[\boldsymbol{a}^{drone}_{move}]$
Move to adjacent location; yield to player env. Caches cannot be detected during this action. Set *cooldown* to 5.
<u>Guard</u>: $\text{cooldown} = 0 \wedge \text{turns} + 5 \leq \text{maxturns} - \text{dist(HB, loc)} \wedge (\text{t} = \text{drone} \rightarrow \text{t}' = \text{env})$.

$[\boldsymbol{a}^{drone}_{search}]$
Remain at current location; recon for caches; yield to player env. Set *recon* to true. Caches can be detected during this action. Set *cooldown* to 1.
<u>Guard</u>: $\text{cooldown} = 0 \wedge \text{turns} + 1 \leq \text{maxturns} - \text{dist(HB, loc)} \wedge (\text{t} = \text{drone} \rightarrow \text{t}' = \text{env})$.

$[\boldsymbol{a}^{drone}_{recon}]$
Move to adjacent location; recon for caches; yield to player env. Set *recon* to true. Caches can be detected during this action. Set *cooldown* to 5.
<u>Guard</u>: $\text{cooldown} = 0 \wedge \text{turns} + 5 \leq \text{maxturns} - \text{dist(HB, loc)} \wedge (\text{t} = \text{drone} \rightarrow \text{t}' = \text{env})$.

$[\boldsymbol{a}^{drone}_{negotiate}]$
Report intel cache; negotiate with homebase; yield to player env. Set *neg* to true. Caches cannot be detected during this action. Set *cooldown* to 5.
<u>Guard</u>: $\text{cooldown} = 0 \wedge \neg\, \text{interference} \wedge \text{turns} + 5 \leq \text{maxturns} - \text{dist(HB, loc)} \wedge (\text{t} = \text{drone} \rightarrow \text{t}' = \text{env})$.

$[\boldsymbol{a}^{drone}_{execute}]$
Accept selected integration configuration; execute surveillance task; yield to player env. Caches cannot be detected during this action. Set *execute* to true. Action requires several turns as determined by the configuration's specified wait time and the number of turns that player drone requires to fully survey the intel cache. A surveillance drone can survey five intel units per turn. Set *cooldown* to $(waitTime + intelUnits/5)$.
<u>Guard</u>: $\text{cooldown} = 0 \wedge \text{conf} \wedge \text{turns} + \text{waitTime} + \text{intelUnits}/5 \leq \text{maxturns} - \text{dist(HB, loc)} \wedge (\text{t} = \text{drone} \rightarrow \text{t}' = \text{env})$.

Action $a^{drone}_{nothing}$ is selected whenever player drone has no other actions available because it is on cooldown or because it does not have enough turns to execute any other actions. Action a^{drone}_{move} returns player drone to homebase when it does not have enough remaining turns to execute any of its other actions. The drone cannot be on cooldown and it must have enough turns remaining to return to homebase. Action a^{drone}_{search} is used to recon the drone's current location. This action is selected when the drone's current location has at least the likelihood of containing an intel cache as all its adjacent locations. Action a^{drone}_{recon} is used to recon one of the drone's adjacent locations. This action is selected when one of the drone's adjacent locations has a greater likelihood of a containing an intel cache than the drone's current location. These two actions cannot be performed while the drone is on cooldown or if the drone does not have enough turns remaining before the mission expires. The action a^{drone}_{recon} is not a combination of the drone's other actions but is a unique action that is executed when the drone intends to both move and recon. Action a^{drone}_{move} is executed when the drone intends to move but not recon, e.g., when returning to homebase. Action a^{drone}_{search} is executed when the drone intends to recon but not move, e.g., when its current location has the greatest likelihood of containing an unidentified intel cache.

Action $a^{drone}_{negotiate}$ negotiates with homebase. It can only be performed if player drone is not being interfered with by player env and if player drone has enough turns remaining to reach homebase after negotiations have concluded. Action $a^{drone}_{execute}$ executes a surveillance task according to the integration configuration selected by player env. It can only be selected if a configuration has been sent to player drone and there are enough turns to execute the task and then return to homebase. When more than one action is available, player drone selects its action according to the following preference order: $a^{drone}_{execute}$, $a^{drone}_{negotiate}$, a^{drone}_{search}, a^{drone}_{recon}, a^{drone}_{move}, and $a^{drone}_{nothing}$.

