Formal Methods in Resilient Systems Design using a Flexible Contract Approach

Technical Report SERC-2018-TR-119

December 21, 2018

Principal Investigator: Dr. Azad M. Madni, USC

Co-Investigator: Dr. Dan Erwin, USC

Research Team:

Dr. Ayesha Madni, USC

Edwin Ordoukhanian, USC

Parisa Pouya, USC

Sponsor: DASD(SE)

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract HQ0034-13-D-0004.

Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense nor ASD(R&E).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Resilience is a much-needed characteristic in systems that are expected to operate in uncertain, disruptive environments for extended periods. Resilience approaches today employ ad hoc methods and piece-meal solutions that are difficult to verify and test, and do not scale. Furthermore, it is difficult to assess the long-term impact of such ad hoc "resilience solutions." This research presents a flexible contract-based approach that employs a combination of formal methods for verification and testing and flexible assertions and probabilistic modeling to handle uncertainty during mission execution. A flexible contract (FC) is a hybrid modeling construct that facilitates system verification and testing while offering the requisite flexibility to cope with non-determinism. This research illustrates the use of FCs for multi-UAV swarm control in partially observable, dynamic environments. However, the approach is sufficiently general for use in other domains such as self-driving vehicle and adaptive power/energy grids.

INTRODUCTION

Resilience, a non-functional property, allows a system or system of systems (SoS) to continue to provide useful service in the face of disruptions (Neches and Madni, 2011). Disruptions can be external, systemic, or human-triggered (Madni and Jackson, 2009). Examples of disruptions in the operational context of multi-UAV swarms include a hacked or compromised swarm member, loss of communication within the swarm or between specific swarm members, and loss of visibility due to extreme weather or sensor malfunction. Resilient responses to such disruptions can take a variety of forms depending on environment observability and available intelligence. These include: circumvent disruptions if they can be anticipated; withstand disruption if within designed performance envelope; and recover rapidly from the negative effects of disruptions outside the performance envelope. Practically speaking, this means dynamically extending system capacity to cope with disruptions; restructuring or reconfiguring system pursuant to disruptions; and continuing to operate at somewhat diminished but acceptable level. The system's design envelope includes system models and adaptation logic incorporated within the system model to produce the requisite resilient responses when such disruptions occur. Doyle (2016) defines resilience as the ability to recognize unanticipated perturbations that fall outside the system's design envelope. This definition implies that resilience is concerned with monitoring the boundary conditions of the system's model for competence (how well resilience strategies match disruption demands), and then adjusting or expanding that model to better accommodate changing demands (Neches and Madni, 2011). The key issue here is assessing an organization's adaptive capacity (i.e., resource buffers that allow resources of a particular type to be increased on demand to a maximum limit) relative to the challenge posed by the disrupting event to that adaptive capacity. Boundaries in the multi-UAV swarm context define the system's competent performance envelope relative to specific classes of disruptions and uncertainties. Therefore, resilience engineering in a certain sense is concerned with introducing transparency into an organization's safety model with the purpose of determining when the model needs to be revised. In other words, resilience engineering is concerned with monitoring a system's decision making with a view to assessing the system's risks and risk envelope relative to unsafe operating boundaries.

Risk monitoring implies proactive and automatic/semi-automatic monitoring of buffers, margins, and tolerances. Buffer capacity is concerned with the magnitude and type of disruptions a system can absorb or adapt to without a substantial degradation in system performance or breakdown in integrity of system structure. *Flexibility* is the ability of a system to restructure or reorganize itself in response to external changes or pressures (Madni, 2009). *Margin* is the proximity of a system's operation regime relative to its designed operational performance envelope or boundary. *Tolerance* is the ability of a system to degrade gracefully (as opposed to collapsing) as stress/pressure increases, or when disruption magnitude and/or severity exceeds its adaptive capacity.

This paper presents a model-based approach that combines formal and probabilistic modeling to engineer resilient system and verify their designs.

TECHNICAL CHALLENGES

There are several technical challenges that have to be overcome in developing formal methods for the engineering of resilient systems including choosing the right system modeling construct, the right technology platform for development and demonstration, and the right application domain. These considerations are discussed next.

Application Domain. We chose UAV swarm control as our application domain. A UAV swarm is a system-of-system (SoS) in which the elements can be either homogeneous or heterogeneous. The elements in the SoS cooperate to perform their assigned mission or mutually agreed to tasks, and coordinate as needed. Each UAV in the swarm is equipped with sensors and communication facilities. UAV swarms are used in a variety of missions in the military and civilian sector. Exemplar missions include search and rescue, reconnaissance and surveillance, humanitarian assistance, and disaster relief.

System Modeling Construct. Selecting the right modeling construct is a key challenge. The model needs to be semantically expressive, scalable, amenable to verification, and sufficiently flexible to support mechanisms needed to handle non-determinism.

Technology Platform. The technology platform for this effort needs to support SoS specification and visualization, deterministic and probabilistic modeling, and integration with analytics and reporting modules (Figure 1).

As shown in this figure, the architecture is layered with each layer assigned to a particular model type. There are different types of models associated with this small SoS: vehicle physics model (Appendix B); probabilistic Partially Observable Markov Decision Process Model and simulated/actual data sources (Figure 1). Each is discussed next.

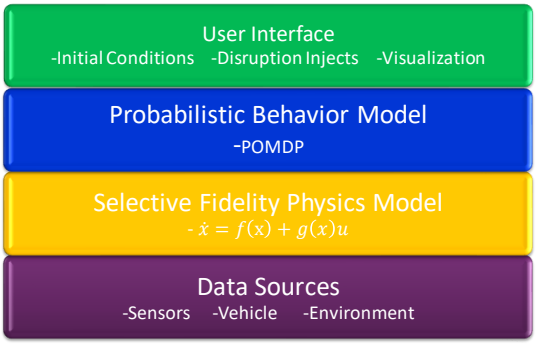


Figure 1. Technology Platform Employs a Layered System Architecture

FORMAL MODELING OF SYSTEMS AND SOS

Formal modeling introduces rigor in system verification, testing, and reasoning. However, formal modeling has limitations. The rigor in formal modeling comes at the expense of flexibility. Ideally, one wants a degree of formality to support model verification and testing, and sufficient flexibility to scale and cope with uncertainty. This recognition provided the impetus for this research.

