Formal Techniques for the Modelling and Validation of a Co-operating UAV Team that uses Dubins Set for Path Planning

Suresh Jeyaraman, Antonios Tsourdos, Rafał Żbikowski and Brian White
{S.Jeyaraman, A.Tsourdos, R.W.Zbikowski, B.A.White}@cranfield.ac.uk

***Abstract*— *Formal methods* have been deployed with great success to validate zero-fault tolerant systems such as hardware chips, real-time operating systems (RTOSs). Another feasible application area for formal techniques is in the modelling and verification of cooperative unmanned aerial vehicle (UAV) teams. The rationale being, multi-UAV coordination for cooperative control is a time-critical, zero-fault tolerant activity involving dynamic planning and real-time decision making. This provides sufficient incentive for designers to prove that the proposed system architecture will *work as advertised*. In this paper, a simulation scenario involving multiple UAVs for co-ordinated arrival at a specified target using Dubins' curves, is modelled using the *Kripke models* of "possible worlds". This formal model is then subjected to a proof verification technique known as *model checking*, for verifying the *safety, reachability, etc*. This novel framework is sophisticated enough to be reusable, and consequently, be able to address *scalability* issues.**

I. INTRODUCTION

UAV teams are a relatively recent addition to defence missions. They have been deployed frequently during conflicts as well as in peacekeeping missions, and are ideal for deployment in D^3 – *dull, dirty* or *dangerous* environments. The use of autonomous UAVs is actively being promoted, as they offer the following incentives: (a) Increased manoeuvrability with low human risk and (b) Significant weight savings on payload with low cost. However, standard operational behaviour for UAV teams involve the ability to replan flight paths dynamically either for interception of enemy or *kin* and decision making in the wake of uncertainty. Several approaches for achieving decentralised, autonomous, co-operative control among aerial robots are proposed in Butenko *et al* [1], Alighanbari *et al* [2], and Beard *et al* [3].

Systems of this nature effectively combine motion planning and decision control, and are therefore treated more effectively using hybrid control schemes. In hybrid control approach of modelling systems, the UAVs' motion is modelled using kinematics and the decision making that comes along with motion within a group is discretely modelled [4], [5]. In this paper, the formalised representation of a UAV system is achieved using *Kripke models* of *possible worlds*. From this representation, temporal logic statements of the system behaviour can be derived, and *model checking* can be performed on the entire system's *predicted* behaviour.

The authors are affiliated with the Department of Aerospace, Power and Sensors, RMCS, Cranfield University, Shrivenham SN6 8LA, United Kingdom.

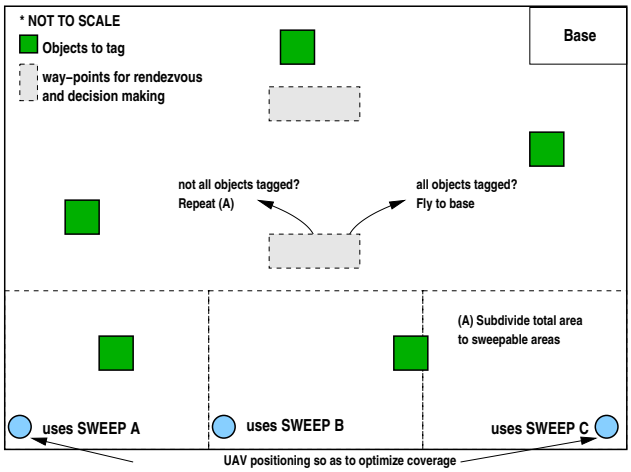

Fig. 1. Schematic representation of the enclosure.

Model checking is an enumerative, exhaustive check procedure that verifies reactive systems against their required specifications, giving *yes/no* answers to conformance [6].