3.3.3. Env actions. Player env is the second player to take their turn. We predicate each action's guard over a random variable Z, whose value is sampled each turn from a uniform distribution. Its guards are intentionally constructed so that player env only has one available action each turn.

$[\boldsymbol{a}^{env}_{nothing}]$ Do nothing and yield to player drone.
<u>Guard</u>: $\big((\text{Z} < \text{cacheStagRate} \wedge \neg\, \text{recon}) \vee \text{Z} < \text{cacheStagRate} \cdot \text{cacheDetectRate}\big) \wedge \neg\, \text{neg} \wedge \text{turns} \leq \text{maxturns} \wedge (\text{t} = \text{env} \rightarrow \text{t}' = \text{drone})$.

$[\boldsymbol{a}^{env}_{move}]$
Move cache to an adjacent location; yield to player drone.
<u>Guard</u>: $\big((\text{cacheStagRate} \leq \text{Z} \wedge \neg\, \text{recon}) \vee \text{cacheStagRate} \cdot \text{cacheDetectRate} \leq \text{Z} < \text{cacheDetectRate}\big) \wedge \neg\, \text{neg} \wedge \text{turns} \leq \text{maxturns} \wedge (\text{t} = \text{env} \rightarrow \text{t}' = \text{drone})$.

$[\boldsymbol{a}^{env}_{hide}]$
Cache avoids the drone's visual detection; yield to player drone.
<u>Guard</u>: $\text{cacheDetectRate} \leq \text{Z} < 1 - (1 - \text{cacheStagRate})(1 - \text{cacheDetectRate}) \wedge \text{recon} \wedge \text{turns} \leq \text{maxturns} \wedge (\text{t} = \text{env} \rightarrow \text{t}' = \text{drone})$.

$[\boldsymbol{a}^{env}_{avoid}]$
Cache avoids the drone's visual detection; move cache to an adjacent location; yield to player drone.
<u>Guard</u>: $1 - (1 - \text{cacheStagRate})(1 - \text{cacheDetectRate}) \leq \text{Z} \wedge \text{recon} \wedge \text{turns} \leq \text{maxturns} \wedge (\text{t} = \text{env} \rightarrow \text{t}' = \text{drone})$.

$[\boldsymbol{a}^{env}_{send}]$
Send integration configuration; yield to player drone. Set *conf* to true. Configuration is selected according to a uniform distribution over all possible configurations.

<u>Guard</u>: neg $\wedge$ turns $\leq$ maxturns $\wedge$ (t = env $\rightarrow$ t$'$ = drone).

$[\boldsymbol{a}^{env}_{destroy}]$ Destroy drones executing a surveillance task; yield to player drone.
<u>Guard</u>: likelihoodTaskSuccess $\leq$ Z $\wedge$ execute $\wedge$ turns $\leq$ maxturns $\wedge$ (t = env $\rightarrow$ t$'$ = drone).

Action $a^{env}_{nothing}$ is player env's action to do nothing, indirectly preventing the cache from changing location and permitting a cache to be identified by a drone performing reconnaissance. It cannot be selected if player drone is currently negotiating. Player env's second action a^{env}_{move} changes the location of a cache to an adjacent location and indirectly permits a cache to be identified by a drone performing reconnaissance. This action cannot be selected if player drone is currently negotiating. Action a^{env}_{hide} prevents player drone from identifying a cache if it is performing reconnaissance and prevents the cache from changing location. This action can only be selected if player drone is performing reconnaissance. Action a^{env}_{avoid} prevents player drone from identifying a cache and changes the cache location. This action can only be selected if player drone is performing reconnaissance. If and only if player drone is negotiating, then player env must select action a^{env}_{send}, sending an integration configuration to player drone. If player drone is executing a surveillance task, then player env may select action $a^{env}_{destroy}$ to destroy the drone.