Our modeling approach extends the concept of a "contract" in Contract-Based Design (CBD) to address uncertainty and partial observability that contribute to non-deterministic system behavior (Madni, 2015; Sievers, 2014). CBD is a formal method for explicitly defining, verifying and validating system requirements, constraints and interfaces. An implementation satisfies a design contract if it fulfills guarantees when assumptions are true. This is the "assert-guarantee" construct used in CBD. The rationale for choosing CBD is that statements in contracts are mathematically provable. The limitation of a traditional contract or CBD is that the assertions are invariant. The key innovation in our approach is the relaxation of invariant assertions requirement to introduce flexibility in the contract. The resulting "resilience contract (RC)" is a hybrid modeling construct that combines traditional contract, and flexible assertions, with Partially Observable Markov Decision Processes (POMDP). A POMDP is a special form of a Markov Decision Process that includes unobservable states and state transitions that are trained during system use. POMDPs introduce flexibility into a traditional contract by allowing incomplete specification of legal inputs and flexible definition of post-condition corrections (Madni, 2015; Sievers, 2014). A resilience contract extends a deterministic (i.e., traditional) contract for stochastic systems.

Figure 2 shows a hierarchical resiliency model using SysML block definition notation. The system comprises two subsystems as shown. Each subsystem and the system have individual RCs that comprise parameters and operations associated with its POMDP. As described below, RCs are software agents that update a belief state and determine the next action based on observing element outputs, the current belief state, the transition probabilities associated with the current state, and the reward (or penalty) for taking a given action. A belief state represents an entity's most probable state.

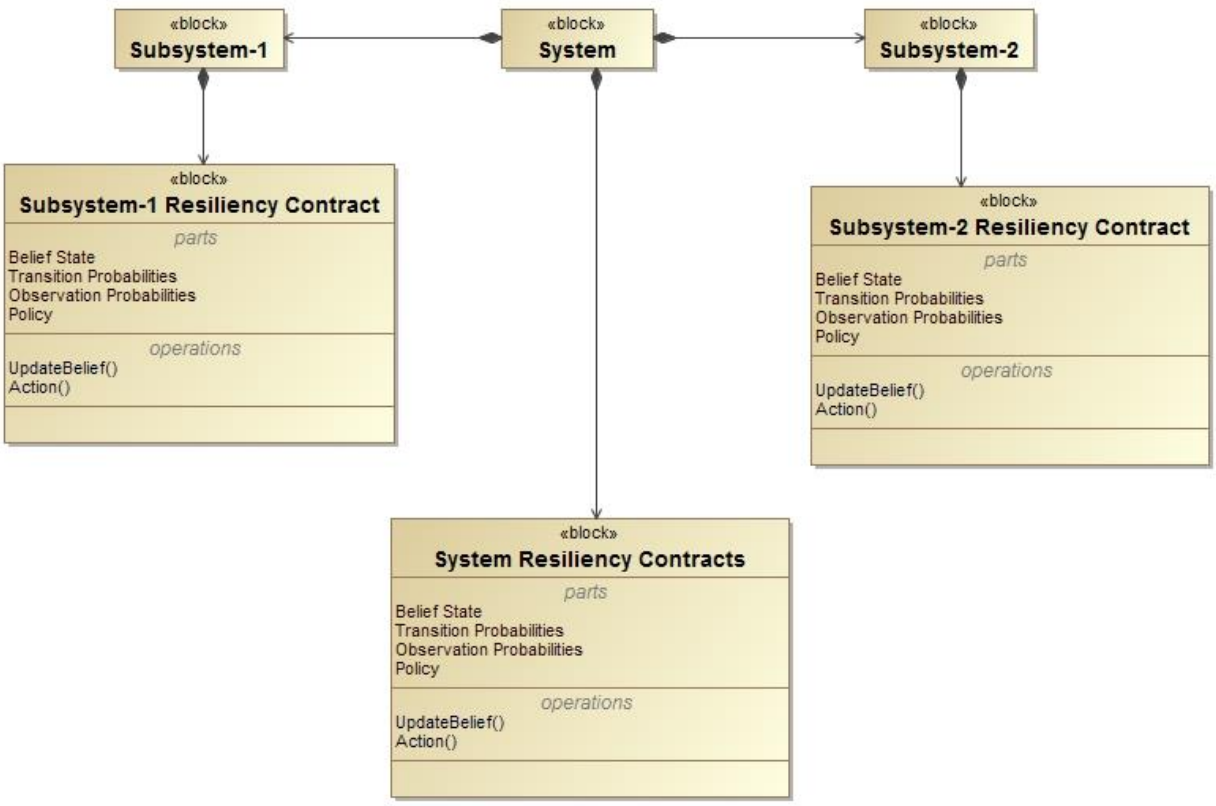


Figure 2. Resiliency Model

The assertions associated with a resilience contract are flexible, and the techniques employed include in-use learning, uncertainty handling, and pattern recognition. A RC is developed at design time and trained during system use ("learning"). It allows trading of model verification for model flexibility, and vice versa.

Contract flexibility is introduced by: relaxing the time invariance restrictions on the state space and action space, adding evaluation metrics for determining best action, and updating emission and transition probabilities of hidden states. By replacing the "assert-guarantee" construct with a "belief-reward" construct, a traditional contract can be made flexible without compromising model verification to an appreciable degree.

UAV SWARM MODELING AND ILLUSTRATIVE EXAMPLE

For UAV swarm modeling, we employ a combination of methods. To convey the key modeling concepts, we employ a swarm of quadcopters as our illustrative example. To model and evaluate system and SoS resilience, the questions that need to be answered pertain to model fidelity, model verifiability, and model flexibility. *Fidelity* pertains to the depth of modeling and the perspectives needed to answer the questions posed. *Verifiability* pertains to model correctness analysis. *Flexibility* pertains to the ease of extending or augmenting the model with reasoning mechanisms that introduce various forms of resilience. Ideally, we want just enough fidelity, and

adequate flexibility to respond to disruptions. At the single UAV level, just enough flexibility means rudimentary dynamics of the UAV (i.e., quadcopter), basic sensor model and a basic collision avoidance algorithm. The model could be run offline to generate parametric curves that could then be used to accept commands from the probabilistic model and generate new locations that can be used by the graphic visualizations. The model needs to support waypoint navigation and trajectory following. And, the model should be easily replicable to realize SoS behavior.