In Jeyaraman *et al* [7], the feasibility of achieving co-operative control (co-ordinated arrival at destination) of multiple UAVs using Dubins' curves was established. In Shanmugavel *et al* [8], Dubins curves are used for non-intersecting paths among co-operating vehicles. The *novelty* in this paper will be the (a) co-operative area sweep by a group of UAVs by using a set of Dubins paths, in order to identify *artefacts* of interest and, (b) encapsulation of the entire approach in a formal approach framework.

In this scenario, the UAVs are assigned to sweep a fixed area of appropriately scaled rectangular dimensions (refer Fig. 1). In doing so, they must allocate themselves to specific areas on the map. This will demonstrate task allocation and avoid duplication of effort. The UAVs must identify the "specified" objects on their environment map and once all the objects have been tagged, they should fly straight back to the base. The UAVs are provided with information regarding their environment and have a circular sensor signature for detecting their *kin* and for communication. The communication between UAVs is assumed to be lossless and continuous. The UAVs co-ordinate their arrival at waypoints and exchange the list of tagged items in these designated waypoints. They should plan their paths in order to collect as much information as possible and move towards the target at the same time.

The remainder of the paper is organised as follows: Sec-

tion II details the modelling approach. Section III provides an overview of the formal modelling approaches. Section IV discusss the actual formal model. Section V deals with the the discussions of the formal model and Section VI presents our conclusions.

II. GEOMETRICAL MODEL OF UAV TEAM

A. Introduction

Consider a group of N vehicles ($N = 3$ in our case), moving in the $\mathbb{R} \times \mathbb{R}$ plane. The aerial vehicles are restricted to flying at constant altitude and therefore not required to perform pitch, roll or yaw motion. In this case, the kinematics of the vehicle can be abstracted to the familiar unicycle model (Fig. 1 (B)):

$$\left.\begin{array}{ll} \dot{x}_i & = v\cos\theta_i \\ \dot{y}_i & = v\sin\theta_i \\ \dot{\theta}_i & = \omega \end{array}\right\} \quad (1)$$

We fix all the UAVs to travel with the same linear velocity v such that $v = v_1 = v_2 = \ldots = v_N$.

The turning radius of the UAV R_{cur} (where $R_{cur} > 0$), can be chosen from a small finite set, i.e.,

$$R_{cur} \in \{R_1, R_2, \ldots, R_n\}, \text{ such that,}$$
$$\text{for } k \in [1,n],\ R_k - R_{k-1} = p \in \mathbb{N},$$

for all values in R_{cur}. From this, the turning speed of a UAV, when executing a turn, can be given as

$$|\omega| \leq \frac{v}{R_{cur}}$$

In other words, the UAV has the freedom of choosing a turning radius from the set of circles provided. This chosen turning radius remains constant throughout the UAV's manoeuvre. This allows for controlling the path lengths of UAVs accurately, while the choice in the turning radii allows the UAVs to somewhat *"wander"*, in order to collect information. For the purposes of this study, time is discrete.

B. Dubins' Curves

Dubins result has been widely studied in path planning [9], [10], [11]. Dubins result shows that, given any two points, the geodesic (i.e., the shortest path) consists of exactly three path segments and presents a sequence of circular arcs (CCC) or circular arcs with a connecting tangent between them (CSC). Each arc has two options – turning left or turning right. Denote these as L and R respectively, and the Dubins set can be written as $\mathscr{D} = \{LSL, RSR, RSL, LSR, RLR, LRL\}$. It has been shown previously that initial and final configurations define a sufficient set of 48 paths which contains the optimal path [12]. In this paper, we have implemented a subset of Dubins result for our scenario. The model enables one to select the shortest path from the Dubins set directly by using the concept of *equivalency groups*, based on the angle quadrants of the corresponding pairs of the initial and final orientation angles supplied to the UAV. Each equivalency group consists of a few classes of paths, such that any path in a group is equivalent, up to an orthogonal transformation, to any other path in the same group [9].