3.3.4. Reward. The output of the SMG is defined as a set of state variables updated over the course of the SMG lifecycle. The outputs, $P("identifying\ cache")$ - the expected likelihood of locating a cache, $P("task\ success")$ - the expected task successes, and $P("task\ failure")$ - the expected task failures, assign a reward to the final state of the SMG according to the following equations:

$$\begin{aligned}
&r(S) = \big(r_1(S) + r_2(S) + r_3(S)\big)/70,\\
&r_1(S) = 30 \cdot P("identifying\ cache"),\\
&r_2(S) = 25 \cdot P("task\ success") \cdot 65 \cdot intelUnits/1800,\\
&r_3(S) = 15 \cdot 100 \cdot [1 - P("task\ failure")].
\end{aligned}$$

The reward function r measures the drone's expected performance with respect to its requirements. It is the weighted sum of the reward functions r_1, r_2, and r_3. Other reward systems are outside the scope of this work. The ISR scenario has strict grading criteria that are integral to the scenario. A change to any of the reward functions would result in the evaluation of a different ISR scenario.

The reward function r_1 measures expected performance with respect to requirement DRONE.1 (Section 3.1.2) to maximize the number of caches located. To compute r_1, we weigh the likelihood of locating a cache. Weights are determined by Table 1. The reward function r_2 measures expected performance with respect to requirement DRONE.2 to maximize the number of intel units surveyed. To compute r_2, we weigh the percentage of intel units collected as determined by the expected number of task successes multiplied by the expected cache value of each success. The reward function r_3 measures expected performance with respect to requirement DRONE.3 for drones to return to homebase before the mission ends. The guards for the drone's actions comply with this safety requirement. Therefore, drones that do not comply are those destroyed during the mission. Drones are destroyed by an adversary when they fail to complete a surveillance task. To compute r_3, we first compute the expected number of drone losses which is the expected number of task failures multiplied by the number of drones lost per failure, which is two. We then measure the total number of drones that return home which is the total drones (200) minus expected losses.

4. Ordinary least squares regression

OLS regression attempts to compute a set of coefficients that minimize the sum of squared residuals, where a residual is the difference between the model's prediction of reward and the true reward as output by the SMG. Each coefficient is assigned to a specific feature contained in the training data and the OLS model also includes a constant. OLS can be used for linear or polynomial regression depending on whether the data has a linear or curvilinear relationship to the data. Since the SMG's outputs are determined by products of its inputs, we choose to use polynomial regression. If it is not known whether the data is linear or curvilinear with respect to the input features, there are many linearity tests that can be applied to the data. In addition, OLS assumes that the variance of each residual is the same for any input and that the data is normally distributed.

The primary advantages of using OLS regression is that OLS models are extremely concise and can predict the output reward in constant time. This is because the model is defined as an array of coefficients, or weights, making OLS regression useful for low power devices. Predictions can be computed in constant time which poses the smallest burden possible on a device's already limited battery life.

We employ OLS regression to predict the expected reward of an integration configuration. We use a subset of the SMG's input parameters – cache stagnation rate, cache detection rate, communication rate, likelihood of task success, and intel units – to compute the model's features. We also include downtime as a model parameter, which represents the number of turns that the drone is non-productive (no reconnaissance or surveying). Each drone action has a non-negative cooldown incurred when the action is executed. However, it is also influenced by the integration configuration. Every configuration must specify a wait time required for the secondary team member to reach the intel cache's location. This wait time is non-productive and added to the drone's downtime. Even though downtime is considered an output of the SMG, we include it as a parameter since the model evaluates configurations where the downtime is specified as a wait time.

$$\hat{r} = \hat{r_1} + \hat{r_2} + \hat{r_3},$$

The expected reward $\hat{r}$ of the SMG is a function of its outputs – expected number of caches located $\hat{r_1}$, the expected amount of intel units collected $\hat{r_2}$, and the expected number of task failures $\hat{r_3}$. These outputs are computed within the SMG using a linear combination of higher order polynomial terms. For example, to determine the expected number of caches located, we consider both cache stagnation rate and cache detection rate. The likelihood of a cache being at any location within the map is the sum of the likelihoods that the cache was either at that location and remained there or at an adjacent location and moved there. This remains true throughout SMG execution resulting in a higher order polynomial of the term cache stagnation rate. The highest degree amongst these polynomial terms is maxturns (the length of the game).