At the SoS level, the model needs to support different missions, communication protocols, and SoS configurations. The model should be capable of reflecting the behavior of hacked or compromised UAV in the SoS, loss of communication, loss of a UAV, loss of sensing, and malfunctioning SoS member. At both the individual UAV level and the swarm level, it should be possible to evaluate different resilience concepts. Figure 3 shows a UAV-SoS Concept of Operations that informs system and SoS modeling.

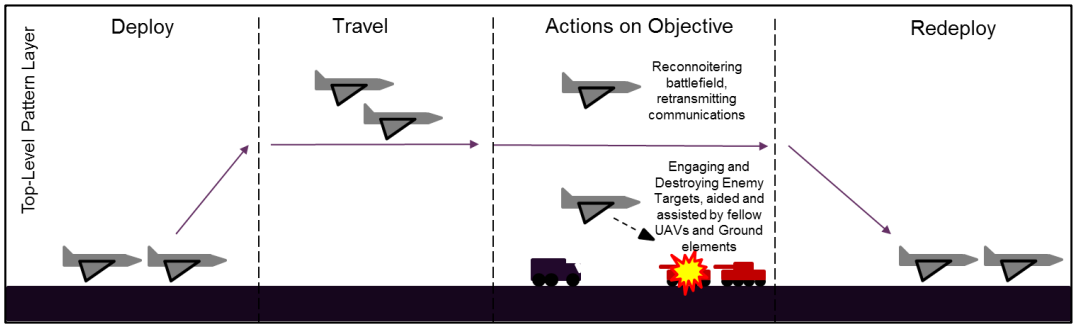


Figure 3. UAV SoS CONOPS

BEHAVIORAL PATTERNS

UAV swarm behaviors can be conveniently grouped into four behavior patterns: deployment; enroute; action on objective; and redeployment. Each behavior pattern, associated with a mission phase, is discussed next.

Deployment (or takeoff) pattern: act of putting SoS into operation. UAVs initiate operations and take flight. Variations in Pattern come in the form of: Takeoff Method: Vertical (VTOL), Horizontal or Conventional (CTOL), Assisted (Mechanical or Human Catapult, piggybacked from aircraft, propulsion assistance for short takeoff), etc.; Takeoff Order: Sequential vs. Parallel; Swarm Size, Hierarchy, and Homogeneity; Mission: new, clean sheet deployment, or are UAVs reinforcing another UAV swarm; Platform: airfield, airport, grass field, naval ship, and improvised (such as a road or building top). The key factors affecting operation are characterized by Mission-Enemy-Troops-Terrain-and-Weather-Time Available-Civilian (METT-TC). An example of METT-TC factor is “enemy has robust air-defense in area necessitating unique flight maneuvers on takeoff.”
Enroute (or cruise) pattern: act of deployed swarm flying from one location to another in pursuit of overall mission. UAV SoS objectives are: navigate as appropriate in support of global mission, pathfind at a local level, and maneuver through terrain, weather, other UAVs in SoS, and

neighboring systems not a part of SoS (e.g., coalition aircraft, enemy aircraft, noncombatant aircraft), as well as make trade-offs in pathfinding and navigating using attributes of METT-TC. Variations in Pattern comes in the form of Tactical Flight Considerations; high altitude vs. mid-altitude vs. NAP of the earth vs. a combination; formation and disposition during cruise; swarm size, composition, and capabilities (swarm heterogeneity factors); enemy air defense capabilities and presence; and weather.

Actions on Objective pattern: key part of overall CONOPS. Swarm achieves commander's intent and mission purpose (e.g. Reconnaissance, Observation, Sensing, Collecting, Aerial communications retransmit) Kinetic examples are: destroy enemy assets; and neutralize enemy unit. UAV SoS Objectives can be tactically addressed at a local level both as individual systems and as a swarm to successfully execute actions on objectives and deploy UAV Systems as a SoS to achieve desired tactical and operational objectives in the battlespace. Variations in Pattern - highly METT-TC dependent; examples: Coordinated payload delivery to destroy a bridge and conduct recon; Battlefield sensing and communications retransmission to support a focused, ground-based operation; Routine mapping and imagery collection; Search & Rescue operation to locate downed aircraft in suspected geographical "crash window".

Redeployment pattern: act of safely taking SoS out of operation. UAVs must RTB (return to base) and land, while preserving themselves and collected data (if held onboard). Variations of the Pattern include: Landing Method: Vertical (VTOL), Horizontal or Conventional (CTOL), Assisted (tail hook and cable, parachute landing or drag chute once landed), Landing Order: Sequential vs. Parallel, Swarm Size, Hierarchy, and Homogeneity, Mission: new, clean sheet deployment or are UAVs reinforcing another UAV swarm, Platform: airfield, airport, grass field, naval ship, improvised (e.g., a road or building top), Other METT-TC factors: e.g., enemy has robust air-defense in area necessitating unique flight maneuvers on landing, Hasty landing: e.g., a damaged UAV improvises and lands in a clear area and sends out a distress signal.

Each basic pattern can adapted and be decomposed into multiple more nuanced, specific scenarios using METT-TC considerations that apply to the SoS mission. Fundamental concepts for top layer patterns are adapted and developed for highly specific use cases (e.g., fundamentals of an attack apply, but tactics behind attacking an enemy tank column vary - in the open versus enemy ground troops in wooded mountains). The right level of decomposition and detail for each top-level pattern help answer questions about where to introduce resilience and how best to incorporate resilience logic/reasoning within the SoS.

Figure 4 shows the state transition diagram for a quadcopter. In this figure, some transitions are labeled with belief values, e.g., b (failed) ≥ 0.95 is threshold of transition from "normal motors" to "failed motor" i.e. transition happens if belief ≥ 0.95 that a motor has failed. Some transitions have fixed assertions, e.g., failed Motor and Operational, Transition from "Evaluate Environment" to "Auto Plan Enabled" has three beliefs with different probabilities in our example. Auto Planner determines the course of action to take based on environment beliefs, motor condition beliefs, and the goals (action taken is the one that maximizes reward or minimizes penalty).