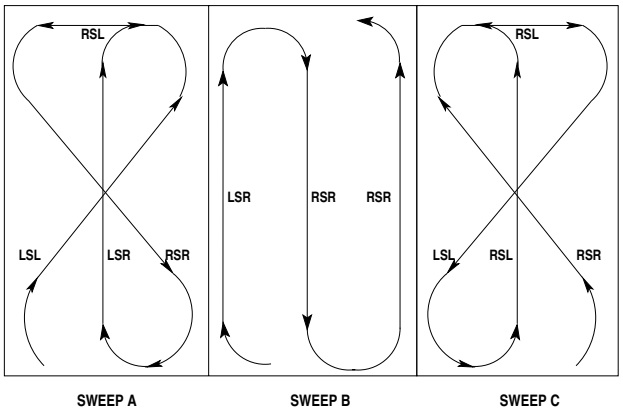

Fig. 2. Possible path sweeps that UAVs can make to cover a rectangular area.

A lookup table can be constructed from the set of solutions. The table is always dynamically populated and the values in each column are obtained based on the functions listed in Shkel and Lumelsky [9]. All the UAVs possess this lookup table at their disposal. The optimal path type for a given distance can be easily deduced from this table. The table relies solely on the initial and final orientation UAV at the waypoints. Readers are referred to Shkel and Lumelsky [9] for the equations regarding the constituent segments.

III. FORMAL MODELLING: INTRODUCTION

Formal modelling techniques originally evolved around the study of *reactive systems*. A reactive system/program is one that constantly maintains an interaction with its environment and is influenced by it. Therefore, its specification must be done in terms of its ongoing behaviour (that changes with time) [6]. This definition closely tallies with an informal description of a robot application, and many researchers have considered applying these techniques to studying robot applications. To date several formal modelling techniques such as Message Sequence Charts, Finite State Automata, Petri Nets, Kripke models and temporal logic have been proposed to specify, analyse, understand and verify the correctness of reactive systems. Some of these formalisms are more suitable than the others for representing multi-vehicle applications. (i) Petri nets and, more recently, (ii) hybrid modelling approach [13] (with hybrid automata representing both system interplay and the constituent component control variables) have gained immense popularity as formalisms of choice for representing reactive systems. A unique formalism in its own right, the hybrid automata approach is analogous to the Kripke modelling approach [14]. Kripke models are used for formal modelling purposes in this study.

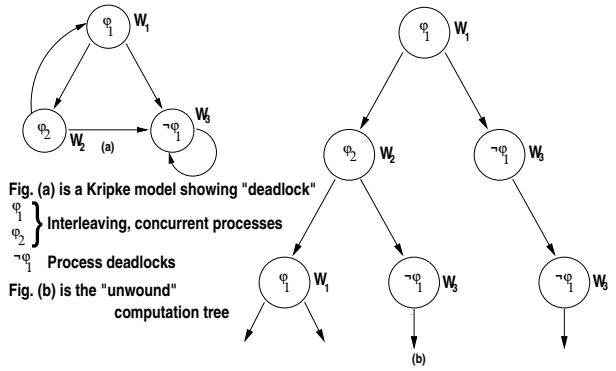

Fig. 3. To understand how Kripke models work in our case, view this figure in conjunction with Table I. Graphically, Kripke models are represented as directed, unlabelled graphs (as in Fig (a)). *Possible worlds*, capture overall system behaviour that evolves over time. *Labelling functions*, propositional formulae about conditions that hold true in each world. Lastly, *accessibility relations*, that maps the transitions (or interactions) between the worlds. Notions such as *uncertainty* and *decision making* can be modelling with this approach. This Kripke model can be "unwound" (see Fig (b)) into an infinite computation tree, which captures notions such as *computation sequence* and *system interplay*. Every Kripke model can be represented as ω-automata.