$$\hat{r_1} = (cacheStagRate + cacheStagRate^2 + \cdots) \cdot cacheDetectRate,$$
$$\hat{r_2} = \hat{r_1} \cdot communicationRate \cdot P(\text{"}task\ success\text{"}) \cdot intelUnits,$$
$$\hat{r_3} = \hat{r_1} \cdot communicationRate \cdot [1 - P(\text{"}task\ success\text{"})],$$

To model the expected number of caches located, we use the product of the sequence of polynomial terms defined over cache stagnation rate and cache detection rate. The cache must both be at the location and detected. We refer to this sequence of terms as the ***detection estimator*** $\hat{r_1}$. A task can only be attempted if a cache is detected and then reported. Attempted tasks succeed according to the likelihood of task success and every successful task awards the cache's total number of intel units. We refer to this sequence of terms as the ***intel estimator*** $\hat{r_2}$. Tasks that are attempted fail according to the inverse of the likelihood of task success. We refer to this sequence of terms as the ***failure estimator*** $\hat{r_3}$.

5. Evaluation

Our evaluation is divided into two subsections. First, we investigate the use of OLS polynomial regression. Second, we include the regression model in the SMG and modify its actions so that negotiation is always terminated if the proposed integration configuration would result in a smaller expected reward than a failed negotiation, as predicted by the regression model. We compare the output of an SMG with and without an OLS model. The results of our evaluation show that the OLS model can predict the expected performance of the drone within an error of 0.0024 and that including this model into the SMG improves expected performance and significantly improves the drone's success-to-failure ratio.

5.1. Evaluating the regression models

The sample dataset for these results includes 2,400 samples of the SMG. Samples are generated using a 25x25 grid, a maxturns of 720, a constant cache stagnation rate of 0.9, a range of all inputs of cache detection rates from 0.05 to 1.0 in increments of 0.05, a constant communication rate of 1.0, a range of intel units that include 10, 20, and 70, a non-uniform distribution of wait times based on the distribution of drones throughout the 25x25 grid which results in wait times from 5 to 35 turns, and 40 random samples of the likelihood of task success using a uniform distribution. 60% of the samples are randomly selected and compiled into a training set while the other 40% are compiled into a holdout set. The model is trained using the training set and scored against the holdout set. We apply 10-fold cross validation to the training set using 60% of the set for validation and 40% of the set for testing in each fold. The performance of the model is compared to the results of the 10-fold cross validation to check for overfitting.

Table II presents the results from the model attempting to predict the test set after having been trained using the training set. The OLS model is parameterized over the detection estimator, intel estimator, failure estimator, and downtime. The degree of each polynomial term in the higher order models is limited to a maximum of three. Degrees larger than three improve the results of the model but only by less than 0.0001. The model is scored using the mean absolute error (MAE), the root mean squared error (RMSE), and R-Squared (R^2). MAE measures the absolute distance between the model's predictions and the true reward value contained in the test set to indicate model accuracy. RMSE captures the average magnitude of the error – a second indication of accuracy that is more sensitive to larger errors. R^2 indicates the variance of the data that can be attributed to the model's features. An R^2 value of 0.0 occurs when the model ignores the training data, and an R^2 value of 1.0 occurs when the data's variance is fully determined by the model's features.

Table II: MAE, RMSE, and R^2 scores from model prediction.

Model	MAE	RMSE	R^2
OLS	0.0017	0.0024	0.9973

Table II results show that the OLS model has a MAE value of 0.0017 and an RMSE value of 0.0024. Thus, the model's prediction will be, on average, within 0.0024 of the true reward of the SMG. Since the values of the RMSE and the MAE are relatively close, there is no clear indication of large errors present in the data. The OLS model has an R^2 value 0.9973, which indicates that the model's features explain nearly all the variance present in the data.

Table III: Mean & std deviation of MAE, RMSE, and R^2.

	OLS	
	Mean	Stdev
MAE	0.0017	<0.0001
RMSE	0.0025	0.0001
R^2	0.9972	0.0004

These results are compared to the results of the 10-fold cross validation that we apply to the model's training set. Table III shows the mean and standard deviation of the MAE, RMSE, and R^2 for the model across all 10 folds. Comparing the mean values in Table III to the results in Table II, the values are within 0.0001 for MAE, RMSE, and R^2. The standard deviation is less than 0.0001 for both MAE and RMSE, indicating the error of the prediction is similar across all 10 folds. Thus, the model has most likely not overfit the training data and should perform well for larger datasets.