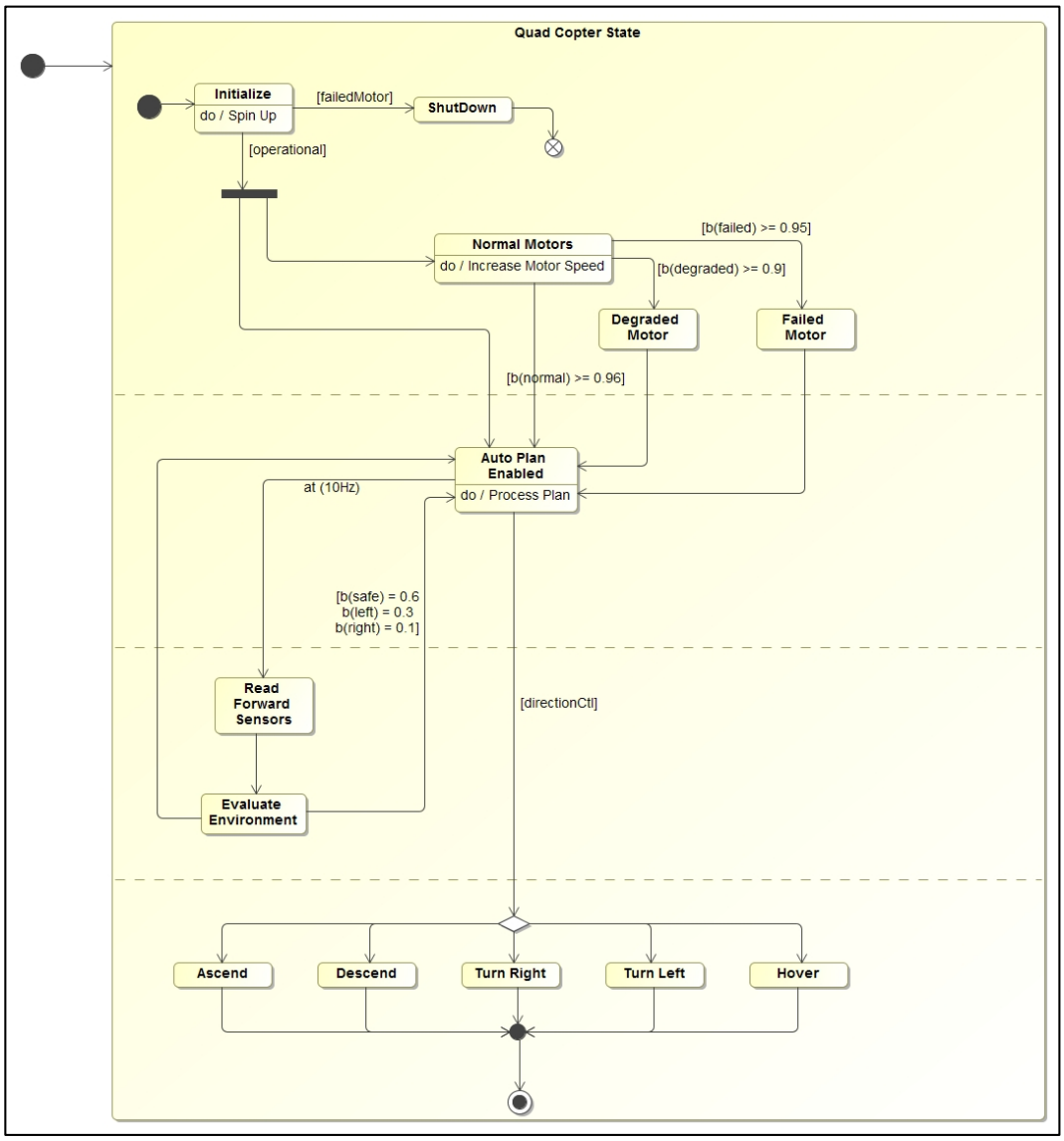


Figure 4. State Transition Diagram for Multi-UAV SoS

A UAV swarm can be viewed as a system-of-systems (SoS) because multiple UAVs need to cooperate to accomplish an end-to-end mission. A UAV swarm, especially a heterogeneous swarm, exhibits the characteristics of a SoS (Table 1).

Table 1. UAV Swarm Exhibits Characteristics of a SoS

- **Operational Independence of UAVs**
  - UAVs operate independently to satisfy mission requirements
- **Managerial Independence of UAVs**
  - each governed independently while being part of swarm
- **Evolutionary Development of SoS**
  - functions and purposes added / removed / modified with experience and need
- **Emergent SoS Behavior**
  - SoS performs functions that do not reside in any single UAV
  - emergent behavior cannot be realized by a single UAV
- **Geographic Distribution**
  - UAVs are displaced in space and time and primarily exchange information

UAV SWARM CONTROL ARCHITECTURE AND CONOPS

Figure 5 presents swarm control architecture based on creating an optimal policy based on belief estimates provided by the state estimator. The state estimator relies on observations from the UAV swarm, environment sensors, and MDP belief model to generate updated belief estimates. Policy actions act on the UAV swarm and are used by the state estimator to update state information.

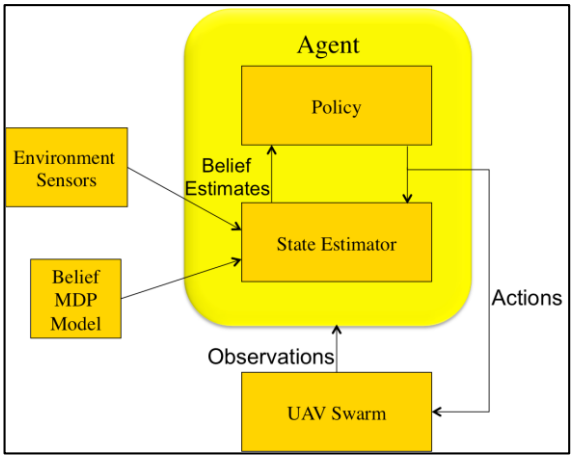


Figure 5. Example Swarm Control Architecture

A simple example is used to convey the key ideas of UAV swarm control. In this example, the UAV swarm needs to turn either left or right to avoid an obstacle. There is uncertainty regarding the location of the threat, in that the threat could be to the left or the right of the swarm. A decision needs to be made to veer left or veer right. If the swarm veers right and the threat is located/headed to the right, serious consequences could ensue. The same is true if the swarm veers left and the threat is heading left. There are three possible actions that the swarm can take: veer left; veer right; continue flying straight ahead and collect more observations on the threat.