TABLE I

MODEL CHECKING PROPERTIES USING TEMPORAL LOGIC. RECALL FROM FIGURE 3 THAT ϕ_1, ϕ_2 REPRESENT "SPECIFIC" PROPERTIES OF A SYSTEM THAT CAN NOW BE VERIFIED BY A MODEL CHECKER

Property Definition	**Specification Formula: LTL**
reachability property — some particular property *can be reached*	Not Suitable: Expresses reachability negatively – nested reachability impossible
safety property — under certain conditions, something *never occurs*	$\Box\neg(\phi_1 \wedge \phi_2)$
liveness property — under certain conditions, something *will ultimately occur*	$\Box(\phi_1 \rightarrow \Diamond\phi_2)$
fairness property — under certain conditions, something will (or will not) occur *infinitely often*	possible using ω-automata

TABLE II

COMPANION TABLE FOR THE KRIPKE MODEL PROPOSED IN FIG. 4

Possible Worlds	Accessibility Relations	Possible Worlds	Accessibility Relations
$W_1 - W_2$	r_1, r_2	$W_5 - W_4$	r_{11}, r_{12}
$W_1 - W_3$	r_3, r_4	$W_5 - W_7$	r_{13}, r_{14}
$W_1 - W_4$	r_5, r_6	$W_5 - W_8$	r_{15}, r_{16}
$W_2 - W_5$	r_7, r_8	$W_4 - W_6$	r_{17}, r_{18}
$W_3 - W_5$	r_9, r_{10}	$W_5 - W_6$	r_{19}, r_{20}

A Kripke model can be *unwound* into an infinite computation tree along all the possible transition (execution) paths. The desired properties of the system are then expressed using temporal logic statements which must be true on the model (model checking). Temporal logic is a formalism for describing sequences of transitions between states in a reactive system. In temporal logic, different choices of operators and axioms lead to different models of time, with each model proving to be useful in a certain area of application. See [15] for a survey of different temporal logics. Basic Linear Temporal Logic (LTL) uses four special operators to represent time and infinity, without explicitly introducing either of them. These operators are *always* ($\Box$), *eventually* ($\Diamond$), *next* ($\bigcirc$) and *until* ($\mathscr{U}$). The names of the operators are self-explanatory and allow the modelling of processes and their occurrence in time. This "sequence of computation" is critical for real-time systems. Model checking is an automated procedure that exhaustively runs through all possible state traversals of a system and gives *yes/no* answers to system properties that have been extracted in temporal logic. The four common properties verified in model checking procedures are shown in Table I.

Kripke models are compact in their representation of a reactive system. In Fig. 4, not only does the representation capture the overall system behaviour, but also represents the rules that govern the transition between the various states (worlds). Finally, desirable system behaviour is tied into a formal Kripke model through the use of temporal logic formulae. This tied-in behaviour holds advantages that can be leveraged using *model checking* techniques.

IV. FORMAL MODEL: EXPLANATION

A. Remarks on Kripke modelling of UAV group

A group of UAVs can exhibit different levels of dynamics according to its organisation.

At the lowest level, each UAV can be treated as a single entity and therefore the group is a set of N distinguishable objects. If each of these objects has k individual states, there are k^N possible states for the group. The number of possible directed graphs for k^N nodes is very large even for small values of k and N.

From the point of view of the whole group, it may not be necessary to track the state of each individual UAV. The collective state need not be a concatenation of all individual states, so that the combinatorial explosion of possible models can be avoided. Such an approach is adopted in Section IV-B and is just an example of several such parsimonious possibilities.

Also, in following such an approach, Kripke models are that are derived for a certain scenario are independent of the number of UAVs participating in it. This feature and the abilitiy to formally verify a Kripke model can be used effectively in coping with *scalability* issues of cooperative control. We can easily and iteratively check our model by increasing the number of robots; if the model fails, one possibility could be the environment's inability to cater to any more UAVs, thereby leading to a *deadlock*. This can be determined by means of the error trace produced by the model checker. By using this approach, one can define