5.2. Embedding the Model

We embed the OLS model into the SMG and modify the guard of its execute task action so that it is only permitted when the proposed integration configuration is predicted to have a greater reward than terminating the negotiation. We compare the output of the SMG with and without the OLS model to determine the relative impact that embedding the model has on the drone's expected performance. Both SMGs are sampled across 2,400 trials. We use the same process to generate the samples for the experiment as we did to generate the training and test data.

The results of our experiment are provided in Table IV. Each row states the output of the SMG without the OLS Model and the SMG with the OLS Model. The final row shows the final computed reward. Each result is averaged over the number of trials. The results in the final column of Table IV indicate that the inclusion of the OLS model is expected to increase the drone's final reward by increasing the expected number of caches that it identifies and reducing the number of failed surveillance tasks.

Table IV: SMG output with and without the OLS model.

	No OLS	OLS	Change
Expected Likelihood of Caches Identified	0.1180	0.1184	+0.0004
Expected Tasks Rejected	0.0000	0.0667	+0.0667
Expected Task Successes	0.0590	0.0374	-0.0216
Expected Task Failures	0.0590	0.0143	-0.0447
Expected Success Rate	50.00%	72.34%	+22.34%
Expected Failure Rate	50.00%	27.66%	-22.34%
Reward (r)	0.2820	0.2830	+0.0010

Even though fewer tasks are attempted and the expected performance of the surveillance drone only increases by 0.0010, maintaining the same reward while significantly reducing the loss of drones is an extremely positive outcome. Without the OLS model, a single surveillance drone is expected to identify and attempt to survey 11.8% of intel caches, collect intel from 50% of those tasks, and be destroyed attempting the other 50% of tasks. With the OLS model, a single surveillance drone is expected to identify 11.84% of intel caches, attempt to survey 5.17% of caches, collect intel from 72.34% of tasks, and be destroyed attempting the other 27.66% of tasks. With the OLS model, a team of drones is nearly three-times more likely to succeed when attempting to survey an intel cache than they are to fail. Furthermore, the SMG does not measure the relative utility of a surveillance drone over time. Once destroyed, a drone can no longer identify intel caches, survey intel, or interfere with adversaries. The expected performance of each SMG does not consider the potential loss of caches identified or surveyed should surveillance drones be destroyed earlier in the mission. As such, the expected performance of the SMG without the OLS model included is likely to be smaller since it has a much higher failure rate.

6. Conclusion

In this paper, we present an ISR scenario where two collective adaptive systems of drones must work together to survey a target amount of intel within a single hour. The scenario begins with each drone deploying from homebase to recon the search grid for intel caches. Each intel cache is protected by a single adversary, so drones must negotiate to self-organize into a two-drone team to complete surveillance tasks as intel caches are identified. We model the scenario using an SMG and include sources of uncertainty as random variables. The SMG models the behavior of a single surveillance drone. An OLS polynomial regression model is trained on the output of the SMG after combining the SMG's inputs into a set of features. The results demonstrate that the OLS model can predict the output of the SMG within a quarter of a percent on average. We incorporate the OLS model into the SMG by modifying its actions so that it can only execute a surveillance task that has been permitted by the regression model. Comparing the results of the SMG with and without the OLS model shows that the inclusion of the regression model improves the expected performance of the drones and significantly improves the drone's success-to-failure ratio.

For future work, we would like to investigate additional scenarios and reward systems to better evaluate our approach across applications. We would like to extend the SMG to evaluate the expected performance loss should a surveillance drone be destroyed at various times throughout the mission. This would permit us to construct a more accurate model of the drone's performance as well as produce a model that can predict the drone's expected performance over time. Finally, we would like to investigate the use of other regression models and conduct a deeper analysis of each feature used in the regression model.

10. References

[1] A. Singh, A. Payal, and S. Bharti, "A walkthrough of the emerging IoT paradigm: Visualizing inside functionalities, key features, and open issues," J. of Network & Computer Applications, 143:111-151, 2019.

[2] D. King and G. Peterson, "A Macro-Level Order Metric for Self-Organizing Adaptive Systems," IEEE 12th Int'l Conf. on Self-Adaptive and Self-Organizing Systems, pp. 60-69, 2018.