POMDP policy for this simple concept of operations (CONOPS) has to deal with considerations such as: UAVs not crashing into each other; all UAVs getting safely to their destination; UAVs avoiding potentially disruptive events; if one or more UAVs is shot down, the remaining UAVs reorganize and reallocate functionality to ensure accomplishment of mission objective to the extent feasible. The key ideas behind an optimal POMDP policy are two-fold: a POMDP policy maps current belief into an action; and an optimal POMDP policy is a continuous solution of a belief MDP. Figure 5 shows the equation for summation of outcomes based on the path the UAVs take. The equation normalizes the rewards and penalties. As shown in Figure 6, the system starts with a 50-50 belief that the threat could be to the left or the right. The system notices a potential threat to the left. So, the system moves its belief to the left as shown in the figure. That is, there is a greater belief that the threat could be to the left. Also, the system does not observe anything to the right. Thus, belief is updated in accord with Bayesian analysis using observation and current state.

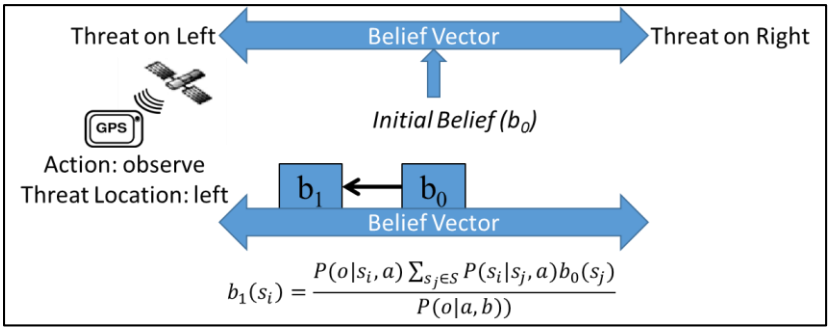


Figure 6. Iterative Update of Beliefs

A key problem with state space models is that they are subject to combinatorial explosion. To contain this explosion, several methods can be applied including: Pruning (Bellman 1957); branch and bound (Morrison, et. al., 2016), heuristic search (Szer 2012), Monte Carlo search (Browne 2012), and policy tree (Golovin 2010). Additionally, we relax the strict Markov assumptions by including heuristic analyses that use state trajectories when necessary for reducing ambiguities that increase the cost of computing the most likely belief state.

REAL-TIME UAV PLANNING AND DECISION-MAKING USING POMDP

The use of Markov Decision Process (MDP) and reinforcement learning have proven to be successful in environments in which an agent (e.g., UAV) has access to reliable state information (Spaan, 2012). With MDP models that employ reinforcement learning, the agent is assumed to receive perfect and complete information about its environment through both onboard and remote sensors. However, assuming perfect state information (full observability) is a strong assumption which is seldom true in the real-world. This is because sensors tend to be error-prone with limited capabilities for monitoring the environment. This limitation can result in ambiguity and uncertainty in determining system and environmental states.

Partially Observable Markov Decision Process (POMDP), an extension of MDP, addresses uncertain and incomplete observations (resulting from the agent's imperfect sensors) and allows for effective decision making in partially observable environments (Spaan, 2012). POMDPs have been used in various types of applications and problems, for which the classic examples are machine maintenance (Smallwood and Sondik, 1973), structural inspection (Tao et al., 1995), robot navigation (Foka et al., 2007), and human-robot interaction (Doshi et al., 2008). A POMDP accounts for imperfect knowledge of system state through the incorporation of hidden states (which are initially limited to observable system states.) The hidden states are intended to explain observations that cannot be explained using the observable states.

A partially observable Markov decision process is a tuple $\langle S, A, \Omega, T, O, R \rangle$ in which S is a finite set of states, A is a finite set of actions, Ω is a finite set of observations, T is a transition function defined as $T: S \times A \times S \rightarrow [0, 1]$, O is an observation function defined as $O: S \times A \times \Omega \rightarrow [0, 1]$, and R is a reward function defined as $R: S \times A \times S \rightarrow \mathbb{R}$ (Spaan, 2012). POMDPs share some features with the MDPs, such as the state, action, and reward definitions. However, POMDPs use uncertain and partial observations as indication of the states rather than observing the state directly. POMDPs are memory-less models that use a probability distribution, the so-called belief vector, to summarize their past information and to interpret their current knowledge of their environment. Each POMDP problem initializes an initial belief b^0, such as a uniform distribution over all states. This belief vector gets updated based on Bayes' rule each time the agent performs an action and makes an observation:

$$b'(S') = p(S'|a, o, b) = \frac{p(o|S', a) \sum_{s \in S} p(S'|s, a) b(s)}{\sum_{s' \in S} p(o|s', a) \sum_{s \in S} p(s'|s, a) b(s)}$$

In this formula, $p(o|S', a)$ is the probability of observing o when you do action a in state S' (i.e. emission probability) and $p(S'|s, a)$ is the probability of doing action a at state s and transitioning to state S' (i.e., transition probability). Finally, $b(s)$ is the current belief vector that shows the probability distribution over the states.

As with MDPs, POMDPs can evaluate a sequence of actions (policies) and find the optimal policy using the Bellman's equation. However, since the states cannot be exactly determined in a POMDP model, continuous probability distributions (belief vectors) are used to represent the best proxies for states in this model. Attempting to solve the Bellman's equation, either for policy evaluation or optimal policy search, is a computationally expensive proposition because there are uncountably many belief states, even for simple problems with small state and action spaces. Bellman's value iteration equation for a belief vector in a POMDP model is presented by the formula:

$$V^*(b) = \max_{a \in A} [\sum_{s \in S} b(S) R(S, a) + \gamma \sum_{o \in O} p(o|b, a) V^*(b_o^a)]$$

In this formula, $V^*(b)$ represents the optimal value for a belief vector b, $b(S)$ is the current belief vector, $R(S, a)$ is the value or reward of doing action a in state S, $0 < \gamma < 1$ is a discounting

factor, $p(o|b,a)$ is the probability of observing o, when you do action a on belief vector b, and $V^*(b_o^a)$ is the value of resulted belief vector after doing action a and observing o.
Various approaches have been introduced in the POMDP literature for reducing the complexity of value iteration in POMDP models. These include enumeration algorithm (Monahan, 1982); one-pass algorithm (Sondik, 1971); linear support algorithm (Cheng, 1988); witness algorithm (Littman et al., 1994); and incremental pruning (Cassandra et al., 1997). Even so, solving the POMDP problem continues to be a complex, computationally-intensive problem.

