Notice

When Human Intuition Fails: Using Formal Methods to Find an Error in the "Proof" of a Multi-Agent Protocol*

Jennifer A. Davis[1], Derek B. Kingston[2], Laura R. Humphrey[3]

***Abstract*— Designing protocols for multi-agent interaction that achieve the desired behavior is a challenging and error-prone process. Proofs of protocol correctness rely on human intuition and require significant effort to develop. Even then, proofs can have mistakes that may go unnoticed after peer review, modeling and simulation, and testing of the resulting system. The potential for errors can be reduced through the use of *formal methods*, i.e. automated or semi-automated mathematically rigorous tools and techniques for system specification, design, and verification. In this paper, we apply a type of formal method called *model checking* to a previously published decentralized protocol for coordinating a surveillance task across multiple unmanned aerial vehicles. The original publication provides a compelling proof of correctness, along with extensive simulation results to support it. However, our analysis found an error in one of the proof's lemmas. In this paper, we provide an overview of the protocol, its original "proof" of correctness, the model checking approach we used to analyze it, the counterexample returned by the model checker, and the insight this counterexample provides into why the original lemma was incorrect. We also discuss how the formal modeling process revealed that certain aspects of the protocol were underspecified, and what future efforts would be needed to fully verify it with formal methods.**

I. INTRODUCTION

Many robotics applications require multi-agent interation, where interactions can range from unintentional and only as necessary, as in maneuvers for inter-agent collision avoidance, to intentional and tightly coordinated, as in swarms. Interactions can be centralized, i.e. a single agent makes decisions for all agents in the system, or they can be decentralized, i.e. each agent makes its own decisions with only partial information about other agents in the system.

In any case, designing protocols for inter-agent interaction that achieve the desired behavior can be challenging. The design process is often manual, i.e. performed by humans, and generally involves creating mathematical models of possible agent behaviors and candidate protocols, then manually developing a proof that the candidate protocols are correct with respect to the desired behavior. However, proofs can have mistakes that may go unnoticed even after peer review, modeling and simulation, and testing of the resulting system.

*This work was supported by AFRL/RQ contract #FA8650-17-F-2220 and AFOSR award #17RQCOR417. DISTRIBUTION A. Approved for public release: distribution unlimited. Case #88ABW-2018-4275.

[1]J. A. Davis is with Rockwell Collins, Cedar Rapids, IA 52498, USA jen.davis@rockwellcollins.com

[2]D. B. Kingston is with Aurora Flight Sciences, Manassas, VA 20110, USA kingston.derek@aurora.com. At the time of writing, he was with the Air Force Research Laboratory[3]

[3]L. R. Humphrey is with the Control Science Center of Excellence, Air Force Research Laboratory, Dayton, OH 45433, USA laura.humphrey@us.af.mil

The potential for errors can be reduced through the use of *formal methods*, i.e. automated or semi-automated mathematically rigorous tools and techniques for system specification, design, and verification [1]. Formal methods come from the computer science community and have traditionally been used for software and hardware verification. However, formal methods are increasingly being used to improve the design of agent and multi-agent systems [2].

Here, we demonstrate the value of formal methods by applying them to a manually designed decentralized protocol for coordinating a surveillance task across multiple unmanned aerial vehicles (UAVs). This protocol, called the Decentralized Perimeter Surveillance System (DPSS), was previously published in 2008 in Ref. [3], has received close to 200 citations to date, and in addition to extensive simulation results, provides a compelling proof that the protocol is correct. However, by formalizing the protocol, we found that certain aspects of it were underspecified. Then by analyzing the fully specified protocol, we found that one of the claims about it, namely an upper bound on the convergence time of one of its phases, was incorrect. Furthermore, the analysis result provided insight into why the claim was incorrect.

One of the major approaches in formal methods is model checking. Here, we use a model checker to model and verify DPSS. Model checking uses efficient algorithms to (in effect) exhaustively explore all possible states of a system model to verify whether it satisfies a formal specification. If not, it returns a counterexample that demonstrates how the system is able to violate its specification. More specifically, we use a model checker called the Assume Guarantee REasoning Environment (AGREE) [4].

In what follows, we start by describing DPSS and its original proof of convergence in Section II. In Section III, we show how we modeled DPSS formally in AGREE. In Section IV, we discuss the analysis results provided by AGREE, in particular how they show that 1) the convergence bound for one of the phases of DPSS is incorrect and 2) the overall convergence bound is overly conservative in some cases. Section V concludes with a discussion of how the formal modeling process also revealed that portions of DPSS were underspecified, and what future work would be required to modify and fully verify DPSS with formal methods.

II. DECENTRALIZED PERIMETER SURVEILLANCE SYSTEM (DPSS)

UAVs can be used to perform continual, repeated surveillance of a large perimeter over an indefinite timespan, e.g. for security or safety monitoring purposes. In such cases, more

frequent coverage of every point along the perimeter can be achieved by evenly dividing surveillance of it across multiple UAVs. However, coordinating this division is challenging in practice for several reasons. First, the exact location and length of the perimeter may not be fully known a priori, and it may even change over time, as in a growing forest fire or oil spill. Second, UAVs might need to go offline and come back online during surveillance, e.g. for refueling or repairs. Third, inter-UAV communication is unreliable, especially over long distances or without line-of-sight, so it is not always possible to immediately communicate locally observed information about perimeter changes or UAVs leaving or joining the search to other UAVs in the system. However, such information is needed to ensure an even division of the perimeter as changes occur.

DPSS provides a method to optimally divide surveillance of a perimeter across multiple UAVs with minimal communication. In DPSS, UAVs only communicate when they are co-located, at which point they share whatever information they have gathered about the perimeter and other UAVs in the system. Each UAV then uses its available information to estimate which segment of the perimeter it is responsible for surveilling. Essentially, DPSS defines a simple decentralized protocol that each UAV individually follows to record locally learned information, share it with UAVs it encounters, and refine its estimate of its perimeter segment over time.