[3] I. Riley and R.F. Gamble, "Evaluating the Impact of Design Constraints on Expected System Performance," 4th Int'l Workshop on Engineering Collective Adaptive Systems, 2019.

[4] I. Breddin, "Self-organisation and emergence," Technical University of Berlin, Tech. Rep., 2006.

[5] I. Riley, S. Jahan, and R. Gamble, "Toward a Negotiation Framework for Self-Integration", 7th Self-Improving Systems Integration Workshop, 2020.

[6] N. Dragoni, F. Massacci, and A. Saidane, "A self-protecting and self-healing framework for negotiating services and trust in autonomic communication systems," Comput. Netw., 53(10):1628–1648, 2009.

[7] E. Bertino, E. Ferrari, and A. Squicciarini, "Trust negotiations: concepts, systems, and languages," Computing in Science & Engineering, 6(4):27-34, 2004.

[8] A. Strunk, "QoS-Aware service composition: A survey," 8th Eur. Conf. Web Services, p. 67–74, 2010.

[9] I. R. Chen, J. Guo, F. Bao, and J. Cho, "Trust management in mobile ad hoc networks for bias minimization and application performance maximization," Ad Hoc Netw., 19:59–74, 2014.

[10] I. Riley, S. Jahan, A. Marshall, C. Walter, and R. Gamble, "Evaluating verification awareness as a method for assessing adaptation risk", Future Generation Computer Systems, 119: 110-135, 2021.

[11] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," R. de Lemos, H. Giese, H.A. Muller, M. Shaw, (eds.) Software Engineering for Self-Adaptive Systems II, LNCS, 7475:214-238, Springer, 2013.

[12] S. Alqahtani, I. Riley, S. Taylor, R.F. Gamble, and R. Mailler, "Task Allocation in Uncertain Environments using a QuadTree and Flow Network," 2018 Int'l Conf. on Unmanned Aircraft Systems, 2018.

[13] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis, "Automatic verification of competitive stochastic systems," Formal Methods in System Design, 43(1):61–92, 2013.

[14] J. Camara, D. Garlan, G.A. Moreno, and B. Schmer, "Analyzing self-adaptation via model checking of stochastic games," R. de Lemos, H. Giese, H.A. Muller, M. Shaw, (eds.) Software Engineering for Self-Adaptive Systems III, LNCS, 9640:154-187, Springer, 2013.

[15] M. Ummels, Stochastic multiplayer games theory and algorithms, Amsterdam: Amsterdam University Press, 2010.

[16] G. Ciaburro, Regression Analysis with R:, 1st ed., Birmingham, U.K.: Packt Publishing, Jan. 2018.

[17] Y. Wang, et al., "CATrust: Context-Aware Trust Management for Service-Oriented Ad Hoc Networks," IEEE Trans. on Services Comp., 11(6):908-921, 2018.

[18] Y. Yu, L. Guo, X. Wang, and C. Liu, "Routing security scheme based on reputation evaluation in hierarchical ad hoc networks," Comput. Netw., 54(9): 1460–1469, 2010.

[19] I. R. Chen, J. Guo, and F. Bao, "Trust Management for SOA-based IoT and Its Application to Service Composition," IEEE Trans. Services Comp., 9(3):482–495, 2016.

[20] Y. Wang, I. Chen, J. Cho, A. Swami and K. Chan, "Trust-Based Service Composition and Binding with Multiple Objective Optimization in Service-Oriented Mobile Ad Hoc Networks," IEEE Trans. on Services Comp., 10(4):660-672, 2017.

[21] C. Glaßer, C. Reitwießner, H. Schmitz, and M. Witek, "Approximability and hardness in multi-objective optimization," in Proc. Programs, Proofs, Processes. 6th Int. Conf. Comput. Eur., p. 180–189, 2010.

[22] Z. Li, X. Li, V. Narasimhan, A. Nayak, and I. Stojmenovic, "Autoregression models for trust management in wireless Ad hoc networks," IEEE Global Telecommun. Conf., p. 1–5, 2011.

[23] R. Venkataraman, M. Pushpalatha, and T. R. Rao, "Regressionbased trust model for mobile ad hoc networks," Inf. Secur. IET, 6(3):131–140, 2012.