We employ for probabilistic planning and decision making in scenarios involving partial observability and disruptions. In other words, in this example we are not attempting to address the complexity of value iteration in POMDPs. Rather, our goal is to simplify POMDP models by relaxing specific constraints that do not detract from our objectives and incorporating heuristics associated with reasonable assumptions.

In this example, we execute POMDP models in operational scenario simulations. In our approach, based on the interaction between the agent and the environment, the agent considers only belief points, that result from previous actions and environmental observations, instead of considering every single belief point that could result from all combinations of observations-actions sequences. In other words, automatic pruning of the belief tree in the POMDP model is performed by limiting the constructed belief space to only those belief points that result from performed actions and observations (i.e. feedback) from the environment. This example divides the agent-environment's actions-reactions into episodes (a series of finite number of time-steps), and the value function in the POMDP model only evaluates the best policy for an episode, rather than evaluating the overall best policy for the entire scenario. In this example, each time-step is defined as the time that the POMDP model is being invoked for planning and decision making. Assuming that the scenario can be divided into discrete, time-steps that cumulative are equal to the required number of actions that the agent takes within a scenario, each episode will contain a specific number of these time-steps (e.g. 2 time-steps). In our POMDP model, the episodes overlap. This means that the tail of the previous episode becomes the initial point for the next episode. This strategy helps in refining and updating predicted actions and policies from previous episodes based on new observations associated with new episodes.

Partitioning the scenario into overlapping episodes helps in determining the optimal belief tree (sequences of beliefs in a scenario) for the whole scenario. The optimal belief tree is achieved by attaching the series of constructed belief trees from episodes to each other.

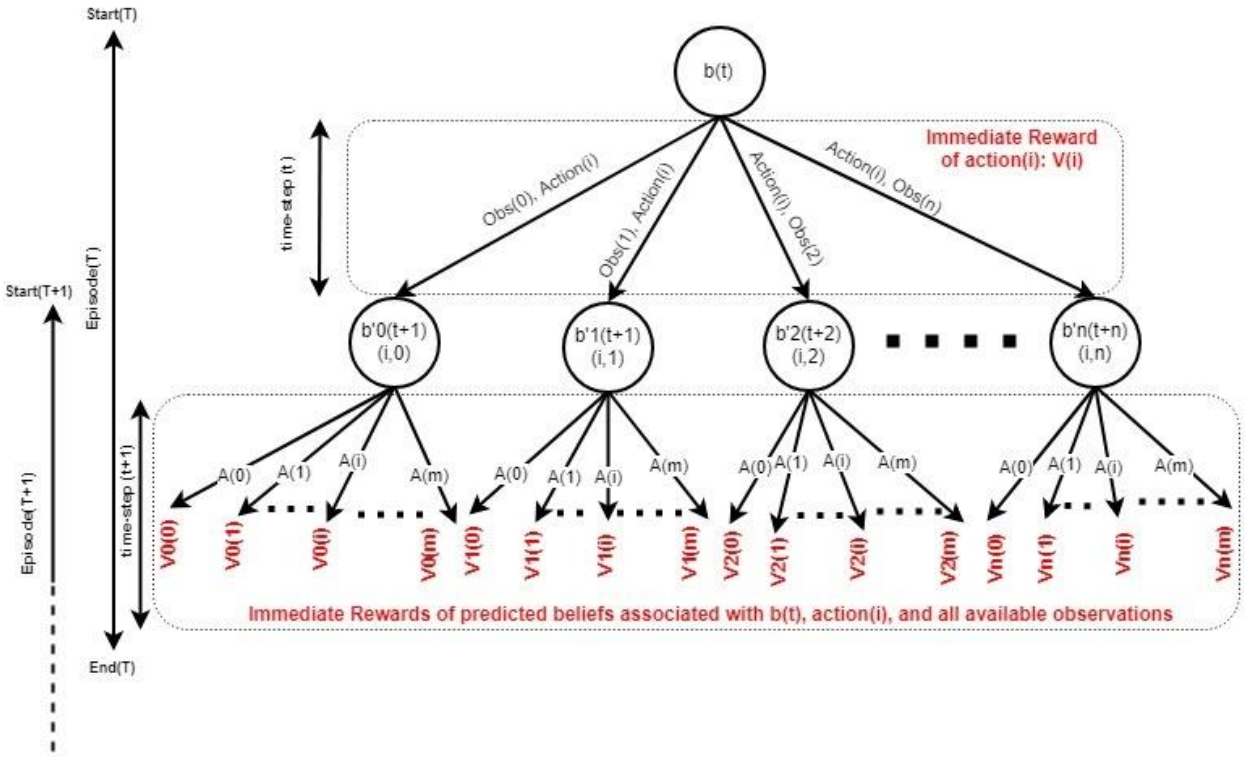


Figure 7. Example of Value Function in a Sequence of Episodes with Predicted Beliefs, $b'(t+1)$, (in one episode based on the current belief b(t), one action that is action (i), and all available observations)

In our POMDP model (Figure 7), we evaluate the mapping between beliefs and actions. These are partial policies in each episode based on the current belief vector, available actions, observation made, and resulting beliefs associated with that episode. We do these instead of evaluating the values of every belief vector using the Bellman's value iteration equation. In other words, the value function in our POMDP model is designed to evaluate the available policies (based on available actions, observation made, and updated belief vectors), and provide a series of actions that can result in the maximum value for that episode. The series of actions resulting from each episode can be refined and completed by actions associated with the next episode that overlaps with the current episode (based on observations made and agent's actions). The value function is discussed in the next section.

CONSTRUCTION OF THE UAV POMDP MODEL

The UAV POMDP model is intended to enable probabilistic decision making and planning for an autonomous UAV in a representable UAV scenario mission that requires the UAV to accomplish its mission safely. In our model, the POMDP is used for higher level planning and decision making based on the UAV's behavior, conditions, and defined mission(s).

Defining the actions, states, and goal: A key challenge with reinforcement learning (RL) is designing an appropriate system model with just enough number of states and actions to avoid state space explosion. State space explosion typically occurs when a model has a large number

of states which make its solution computationally impractical and expensive. To avoid running into these problems in our example, we defined the POMDP model for high-level decisions by using 5 (high-level) states and 5 general actions. This means that each state is a representation of a class of observations rather than only a single type of observation, and the actions are high-level decisions. In other words, the POMDP model provides a high-level solution to the problem, in that, it is the core of the optimal solution that makes only high-level decisions based on evidence and belief. We use heuristics, rules, and helper functions outside the POMDP model to infer observations and translate high-level decisions into reasonable actions that the UAV's controller can understand. Making appropriate and warranted assumptions and using the right rules and heuristics helps with modeling the system with a reasonable number of states and actions.