In DPSS, a perimeter is an open curve, which is isomorphic to a line segment. We therefore define a perimeter as a line segment along the x-axis, with a "left" endpoint at $x = 0$ and a "right" endpoint at $x = P$. Let N be the number of UAVs in the system or on the "team," indexed from left to right as $1, \ldots, N$. The optimal configuration of DPSS is defined as follows.

Definition 1. Consider the following two sets of UAV locations on the perimeter:

1) UAV $i \in 1 \ldots N$ is located at $\lfloor i + \frac{1}{2}(-1)^i \rfloor P/N$
2) UAV $i \in 1 \ldots N$ is located at $\lfloor i - \frac{1}{2}(-1)^i \rfloor P/N$

where $\lfloor \cdot \rfloor$ returns the largest integer less than or equal to its argument. The optimal configuration is realized when UAVs synchronously oscillate between these two sets of locations, where each UAV moves at constant speed V.

Note that these two sets of locations divide the perimeter into N segments of length P/N, one for each UAV. Also, the optimal configuration requires all UAVs to arrive at the set of locations (1) at the same time, and the same for (2). This synchronization is desirable because then every point on the perimeter is surveilled at a regular interval (see Ref. [3] for further discussion). Fig. 1a shows a system of four UAVs evenly spaced along the perimeter. Fig. 1b and Fig. 1c show the UAVs as they move between (1) and (2), respectively.

The goal of DPSS is to achieve the optimal configuration in the steady state, i.e. when the perimeter and involved UAVs remain constant. Let each UAV $i \in 1, \ldots, N$ store a vector $\xi_i = [P_{R_i} \;\; P_{L_i} \;\; N_{R_i} \;\; N_{L_i}]^T$ of coordination variables containing information UAV i has locally gathered, where P_{R_i} and P_{L_i} are the estimated length of the perimeter to the right and left of UAV i, and N_{R_i} and N_{L_i} are the estimated number of UAVs to the right and left of UAV i, respectively. UAV i can then estimate its perimeter segment by calculating the perimeter length $P = P_{R_i} + P_{L_i}$, the team size $N = N_{R_i} + N_{L_i} + 1$, its relative order on the team $n = N_{L_i} + 1$, and its segment endpoints $\lfloor n \pm \frac{1}{2}(-1)^n \rfloor P/N$.

In what follows, we define the DPSS protocol and prove it converges to the optimal configuration within $5T$ when N and P are fixed, where $T = P/V$ is the time it takes one UAV to traverse the entire perimeter. We start by describing the protocol when every UAV has correct coordination variables, i.e. that all match the true state of the world, and show that it results in the UAVs synchronously oscillating between the optimal configuration locations in Definition 1, i.e. their segment endpoints. We call this Algorithm A and show it converges within $2T$. We then extend the protocol so that UAVs exchange information when they are co-located to handle the case that the UAVs do not start with correct coordination variables. We call this Algorithm B and show that it results in every UAV having correct coordination variables within $3T$. After the coordination variables are correct, Algorithm B converges in the same manner as Algorithm A, so the total convergence time of DPSS is $5T$. Details of the proof follow.

if UAV i rendezvous with neighbor j **then**
  Calculate team size $N = N_{R_i} + N_{L_i} + 1$.
  Calculate perimeter length $P = P_{R_i} + P_{L_i}$.
  Calculate UAV i's relative index $n = N_{L_i} + 1$.
  Calculate UAV i's segment endpoints:
    $\mathcal{S}_i = \left\{ \lfloor n - \frac{1}{2}(-1)^n \rfloor P/N, \lfloor n + \frac{1}{2}(-1)^n \rfloor P/N \right\}$.
  Communicate $\mathcal{S}_i$ to neighbor j and receive $\mathcal{S}_j$.
  Calculate shared border position $p_{i,j} = \mathcal{S}_i \cap \mathcal{S}_j$.
  Travel with neighbor j to shared border position $p_{i,j}$.
  Set direction to monitor own segment.
else if reached perimeter endpoint **then**
  Reverse direction.
else
  Continue in current direction.
end if

Alg. A: Neighbor escort from the perspective of UAV i.

1) Algorithm A: Consider the protocol described in Algorithm A, where each UAV escorts any neighboring UAV it encounters to the shared boundary of their perimeter segments. Note that UAVs do not automatically turn around if they reach a segment boundary; they only turn around if they encounter a *perimeter* endpoint or start or stop a neighbor escort. Meeting and escorting neighbors to segment boundaries is key for both obtaining correct coordination variables and achieving the synchronization required for the optimum configuration. Also note that each UAV $i \in 1, \ldots, N$ continuously updates P_{R_i} and P_{L_i} based on the distance it travels, and UAVs are able to tell when they reach a perimeter endpoint.

Theorem 1. Let the perimeter length P and the N UAVs in the system be fixed. If every UAV's coordination variables are correct, then Algorithm A achieves the optimal configuration within $2T$.

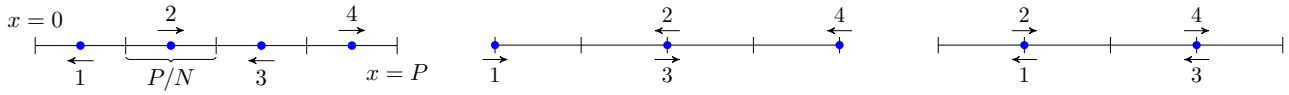

Fig. 1: DPSS in its optimal configuration. a) UAVs are uniformly spaced along the perimeter. b) Neighbors meet at locations (1), exchange information, and turn around. c) Neighbors meet at locations (2), exchange information, and turn around.