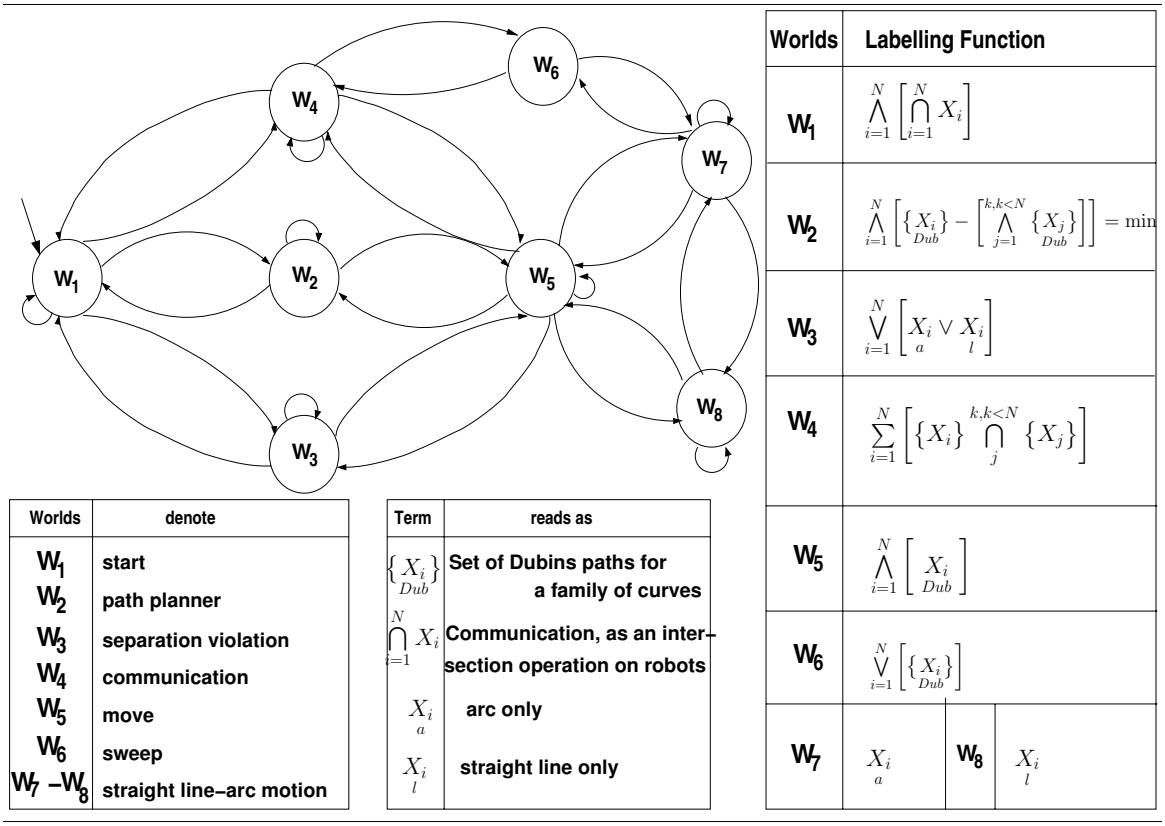

Fig. 4. Kripke Model for scenario.

an upper bound on the number of participants for a predetermined scenario.

B. Kripke model for the system

- $\{W_1, W_2, W_4\}$ are the set of worlds where all the robots perform their path plans and broadcast their respective path plans. In the path plan world, all the robots determine and agree upon paths that produce (as close to) identical path lengths as possible, before setting off towards goal.
- $\{W_3\}$ is the world, which models UAV behaviour, if any UAV violates the minimum separation that needs to be maintained in order to produce collision-free flight.
- $\{W_4\}$ is the world in the *"possible worlds"* universe that governs communication. Communication is modelled as a lossless transfer of state information between UAVs, which are subsequently compared to produce matching path lengths. Not all transfer of state information results in path length calculation. If, the UAVs enter W_4 during rendezvous, only the position information is utilised to maintain minimum separation (W_3 is traversed by a UAV, if minimum separation is violated).
- $\{W_5, W_6\}$ Kripke models can capture different facets of a multi-robot system. In, Jeyaraman *et al* [21], a case where Kripke models representing robotic behaviour can be folded into one another, in other words, subsumed Kripke worlds was discussed. Refer to [21] for more details. In this case, another important notion, i.e., worlds that are related to, but are unique enough to resist subsumption. In this case, W_6, the sweep world is a combination of Dubins paths. However, a move occurs, only when an area is unsweepable. Therefore, each behaviour has its own initiation mechanism. In such cases, the worlds are not subsumed but kept as unique worlds.
- $\{W_7, W_8\}$ are worlds in which the UAVs motion (arc and straight line) is mapped as a "possible" world. Transition is governed based on factors such as minimum distance violation, proximity to goal, keeping formation and so on.