In this example, an autonomous UAV is employed to maintain surveillance and monitoring in a pre-defined area, such as an airplane landing strip. This requires a UAV to make right decisions, such as move or hover anytime it's needed. As noted above, subroutines and helper functions are employed to translate a decision to an action by providing the requisite details. For instance, if the POMDP decides that the UAV should "move", then the helper function tells which direction it should move, or where it is required to be by informing it "move from point A to point B". In addition to mission-based states, the POMDP model needs to ensure that the UAV is physically capable of performing its mission. The latter implies that the UAV has adequate battery power and subsystems are in reasonably good health. Based on the foregoing assumptions, we have defined the following states, actions, and observations for the UAV POMDP model.

Table 2. Physical Readiness State Space of the UAV POMDP Model

State	Definition
S_0 **(Good to Go)**	This state represents the physical capability of the quadcopter in performing assigned missions.
S_1**(Battery Yellow)**	This state represents the situations when the quadcopter has battery power only for performing a few actions (e.g. landing).
S_2 **(Battery Red)**	Similar to the previous state, this state provides information about the quadcopter's battery status and represents the situations in which the UAV's battery is running out of power.
S_3 **(Health Issue)**	This state is defined to provide information about the overall health status of the quadcopter, which includes combinations of health-related information from different subsystems of the UAV. If there is any problem or issue with any of the sub-systems that are being monitored, this state will communicate that to our POMDP model.

Table 2 shows the state space of the POMDP model that is defined for monitoring the physical readiness of the UAV quadcopter. Each state is responsible for monitoring a class of information (observations) resulting from quadcopter subsystems. The initial state or "Good to Go", S_0, will have a higher probability if the quadcopter's safety is ensured. This implies that the combination of information reported from the different quadcopter subsystems have met pre-defined criteria for the quadcopter, which shows physical ability of the quadcopter for performing actions. The

second and third states, which are "Battery Yellow" and "Battery Red", S_1 and S_2, respectively, represent a class of observations made from the UAV's battery voltage. State 1 will have a higher probability when an observation shows that the battery voltage is in a pre-defined "Yellow" range. This color code implies that the UAV's battery will allow for only a limited number of actions, such as moving the quadcopter to the closest charging station. On the other hand, state 2 has a higher probability if an observation shows that the battery voltage is below a pre-defined safe threshold. This condition means that the quadcopter should abort its mission and land immediately.

The last state or "Health Issue", S_3, represents classes of information (observation(s)) that show the quadcopter is not reliable anymore because of abnormal behavior of one or more subsystems. In other words, the probability of transitioning to S_3 is high if the quadcopter makes an observation that does not meet a set of pre-defined health related requirements and thresholds.

Table 3. Mission State Space of UAV POMDP Model

State	Definition
S_0 **(On Spot)**	This state is defined based on the quadcopter's location compared to where it has to be. This state enables the POMDP model to control the UAVs location by ensuring that the UAV is located on a desired and pre-defined location.
S_1 **(Off Spot)**	Similar to the "On Spot" state, this state is defined with respect to the quadcopter's location. However, it represents classes of observations that show the UAV is off a pre-planned location.

Table 3 represents the mission related state space of the UAV's POMDP model. These states are defined to keep track of the quadcopter's locations in the area. S_0 or "On Spot" state is a representation of location related observations that show approximately zero distance between the quadcopter's current location and its pre-planned location. On the other hand, S_1 or "Off spot" state describes those location related observations that the distance between the quadcopter's current location and its pre-planned location is not negligible. This can happen when assigning a new mission(s) or under off-nominal environmental factors, such as wind gust. The UAV's mission related POMDP model (i.e. states and actions) is a separate model, distinct from the UAV's physical readiness POMDP model. This implies that there is no action that can result in direct transition from mission-based states to physical readiness related states. However, the former POMDP model is enabled by the latter. In other words, the mission related state space is nested inside the physical condition state space and is only enabled, if needed. As noted earlier, we want to ensure physical readiness before undertaking a mission. Thus physical readiness-related decision making and planning needs to occur prior to mission related planning and decision making. For instance, if the quadcopter's POMDP model believes that the quadcopter is in the "Battery Red" state, then planning for any mission(s) is irrelevant, because the quadcopter is unable to perform the mission. The mission related POMDP is enabled only when the quadcopter's physical readiness POMDP model believes that the quadcopter is

physically capable of performing the mission or the quadcopter can take a series of actions to stay safe, such as moving to the closest safe state (e.g. charging station).

Tables 4 and 5 show all high-level actions associated with the quadcopter's physical readiness and mission based POMDP models, respectively. A_0 or "None/Wait" action is a joint action in the two quadcopter POMDP models and it implies that the quadcopter's POMDP model is not going to provide a new plan or decision until a new observation is made. The second action of the quadcopter's physical condition POMDP model is enabled when the POMDP believes that the system's "Good to Go" state has a high probability. This means that the quadcopter is physically able to perform the assigned mission(s). Thus, the lower level or nested POMDP is enabled for mission related decision making and planning. Besides the A_0 action, this POMDP model can inform the quadcopter to hover or move. The former is performed when the associated POMDP model believes that the quadcopter's "On-Spot" state has a higher probability and the quadcopter is on a pre-planned location. On the other hand, when the lower level POMDP believes that the quadcopter's "Off Spot" state has a higher probability, it decides to move the UAV towards a pre-planned location. A_2 or "Go to Safe State" action from the higher level POMDP, is associated with the "Battery Yellow" state from this POMDP. A "safe state" is a set of pre-defined charging stations located on the area or ground itself. When the POMDP informs the quadcopter to perform this action, a sub-function (i.e. helper function) finds the closest safe state (either a charging station or ground) and the quadcopter moves to that safe state. A_3 or "Land" action is the final action from the higher level POMDP. This action is associated with the states to which the POMDP transition when there is a health-related concern, or the battery is out of power.