Proof. UAVs can initially be positioned anywhere along the perimeter and can be traveling either left or right at constant speed V. Each UAV has correct coordination variables and so can accurately calculate its perimeter segment. Also, each UAV is guaranteed to meet its neighbors since each UAV only reverses direction at a *perimeter* (not segment) endpoint or when starting or stopping a neighbor escort.

For N UAVs monitoring a perimeter of length P, order segments of size P/N from the left edge of the perimeter as $1, \ldots, N$, so that UAV i is responsible for segment i. Label the left endpoint of segment 1 as p_1, the right endpoint of segment N as p_N, and the shared endpoint of segments i and j as $p_{i,j}$. Let T_s be the time for a UAV to travel once across a segment, which is the same for every UAV.

Consider first the actions of UAV 1. Once UAV 1 has escorted UAV 2 to $p_{1,2}$, then no UAV to the right of UAV 1 will ever enter segment 1 by crossing $p_{1,2}$ again. This is because UAV 1 will make the round trip from $p_{1,2}$ to p_1 back to $p_{1,2}$ in $2T_s$, whereas UAV 2 would require at least $2T_s$ to return to $p_{1,2}$ after reaching or passing $p_{2,3}$. This is because there are two cases for UAV 2, as depicted in Figure 2. UAV 2 will either meet UAV 3 on segment 2, escort it to $p_{2,3}$, and turn around and reach $p_{1,2}$ in exactly $2T_s$, or UAV 2 will meet UAV 3 to the right of $p_{2,3}$, turn around to escort it to $p_{2,3}$, then continue on toward $p_{1,2}$, which would require more time than $2T_s$. UAV 2 will therefore either meet UAV 1 at $p_{1,2}$, or it will meet UAV 1 to the right of $p_{1,2}$, where UAV 1 will escort it to $p_{1,2}$ and then UAV 2 will reverse direction back towad $p_{2,3}$. In either case, $p_{1,2}$ can now be regarded as a fixed perimeter endpoint for UAV 2, and the same argument can be repeated (once UAV 1 has completed escorting UAV 2) if we consider UAV 2 as the leftmost UAV in a set of $N-1$ UAVs. Therefore, there is a time τ after which all UAVs are constrained to their respective segments, and since UAVs only change direction when meeting their neighbors at their shared segment boundaries or a perimeter endpoint, the optimal configuration in Definition 1 is reached.

The worst case occurs when all UAVs are stacked at one end of the perimeter and are traveling toward the other end. Once T has passed, all UAVs are at the opposite end of the perimeter where they meet both of their neighbors. Each pair will travel to their shared borders, which for the farthest pair will require a travel time less than T. Therefore, the steady-state behavior will be achieved before time $2T$. □

2) Algorithm B: Now consider Algorithm B, which is essentially Algorithm A with the additional steps of communicating and updating the coordination variables. We show that this protocol achieves the optimal configuration within $5T$ for arbitrary initial conditions of position, direction, and coordination variables for each UAV in the system. We first show that every UAV obtains correct coordination variables within $3T$, then by Theorem 1, the system takes an additional $2T$ to converge to the optimal configuration.

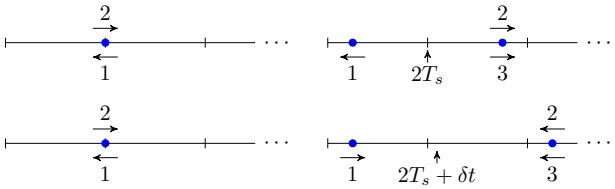

Fig. 2: Possible cases for the rendezvous between UAV 1 and UAV 2. They either meet in $2T_s$ at their shared segment boundary or to the right of it at $2T_s + \delta t$.

1: **if** agent i (left) rendezvous with neighbor j (right) **then**
2: Update perimeter length and team size:
3: $P_{R_i} = P_{R_j}$
4: $N_{R_i} = N_{R_j} + 1$
5: Calculate team size $N = N_{R_i} + N_{L_i} + 1$.
6: Calculate perimeter length $P = P_{R_i} + P_{L_i}$.
7: Calculate relative index $n = N_{L_i} + 1$.
8: Calculate segment endpoints:
9: $\mathcal{S}_i = \left\{ \lfloor n - \frac{1}{2}(-1)^n \rfloor P/N, \lfloor n + \frac{1}{2}(-1)^n \rfloor P/N \right\}$.
10: Communicate $\mathcal{S}_i$ to neighbor j and receive $\mathcal{S}_j$.
11: Calculate shared border position $p_{i,j} = \mathcal{S}_i \cap \mathcal{S}_j$.
12: Travel with neighbor j to shared border $p_{i,j}$.
13: Set direction to monitor own segment.
14: **else if** reached left perimeter endpoint **then**
15: Reset perimeter length to the left $P_{L_i} = 0$.
16: Reset team size to the left $N_{L_i} = 0$.
17: Reverse direction.
18: **else if** reached right perimeter endpoint **then**
19: Reset perimeter length to the right $P_{R_i} = 0$.
20: Reset team size to the right $N_{R_i} = 0$.
21: Reverse direction.
22: **else**
23: Continue in current direction keeping track of traversed perimeter length.
24: **end if**

Alg. B: Neighbor escort with coordination variable update from the perspective of UAV i. UAV j follows the same protocol but updates P_{L_j} and N_{L_j} in lines 3-4 instead.

Lemma 1. Using Algorithm B, every UAV will obtain correct coordination variables within $3T$.

Proof. We first prove that all UAVs converge to correct coordination variables in finite time. Note that since a UAV only changes direction at perimeter endpoints or when starting or

stopping a neighbor escort, all UAVs are guaranteed to meet their neighbors.