Table II tabulates the transition functions as pairs, corresponding to the worlds that they "link". The functions themselves are defined presently.

❑ The labelling formula for W_2 can be explained as follows:

$$\bigwedge_{i=1}^{N}\left[\{X_i\}_{Dub} - \left[\bigwedge_{j=1}^{k,k<N}\{X_j\}_{Dub}\right]\right] = \min$$

Every UAV i shares its state information with a set of immediate neighbours k, such that the most minimal of the path lengths from a set of path lengths generated for manoeuvre by UAV_i, is earmarked for implementation.

❑ Similarily, in the case of W_4,

$$\sum_{i=1}^{N}\left[\{X_i\}\bigcap_{j}^{k,k<N}\{X_j\}\right]$$

Every UAV i shares its state information with a set of immediate neighbours k, with the requirement that, if there is "common" information between the group, each robot *must* utilise it in some form to its advantage. The other formulae provided in Fig. 4 can be parsed likewise, with help from the legend table provided.

❑ $(W_1 - W_2)$: Similarily, one can also fix the transition functions that govern transitions between the different worlds. For instance, transition between W_1 and W_2, governed by r_1 and r_2 are represented as follows:

$$r_1 = \begin{cases} true & \text{if } \bigwedge_{i=1}^{N}\left[\bigcap_{i=1}^{N} X_i\right] \text{ or} \\ & \text{goal/target specified} \\ false & \text{otherwise} \end{cases} \quad (2)$$

$$r_2 = \begin{cases} true & \text{if } \{X_i\}_{Dub} \approx \min_{Dub} X_i \\ false & \text{otherwise} \end{cases} \quad (3)$$

❑ $(W_5 - W_6)$: When a UAV is making a sweep search of an area it uses a combination of Dubins paths in order to smooth its paths (Refer to Fig. 2). Even though a "sweep" is a series of "moves", a move is initiated only when there it is unnecessary or, even impossible to perform a sweep. Therefore the transition functions that initiate a *move* are very different to the ones that will govern a *sweep* world.

$$r_{19} = \begin{cases} true & \text{iff } \{\text{feasible Dubin paths}\} \\ false & \text{otherwise} \end{cases} \quad (4)$$

$$r_{20} = \begin{cases} true & \text{area swept } \| \text{ no feasible sweeps} \\ false & \text{otherwise} \end{cases} \quad (5)$$

$$(6)$$

V. Model Checking

In Holzmann [16], [17] and in Holzmann and Smith [18], [19], a test harness model that can bind the implementation ANSI-C code for model checking is proposed. The idea behind model checking *per se* is that, by capturing the essence of the design in a mathematical model (Kripke model), we can often demonstrate conclusively that the design has certain inevitable properties. The purpose of a verification model, then, is to enable *proof*. If it fails to do so, within the resource limits that are available to the verification system, the model should be considered inadequate. A similar approach was undertaken in Havelund *et al* [20], and was successful in verifying the software's correctness as well as uncovering subtle bugs. The mechanised harness method is described briefly below:

```
%F mainmod2.c
%X -L mainmod.lut -n main
%%
%L
headChan... skip
polarChan... skip
commonPad... skip
/*more data here*/
%%
%P
#define ROBOTS 3
#define STEPS_TO_RUN 50
/*remapping C datatypes to PROMELA datatypes*/
mtype {R_STOP, R_BROADCAST, R_PATHPLAN, R_MOVE};
mtype {L_SEG, R_SEG, S_SEG};
/*more entries..*/
```

Fig. 5. Manually generated FeaVer harness file. Used by model extractor to create an equivalent PROMELA file that serves as input for SPIN.