Label UAVs, segments, and endpoints as in Algorithm A and consider the actions of UAV 1. UAV 1 is guaranteed to visit p_1 either after an escort from UAV 2 or before having met any other UAVs, depending on initial conditions. Once UAV 1 has visited p_1, both N_{L_1} and P_{L_1} are correct. At the next meeting of UAV 1 and 2, UAV 2 updates its "left" coordination variables N_{L_2} and P_{L_2} through communication with UAV 1, thereby obtaining correct values for them. Note that repeated meetings between UAV 1 and 2 will not change the value of these coordination variables since N and P are fixed. Now consider UAV 2 as the leftmost UAV in a team of $N-1$ UAVs, and note that UAV 3 is similarly ensured to obtain correct left coordination variables. Since only one neighbor meeting is required after the leftmost UAV has obtained correct values for its left coordination variables, the number of UAVs needing correct "left" coordination variables is reduced at each stage, and since meetings are guaranteed to occur in finite time, every UAV in the system obtains correct values for its left coordination variables in finite time. Clearly, the same argument holds for the right end of the perimeter and right coordination variables.

During the transient period when the UAVs are learning the correct coordination variables, the calculation of shared segment boundaries is incorrect relative to the optimal configuration, but it is consistent between UAVs involved in the rendezvous and escort. This can be seen by noting that after both UAVs have communicated and updated their coordination variables with each other, they each have the same understanding of P and N and so calculate the same shared segment boundary position. So while they escort each other to the (ultimately) wrong position, they are still guaranteed to continue in the correct direction afterward to ensure that each UAV meets both its neighbors.

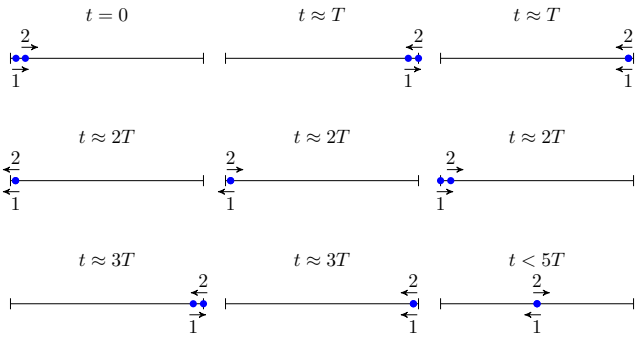

Fig. 3: Worst case scenario for convergence for Algorithm B demonstrated with 2 UAVs.

The worst case for obtaining correct coordination variables occurs when the overlap of the UAVs is the greatest, that is, when UAVs travel together rather than space out along the perimeter. Consider N UAVs stacked near $x = 0$ and headed right (Figure 3a). After time T has passed, UAV N will reach the right perimeter endpoint and update P_{R_N} and L_{R_N} to correct values (Figure 3b). Once UAV N has visited p_N, it will rendezvous with the rest of the UAVs in the system (Figure 3c). At this instant, the UAVs have arbitrary values for their "left" coordination variables. Suppose the estimated shared border position for UAVs N and $N-1$ is at $x = \epsilon$ for $\epsilon << P$ and is less than that for all other UAVs. In this case, the entire team will again nearly travel the length of the perimeter to $x = \epsilon$ which requires less than another T units of time (Figure 3d). UAV N then separates from the team and begins heading toward the right end of the perimeter without learning about the left perimeter endpoint (Figure 3e). UAVs $1 \ldots N-1$ encounter the left endpoint of the perimeter and update to a completely correct set of coordination variables (Figure 3f). With correct coordination variables, UAVs begin to space out equally along the perimeter. Note that UAV $N-1$ will chase UAV N the entire length of the perimeter since it will escort UAV $N-2$ to their shared border position and then continue to the right. UAV $N-1$ then meets UAV N near the right end of the perimeter in T units of time since reaching the left endpoint (Figure 3g). Once they meet, UAV N will update to correct coordination variables (Figure 3h). At this point, $3T$ has passed. □

Theorem 2. Let the perimeter length P and the N UAVs in the system be fixed. Then an upper bound for convergence of Algorithm B to the optimal configuration is $5T$.

Proof. By Lemma 1, an upper bound on the amount of time for all UAVs to obtain correct coordination variables is $3T$. Then by Theorem 1, up to an additional $2T$ is needed for the UAVs to reach the optimal configuration. Therefore, an upper bound for the convergence of Algorithm B is $5T$. □

III. FORMAL MODELS

In this section, we present formal models of DPSS developed in AGREE. AGREE uses assume/guarantee reasoning to perform verification of architectures modeled as a top-level system with multiple lower-level components, each having a formally specified assume/guarantee contract consisting of a set of assumptions (on the inputs) and guarantees (on the outputs). The system-level assumptions and component assume/guarantee contracts are assumed to be true. AGREE then attempts to verify (a) that component assumptions hold given system-level assumptions, and (b) that system-level guarantees hold given component guarantees. The language used by AGREE is an "annex" to the Architecture Analysis and Design Language (AADL), an architecture description language standardized by SAE International and commonly used in the aerospace industry [5].

AGREE poses the verification problem as a satisfiability modulo theory (SMT) problem [6]. SMT allows for reasoning over first-order logic formulas where predicates can come from a variety of underlying theories, e.g. they can include systems of inequalities over real-typed variables and arithmetic expressions over integer-typed variables. Predicates can also encode changes to system state over time, as in bounded model checking [7]. AGREE uses a k-induction model checking approach [8] with SMT to perform

```
const V : real = 1.0; --UAV velocity
const LEFT : int = -1;
const RIGHT : int = 1;
```

Listing 1: Constants file for Algorithm B for 3 vehicles.

an exhaustive search for predicate values that violate the system-level guarantees given system-level assumptions and component-level assume/guarantee contracts. If it finds such a set of values, it returns them as a counterexample.