- A test harness file is created, which contains the name of the C file(s) to be tested along with test drivers and the like.
- A mapping table for mapping the data within the source into the model checker's language is also defined.
- Determine the desired selection for the test harness from the source
- Generate model checking code
- Run it in verification mode by compiling the source file generated by the model checker.

A sample from the test harness file derived for this purpose, is shown in Fig. 5. By this extraction process, we derived a complete and syntactically sound PROMELA model of around 200 lines. The purpose of the code was to ascertain that the execution of our program is deterministic and round-robin, despite the absence of explicit thread scheduling. By successfully conducting the *safety and reachability* tests, we can effectively say that stability analysis has been performed on the system [21]. The PROMELA code was compiled and run under simulation and verification mode. Program's output listing is provided in Fig. 6. A snippet from the PROMELA code, automatically derived from the C code (using the harness file), is provided in Fig. 7. Due to lack of space, other similar model checking properties like deadlock detection are not mentioned here.

VI. Conclusions

We have proposed a novel hybrid control scheme that brings together the formalised modelling approach, while focussing on the implementation aspects of the model. To this effect, we have utilised the Dubins set to implement

```
(Spin Version 4.1.0 -- 6 December 2003)
+ Partial Order Reduction

Bit statespace search for:
never claim          - (none specified)
assertion violations +
cycle checks         - (disabled by -DSAFETY)
invalid end states - (disabled by -E flag)

State-vector 140 byte, depth reached 9, errors: 0
      10 states, stored
       0 states, matched
      10 transitions (= stored+matched)
       0 atomic steps
hash factor:1.5252e+06(expected coverage:>=99.9% on avg)
(max size 2^24 states)

Stats on memory usage (in Megabytes):
0.001  equivalent memory usage for states
(stored*(State-vector + overhead))
4.194  memory used for hash array (-w24)
0.036  memory used for DFS stack (-m1000)
4.399  total actual memory usage
```

Fig. 6. Output listing of SPIN for full statespace checks through FeaVer.

```
inline runRobot(rob, mesg, paths) {
/*non-deterministic FSM*/
do
:: full(mesg) ->
if
:: (full(mesg) && rob.State == R_BROADCAST) ->
assert(full(mesg));
rob.State = R_PATHPLAN;
break
:: (len(mesg) < ROBOTS && rob.State == R_BROADCAST) ->
mesg[rob.robUID]!1;
break
:: (full(paths) && rob.State == R_PATHPLAN) ->
assert(full(mesg));
rob.State = R_MOVE;
/*paths[rob.robUID]!0;*/
break
:: (len(paths) < ROBOTS && rob.State == R_PATHPLAN) ->
paths[rob.robUID]!1;
break
fi;
/*more code here...*/
od
}
```

Fig. 7. PROMELA code snippet derived from the C source for this study. Harness file syntax is provided in Fig. 5.

co-operative control among a team of UAVs while mechanically proof-checking the stability of our system at the same time. Future work will include re-implementation of the program in GNU threads and scaling up the number of UAVs in the scenario and observing if the Kripke model as well as the model checker, can cope with them.

REFERENCES

[1] S. Butenko, R. Murphey, and P. M. Pardalos, Eds., *Cooperative Control: Models, Applications and Algorithms*. Kluwer Academic Publishers, 2003, vol. 1.

[2] M. Alighanbari, Y. Kuwata, and J. P. How, "Coordination and Control of Multiple UAVs with Timing Constraints and Loitering," in *Proc. of the American Control Conference*, vol. 6, June 2003, pp. 5311–5316.

[3] R. W. Beard, T. W. McLain, and M. Goodrich, "Coordinated Target Assignment and Intercept for Unmanned Air Vehicles," in *Proc. of the IEEE International Conference on Robotics and Automation*, vol. 3, May 2002, pp. 2581–2586.