For brevity, we present only some portions of the AGREE model for Algorithm B. The model for Algorithm A is similar, except it is simpler because it does not need to model updates of the UAV coordination variables. The complete models for both can be found on GitHub [9][1].

The model consists of a top-level system model and a component-level UAV model that is instantiated multiple times. We henceforth refer to these simply as the "system" and the "UAV(s)." The system sets valid ranges for the initial conditions for each UAV through a set of top-level assumptions. It also essentially runs a discrete event simulation of the UAVs as they follow the DPSS protocol. Events include a UAV reaching a perimeter endpoint or two UAVs starting or stopping an escort. At initialization and after each event, the system uses the globally known constant UAV speed `V` and direction information from each UAV to determine the amount of time `deltaT` until the next event, and it uses this to update the positions of the UAVs. The UAV essentially uses a component-level assume/guarantee contract to model the behavior of a UAV, which includes updating the values of coordination variables when neighbors meet and changing direction at a perimeter endpoint or when starting or stopping an escort. The UAVs assist the system by computing each UAV's next "goal" location, i.e. estimated perimeter endpoint or shared segment boundary, and the system determines `deltaT` by finding the minimum time until the next event across all UAVs. Note that the system handles the case that a UAV's goal is an estimated perimeter endpoint that falls short of the actual perimeter endpoint.

Global constants accessible to both the system and UAVs are shown in Listing 1. These define the UAV speed `V` as a real value of `1.0` and enumerate directions `LEFT` and `RIGHT` as integer values of `-1` and `1`. Note that all AGREE variables have types. Ports have AADL `Base_Types` of `Integer`, `Float`, and `Boolean`, whose values are converted to `int`, `real`, and `bool` for reasoning in AGREE.

1) UAV Model: Start by considering the UAV model. Its inputs and outputs are shown in Listing 2. Values for UAV inputs are passed in by the system and include the UAV's initial direction, initial position, and updated position after every event. As discussed in Section III-.2, system assumptions provide bounds on UAV initial positions as a function of perimeter length and the initial position of other UAVs. They also allow the initial direction to be either

[1]AADL projects for this work are located under AADL_sandbox_ projects. Algorithm A and Algorithm B models for three vehicles are in the DPSS-3-AlgA-for-paper project and the DPSS-3-AlgB-for-paper project.

```
--Initial direction and position, set by system and
--essentially constrained by system-level assumptions
initial_direction : in data port Base_Types::Integer;
initial_position : in data port Base_Types::Float;
--This UAV's position, updated by system after an event
pos : in data port Base_Types::Float;
--Set by system when right/left neighbor is co-located
meet_RN : in data port Base_Types::Boolean;
meet_LN : in data port Base_Types::Boolean;
--PR and NR of right neighbor, PL and NL of left neighbor,
--passed in by system
PR_RN : in data port Base_Types::Float;
PL_LN : in data port Base_Types::Float;
NR_RN : in data port Base_Types::Integer;
NL_LN : in data port Base_Types::Integer;
--Actual right and left perimeter endpoints
right_endp_truth : in data port Base_Types::Float;
left_endp_truth : in data port Base_Types::Float;

-- This UAV's current direction and goal
direction: out data port Base_Types::Integer;
goal : out data port Base_Types::Float;
-- This UAV's coordination variables
PR : out data port Base_Types::Float;
PL : out data port Base_Types::Float;
NR : out data port Base_Types::Integer;
NL : out data port Base_Types::Integer;
```

Listing 2: Inputs and outputs for the Algorithm B UAV model.

```
assume "right_endp_truth is greater than left_endp_truth":
  right_endp_truth > left_endp_truth;

assume "right_endp_truth is fixed":
  true -> (right_endp_truth = pre(right_endp_truth));
```

Listing 3: Assumptions for the Algorithm B UAV model.

left or right. Input values from the system also include information on whether this UAV is co-located with its right or left neighbor and the values of the coordination variables of neighbors (unused if there is no left or right neighbor). Finally, they also include the true values of the left and right perimeter endpoints, which the UAV only uses to check if it is at a perimeter endpoint. UAV outputs include the UAV's direction, its next goal location, and its current coordination variables. These are sent back to the system so it can determine the time of the next event and pass the values of coordination variables between neighbors.

UAV behavior is essentially captured by the UAV's assume/guarantee contract. UAV assumptions include that the value of the right perimeter endpoint is greater than the left and that perimeter endpoints are fixed. Assumptions for the right perimeter endpoint are shown in Listing 3. Assumptions for the left perimeter endpoint are analogous. Note that a statement of the form `x -> y` is not a logical implication; rather, it means use `x` on the initial timestep, and `y` on all subsequent timesteps. This is often used in conjunction with the `pre` operator, where `pre(x)` returns the value of variable `x` on the previous timestep, which is not defined on the initial timestep. On the initial timestep, `pre(x)` can return any valid value given `x`'s type.

UAV guarantees specify the values of the UAV's `goal`, `direction`, `PR`, `PL`, `NR`, and `NL` based on the inputs and assumptions. These are shown in Listing 4, with some omitted for brevity. Equations defined by the `eq` keyword

```
--UAV's estimate of id n_id, N, P, and perimeter endpoints
eq n_id : int = NL + 1;
eq N : int = NL + NR + 1;
eq P : real = PR + PL;
eq left_endp_est : real = pos - PL;
eq right_endp_est : real = pos + PR;

--Shared border positions where n_id_real and N_real
--are N and n_id as real types returned by a lookup table
eq S_L : real = (n_id_real -1.0)*P/N_real + left_endp_est;
eq S_R : real = n_id_real*P/N_real + left_endp_est;

--Previous direction
eq pre_direction : int =
        prev(direction, initial_direction);

--Previous PR; initially can be any real value
eq pre_PR : real = pre(PR)

--Booleans to check whether an endpoint has been reached
eq reach_right_endp_truth : bool=(pos>=right_endp_truth);
eq reach_left_endp_truth : bool=(pos<=left_endp_truth);

--max_real returns the max of its arguments
guarantee "PR formula":
  PR = if meet_RN then PR_RN
       else if reach_right_endp_truth then 0.0
       else if pre_direction = DPSS_Constants.RIGHT then
            DPSS_Node_Lib.
              max_real(0.0, pre_PR - (pos - pre_pos))
       else DPSS_Node_Lib.
              max_real(0.0, pre_PR + (pos - pre_pos));

guarantee "NR formula":
  NR = if meet_RN then NR_RN + 1
       else if reach_right_endp_truth then 0
       else pre_NR;

guarantee "PL formula":
guarantee "NL formula":
  --Omitted for brevity; analogous to above

guarantee "Direction formula":
  direction = (
    --Turn around at the boundaries
    if reach_left_endp_truth then DPSS_Constants.RIGHT
    else if reach_right_endp_truth then DPSS_Constants.LEFT
    --If meeting a neighbor, travel to shared border
    else if meet_LN then
      if pos <= S_L then DPSS_Constants.RIGHT
      else DPSS_Constants.LEFT
    else if meet_RN then
      if pos < S_R then DPSS_Constants.RIGHT
      else DPSS_Constants.LEFT
    --In all other cases, proceed in the same direction
    else
      pre_direction
  );

guarantee "Goal formula":
  --Omitted for brevity. Sets goal to either S_R, S_L,
  --right_endp_est, or left_endp_est
```

Listing 4: Guarantees for the Algorithm B UAV model.

are used to make it easier to compute the UAV's estimated segment boundaries. Equations with the `prev` operator are used to store the UAV's previous direction and position, where `prev(x,y)` means use the value of `y` on the initial timestep and the value of `pre(x)` on all other timesteps.

2) System Model: Now consider the top-level system model. It computes the time `deltaT` until the next event using an equation. It also updates the positions of the UAVs, flags when UAVs are co-located, and passes the values of coordination variables between neighboring UAVs. It sends this information to the UAVs through their input data ports. It should be noted that assumptions of the system define

```
assume "time update": time = (0.0 -> pre(time) + deltaT);

assume "UAVs are numbered according to their position from
       left to right and do no start co-located":
(initial_pos_UAV1 < initial_pos_UAV2 and
 initial_pos_UAV2 < initial_pos_UAV3) -> true;

assume "Initial positions are between perimeter endpoints":
(initial_pos_UAV1 >= LEFT_ENDP_TRUTH and
 initial_pos_UAV2 >= LEFT_ENDP_TRUTH and
  ...
 initial_pos_UAV2 <= RIGHT_ENDP_TRUTH and
 initial_pos_UAV3 <= RIGHT_ENDP_TRUTH) -> true;

assume "Initial directions are LEFT or RIGHT":
((initial_direction_UAV1 = DPSS_Constants.LEFT or
  ...
  initial_direction_UAV3 = DPSS_Constants.RIGHT)) -> true;
```

Listing 5: Assumptions for the Algorithm B system model.

```
--Shared border positions
eq P_12 : real =
  1.0*P_TRUTH/N_TRUTH_REAL + LEFT_ENDP_TRUTH;
eq P_23 : real =
  2.0*P_TRUTH/N_TRUTH_REAL + LEFT_ENDP_TRUTH;

--Coordination variables are correct
--Values from UAV models obtained through data ports
eq correct_coordination_variables : bool =
  NL_UAV1 = 0 and NL_UAV2 = 1 and NL_UAV3 = 2 and
  NR_UAV1 = 2 and NR_UAV2 = 1 and NR_UAV3 = 0 and
  PL_UAV1 = pos_UAV1 - LEFT_ENDP_TRUTH and
  ...
  PR_UAV3 = RIGHT_ENDP_TRUTH - pos_UAV3;

eq optimal : bool = correct_coordination_variables and
  --Either the UAVs are at LEFT_ENDP_TRUTH, P_23, and P_23
  --and were previously at P_12, P_12, and RIGHT_ENDP_TRUTH
  (
    (pos_UAV1 = LEFT_ENDP_TRUTH  and
     pos_UAV2 = P_23 and pos_UAV3 = P_23)
    and
    (pre_pos_UAV1 = P_12 and pre_pos_UAV2 = P_12 and
     pre_pos_UAV3 = RIGHT_ENDP_TRUTH)
  )
  --or the UAVs are at P_12, P_12, and RIGHT_ENDP_TRUTH
  --and were previously at LEFT_ENDP_TRUTH, P_23, and P_23
  or (
    (pos_UAV1 = P_12 and pos_UAV2 = P_12 and
     pos_UAV3 = RIGHT_ENDP_TRUTH)
    and
    (pre_pos_UAV1 = LEFT_ENDP_TRUTH and
     pre_pos_UAV2 = P_23 and
     pre_pos_UAV3 = P_23)
  )
);
```

Listing 6: Key equations for the Algorithm B system model.

valid ranges for initial UAV positions and directions. An assumption is also used to update the cumulative running `time`. Assumptions for the system are shown in Listing 5.