[4] R. Alur, J. Esposito, M. Kim, V. Kumar, and I. Lee, "Formal Modeling and Analysis of Hybrid Systems: A Case Study in Multirobot Coordination," in *Proceedings of the World Congress on Formal Methods*, ser. Lecture Notes in Computer Science 1708. Springer – Verlag, 1999, pp. 212–232.

[5] R. Alur, A. Das, J. Esposito, R. Fierro, *et al.*, "A Framework and Architecture for Multirobot Coordination," in *Seventh International Symposium on Experimental Robotics*, December 2000.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, 2001.

[7] S. Jeyaraman, A. Tsourdos, R. Żbikowski, B. White, *et al.*, "Formalised Hybrid Control Scheme for a UAV Group using Dubins Set and Model Checking," in *Proc. of the 43rd IEEE Conf. on Decision and Control*, Paradise Island, Bahamas, 14-17 December 2004.

[8] M. Shanmugavel, A. Tsourdos, R. Żbikowski, and B. White, "Path Planning of Multiple UAVs using Dubins Sets," in *To appear in the AIAA Guidance, Navigation, and Control Conference*, San Francisco, California, 15-18 August 2005, Paper submitted for review.

[9] A. M. Shkel and V. Lumelsky, "Classification of the Dubins Set," *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–274, March 2001.

[10] A. Balluchi, P. Souères, and A. Bicchi, "Hybrid Feedback Control for Path Tracking by a Bounded-Curvature Vehicle," in *Hybrid Systems: Computation and Control (HSCC)*, ser. Lecture Notes in Computer Science 2034, March 2001, pp. 133–146.

[11] T. Fraichard, A. Scheuer, and R. Desvigne, "From Reeds and Shepp's to Continuous-Curvature Paths," in *Proceedings of the International Conference on Advanced Robotics*, October 1999, pp. 585–590.

[12] J. A. Reeds and L. A. Shepp, "Optimal Paths for a Car that Goes Both Forwards and Backwards,," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[13] T. A. Henzinger, "The Theory of Hybrid Automata," in 11th *Annual IEEE Symposium on Logic in Computer Science (LICS 96)*, 1996, pp. 278–292.

[14] R. Huuck, B. Lukoschus, G. Frehse, and S. Engell, *Modelling, Analysis, and Design of Hybrid Systems*, ser. LNCIS 279. SV, 2002, ch. Compositional Verification of Continuous-Discrete Systems, pp. 225–244.

[15] B. Bérard, M. Bidoit, A. Finkel, *et al.*, *Systems and Software Verification*. Springer – Verlag, 2001.

[16] G. J. Holzmann, *The* SPIN *Model Checker*. Addison-Wesley, 2003.

[17] ——, "Logic Verification of ANSI-C Code with SPIN," in *Proceedings of the 7th International SPIN workshop*, ser. Lecture Notes in Computer Science 1885, Sep. 2000, pp. 131–147.

[18] G. J. Holzmann and M. H. Smith, "An Automated Verification Method for Distributed Systems Software Based on Model Extraction," *IEEE Transactions on Software Engineering*, vol. 28, no. 4, pp. 364–377, April 2002, ISSN 0098-5589.

[19] ——, *FeaVer 1.0 User Guide*, 1st ed., Lucent Technologies Inc, 2002.

[20] K. Havelund, M. Lowry, and J. Penix, "Formal Analysis of a Space-Craft Controller using SPIN," *IEEE Transactions on Software Engineering*, vol. 27, no. 8, pp. 749–765, August 2001, ISSN 0098-5589.

[21] S. Jeyaraman, A. Tsourdos, R. Zbikowski, and B. white, "Kripke modelling of multiple robots with decentralized cooperation specified with temporal logic," *Part I: Journal of Systems and Control Engineering*, vol. 219, 2005, DOI:10.1243/095965105X9506.