The system also receives information from the UAVs through their output data ports, including the values of their coordination variables. It stores these and the current and previous values of the some of the inputs it sends to the UAVs, including UAV positions. It uses these to define an equation that checks whether all the UAVs' coordination variables are correct. It also defines an equation that checks whether the optimal configuration has been reached. These equations are shown in Listing 6 and are used in the system guarantees. The system guarantees correspond to the theorems of Section II and are discussed in the next section.

```
--Main Theorem
guarantee "Theorem 1 for Algorithm A for 3 Vehicles":
  (time >= ((2.0*DPSS_Constants.N_real - 1.0)/
            DPSS_Constants.N_real)*DPSS_Constants.T) =>
  (optimal and
   deltaT = DPSS_Constants.T/DPSS_Constants.N_real);

lemma "Time to optimal configuration is less than 2T":
  (optimal and not (pre(optimal))) =>
    (time < 2.0*DPSS_Constants.T);

lemma "Once in the optimal configuration, it stays there":
  true -> pre(optimal) => optimal;
```

Listing 7: Main Theorem and Lemmas for Alg. A for 3 UAVs

IV. FORMAL ANALYSIS RESULTS

In this section, we discuss the analysis results provided by AGREE for Algorithm A and Algorithm B.

A. Model Checking with AGREE for Algorithm A

The model for Algorithm A is similar to the model for Algorithm B, except that it starts with correct UAV coordination variables. Listing 7 shows Theorem 1 as a guarantee in AGREE, and two key lemmas regarding convergence.

The AGREE guarantee is phrased similarly to the original statement of Theorem 1, but there are some differences. First, the timestep `deltaT` between events varies in general. However, in the optimal configuration, `deltaT` should be T/N, so we include this in the guarantee. As a sanity check, we also include two lemmas that are closer to the original statement of Theorem 1. The first ensures that the first timestep on which the system is in the optimal configuration is less than 2T. The second checks that once the system is in the optimal configuration, it says there. Note that a user can experiment with the bound, and a counterexample will be returned if it is too low. In fact, we note that arguably the system is in the optimal configuration the very first time all UAVs reach one of the sets of locations defined in Definition 1, whereas our definition requires them to move between the two sets at least once, which takes time T/N. We could therefore increase the time bound in the guarantee and first lemma by T/N, but AGREE is able to prove the current time bound, which is tighter.

Using AGREE configured to utilize the JKind k-induction model checker [8] and the Z3 SMT solver [6], we have proven that the main theorem and key lemmas hold for N = 1, 2, 3, 4, 5, and 6 UAVs. Since model checking is exponential in the size of the model [10], analysis time prevented us from proving results for more than six vehicles. For reference, N = 1 through N = 4 ran in under 10 minutes each on a laptop with two cores and 8 GB RAM. Running overnight on the same laptop was sufficient for N = 5. For N = 6, the analysis took approximately twenty days to complete on a computer with 40 cores and 128 GB memory.

B. Model Checking with AGREE for Algorithm B

For Algorithm B, we were able to prove Theorem 2 for up to three UAVs, and in fact, we can show the convergence time is $4T$, which is significantly less than the originally stated $5T$. However, AGREE produced a counterexample to Lemma 1, which claims that every UAV obtains correct coordination variables within $3T$. Using the model checker, we experimented with the bound in the last lemma in Listing 8 and found the bound to be $(3 + \frac{1}{4})T$ for three UAVs.

A counterexample provided by AGREE shows the UAVs obtaining correct coordination variables in 3.0129T. Full details are available on GitHub[9],[2] but we outline the steps in Fig. 4. In this counterexample, UAV 1 starts very close to the left perimeter heading right, and UAVs 2 and 3 start in the middle of segment 3 headed left. UAVs 1 and 2 meet near the middle of the perimeter and head left toward what they believe to be their shared segment boundary. This is very close to the left perimeter endpoint because, due to initial conditions, they believe the left perimeter endpoint to be much farther away than it actually is. Then they split, and UAV 1 learns where the left perimeter endpoint actually is, but UAV 2 does not. UAV 2 heads right and meets UAV 3 shortly afterward, and they move to what they believe to be their shared segment boundary, which is likewise very close to the right perimeter endpoint because they believe it to be farther away than it actually is. Then they split, and UAV 3 learns where the right perimeter endpoint is, but UAV 2 does not. UAV 2 heads left, meets UAV 1 shortly after, and learns correct "left" coordination variables. However, UAV 2 still believes the right perimeter endpoint to be farther away than it actually is, so UAV 1 and 2 estimate their shared segment boundary to be near the middle of the perimeter. They then head toward this point and split apart, with UAV 1 headed left and still not having correct "right" coordination variables. UAV 2 and 3 then meet, exchange information, and now both have correct coordination variables. They go to their actual shared boundary, split apart, and UAV 2 heads left toward UAV 1. UAV 1 and 2 then meet on segment 1, exchange information, and now all UAVs have correct coordination variables.

The counterexample reveals a key intuition that was missing in Lemma 1. The original argument did not fully consider the effects of initial conditions, and so it only considered a case in which UAVs came close to *one* end of the perimeter without actually reaching it. The counterexample shows it can happen at *both* ends if the UAVs have initial conditions such that they believe the perimeter endpoints to be farther away than they actually are. This would happen if the perimeter suddenly shrinks and the UAVs must re-learn correct coordination variables.

Analysis for three UAVs completed in 20 hours on a machine with 256 GB RAM and 80 cores. Analysis for one and two vehicles completed in less than five minutes each on a laptop with 2 cores and 8 GB memory.

V. DISCUSSION AND CONCLUSIONS

Formal modeling and analysis through AGREE was a valuable exercise. First, it allowed us to analyze DPSS, a decentralized protocol for distributing a surveillance task across

[2]A spreadsheet showing the values of all model variables in the counterexample is located under AADL_sandbox_projects/DPSS-3-AlgB-for-paper/results_20180815_eispi.

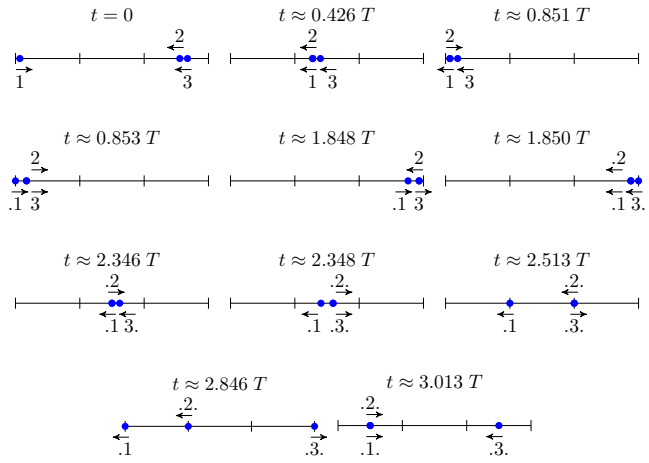

Fig. 4: Counterexample in which three UAVs take more than $3T$ to obtain correct coordination variables. Dots to the left of a UAV number indicate it has correct "left" variables, and likewise for the right.

```
--Main Theorem
guarantee "Theorem 2 for Algorithm B for 3 Vehicles":
  (time >= ((5.0*N_TRUTH_REAL - 1.0/N_TRUTH_REAL)*T) =>
    (optimal and deltaT = T/N_TRUTH_REAL);

lemma "Time to optimal configuration is less than 4T":
  (optimal and not (pre(optimal))) => (time < 4.0*T);

lemma "Time to correct coord. variables is < (3 + 1/4)T":
  (correct_coordination_variables and
    not (pre(correct_coordination_variables))) =>
  (time < (3.0 + 1.0/4.0)*T);
```

Listing 8: Main Theorem and Lemmas for Alg. B for 3 UAVs

multiple UAVs. Though the original publication on DPSS provided a convincing human-generated proof and extensive simulation results to support claims about its convergence time, analysis revealed that one of the key lemmas used in the proof was incorrect. Furthermore, the counterexample returned by AGREE provided insight into why the original lemma was incorrect. Second, formal modeling in and of itself allowed us to find mistakes in the original paper. For example, we found that the formula for dividing the perimeter across UAVs in the original paper only accounted for changes at the right end of the perimeter and not the left, so the location of the left endpoint, `left_endp_est`, had to be added to the shared border computation. We also discovered that certain key aspects of the protocol were underspecified. In particular, it is unclear what should happen if more than two UAVs meet at the same time. Analysis showed that this can occur for as little as three UAVs in Algorithm B. We therefore had to specify what happens in the case of three UAVs meeting at the same time. Here, we decided that if UAV 3 is to the left of its estimated segment, it immediately heads right and the other two follow the normal protocol to escort each other to their shared border. Otherwise, the UAVs all travel left together to the boundary between segments 2 and 3, then UAV 3 breaks off and heads right while the other two then follow the normal protocol. Simulations in the original paper on DPSS implement a more complex behavior where UAVs head to the closest shared boundary and then split apart into smaller groups, repeating the process for sub-groups until reaching the standard case of two co-located UAVs.

We are currently working to formalize a more general version of DPSS that covers the case that more than two UAVs meet at a time. Simultaneously, since model checking with AGREE can only handle up to three UAVs for Algorithm B, we are transitioning to using theorem provers such as ACL2 [11] and PVS [12] to develop a proof for DPSS. Theorem proving requires more human effort but does not have the high computational complexity of model checking. It should also allow us to reason parametrically about the protocol for N UAVs, where N is any finite integer value. We therefore hope to use theorem proving to prove the $5T$ convergence bound for Theorem 2 for a fully and formally specified version of DPSS, and it may be possible to prove the $4T$ convergence bound suggested by our analysis here.

Acknowledgment

We would like to thank John Backes for his guidance on efficiently modeling DPSS in AGREE and Aaron Fifarek for running some of the longer AGREE analyses.

References

[1] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[2] M. Fisher and M. Wooldridge, "On the formal specification and verification of multi-agent systems," *Int. J. Cooperative Information Systems*, vol. 6, no. 01, pp. 37–65, 1997.

[3] D. Kingston, R. W. Beard, and R. S. Holt, "Decentralized perimeter surveillance using a team of UAVs," *IEEE Trans. Robotics*, vol. 24, no. 6, pp. 1394–1404, 2008.

[4] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, "Compositional verification of architectural models," in *NASA Formal Methods Symp.* Springer, 2012, pp. 126–140.

[5] P. H. Feiler, B. A. Lewis, and S. Vestal, "The SAE Architecture Analysis & Design Language (AADL): A standard for engineering performance critical systems," in *IEEE Int. Conf. Computer Aided Control System Design*. IEEE, 2006, pp. 1206–1211.

[6] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[7] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu, *et al.*, "Bounded model checking." *Advances in Computers*, vol. 58, no. 11, pp. 117–148, 2003.

[8] A. Gacek, J. Backes, M. Whalen, L. Wagner, and E. Ghassabani, "The JKind model checker," in *Int. Conf. Computer Aided Verification*. Springer, 2018, pp. 20–27.

[9] OpenUxAS GitHub repository, architecture branch. [Online]. Available: https://github.com/afrl-rq/OpenUxAS/tree/architecture

[10] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman, "Completeness and complexity of bounded model checking," in *Inter. Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2004, pp. 85–96.

[11] M. Kaufmann and J. S. Moore, "An industrial strength theorem prover for a logic based on Common Lisp," *IEEE Trans. Software Engineering*, vol. 23, no. 4, pp. 203–213, 1997.

[12] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," in *Int. Conf. Automated Deduction*. Springer, 1992, pp. 748–752.