Table 4. Physical Readiness Action Space of UAV POMDP Model

Action	Definition
A_0 **(None/Wait)**	This action means that the POMDP needs more evidence from the environment to make a decision, so it will maintain status quo until a new observation is received.
A_1**(Enable the Lower Level POMDP)**	This action means that the lower level POMDP (mission related POMDP) should be enabled and the overall action has to be decided by the lower level POMDP.
A_2 **(Go to Safe State)**	This action means that the quadcopter should find the closest "safe" state and move to there.
A_3 **(Land)**	This action requires the quadcopter to abort its mission and land on the ground immediately.

Table 5. Mission Action Space of UAV POMDP Model

Action	Definition
A_0 **(None/Wait)**	This action means that the POMDP needs more evidence from the environment to make a decision, so it will maintain status quo until a new observation is received.
A_1 **(Move)**	This action means that the quadcopter's destination location is different from its current location and the quadcopter should move from the current location to the destination location.

A_2 **(Hover)**	This action means that the quadcopter should hover at its current location.

Table 6 presents the observations of the POMDP model. At each time step, the quadcopter's POMDP model makes a decision and receives feedback (observation) from its environment based on its decision and will update its belief state based on the action and feedback. Since, the POMDP is designed to keep track of several types of information using its states, an observation needs to be a combination of all the features identified in the table above. In other words, the POMDP will receive a tuple *⟨Loc0, Loc1, Voltage, Vibration Rate, Error Rate of Quadcopter Performance, Battery Discharge Rate, Engines Temperature⟩* in which $Loc0$ and $Loc1$ are 2 individual vectors that provide information about the current location and desired location of the quadcopter, respectively (i.e. local GPS information). $Voltage$ is the current battery voltage and provides the required information for POMDP's "Battery Yellow" and "Battery Red" states of the system's battery. $Vibration\ Rate$ is in the form of time series data. It provides information on vibrations in a quadcopter. The $Error\ Rate\ of\ quadcopter's\ Performance$ observation from the tuple is an observation that provides information about the accuracy of the quadcopter in performing assigned actions. For instance, this observation can be GPS error if the action is move from point A to point B to show how accurate the UAV can be in its maneuvers. $Battery\ Discharge\ Rate$ is an observation that provides information about the quadcopter's battery discharge rate, so it can be compared to an average discharge rate of a healthy battery. $Engines\ Temperature$ is the final entry in the observation-tuple that provides the quadcopter's engine temperature (Celsius or Fahrenheit).

Table 6. Observations for Quadcopter POMDP Model

Feature	Definition
Current Location	This is a vector (x, y, z) that shows the current location of the quadcopter.
Desired Location	This is a vector (x', y', z') that shows the location where the quadcopter should be.
Battery Voltage	This observation provides information about the status of the battery.
Vibration Data	This observation provides information about the vibration rate of the quadcopter.
Performance Accuracy	The Performance Accuracy observation is associated with the ability of the quadcopter in performing assigned actions.
Battery Health	This observation provides information about the discharge rate of the quadcopter's battery.
Engine Heat	This observation is associated with the measured heat from quadcopter's engines.

A specially designed helper function is used to analyze the information in the tuple based on a set of pre-defined rules and heuristics. This function determines the class of observations to which each observation belongs based on importance and priority of the features in the tuple. It is important to define appropriate heuristics and rules to prioritize the information in the tuple. For instance, one of the most important heuristics in the helper function is to analyze the quadcopter's overall health first, which gives higher priority to the observations that are related

to quadcopter's health, such as Vibration Data, Battery Health, and Engine Temperature. The reason for this check is that the quadcopter can perform its assigned missions if and only if its engines and batteries are in good health; if these conditions are not met, then the quadcopter's location or mission become irrelevant.

POMDP Model and Value Function (Table 7): Once the states, actions, and observations of the quadcopter's POMDP model are defined, we can initialize the POMDP model. This initialization comprises initializing the belief state, defining the transition and emission probabilities, and building the transition, T, emission, Z, and reward matrices, R. The initial belief states of both POMDP models associated with the quadcopter are uniform distributions over the respective state spaces of the POMDP models. This means that equal beliefs (probabilities) are assigned to each available state in quadcopter's POMDP models. Considering the number of states in the quadcopter's POMDP models (4 and 2 states in the physical condition and mission related state spaces, respectively), the initial belief states will be $b^0 = [0.25, 0.25, 0.25, 0.25]$and $b^0 = [0.5, 0.5]$ respectively. There are two reward matrices in this example. Each reward matrix is associated with a corresponding POMDP model. The reward matrices are initialized by assigning a reward (+10) for the right action at a specific state and penalties of (-20) to the wrong actions. Performing A_0 or none/wait will have a cost of -1 at each state.

Defining the transition and emission probabilities and building the matrices require initial guesses based on heuristics. In other words, Reasonable assumptions and heuristics are used to assign higher probabilities to common events, e.g. doing an action and transitioning from one state to another, while we assign lower probabilities to rare and uncommon events. For instance, an initial guess for the transition probability of doing "move" and transitioning from "Off Spot" state to "On Spot" state is very high. These probabilities are refined and updated using data collected from running quadcopter's POMDP models in operational scenario simulations.

As stated above, the value function in the quadcopter's POMDP model is designed to evaluate policies for a short episode in the scenario, rather than evaluating all available policies. At the beginning of each episode (time step = t), the value function receives the belief vector from the previous episode, and uses its current available information (i.e. the current belief vector, transition and emission, and reward matrices) to: 1) calculate the value of each policy resulting from the current belief vector and each action, and update the belief vector for that action considering all available observations (time step = t+1), and 2) calculate the values of resulting policies from the updated beliefs and all available actions (time step = t+2). Basically, the value function investigates all different beliefs, resulting from the current belief vector and each action, by looking ahead for one-time step and searches for the best series of actions that produces the maximum reward for that episode. For instance, if we assume that the current belief vector (from previous episode) is $b^t = [b_0, b_1, b_2, b_3]$ the value function will calculate the value, $V(a)$, of the policy that results from $b^t$and action a, from the action space and update the belief vector using b^t, a, for all available observations $O = [O_0, O_1, O_2, O_3]$. After updating the belief vectors, $b'0^{t+1}, b'1^{t+1}, b'2^{t+1}$, $b'3^{t+1}$, the value function calculates the values of policies $V0(0), \dots, V3(4)$ for each updated belief vector by considering the probability of transitioning to that belief vector, which is equal to the overall probability of receiving the specific observation

in b^t that resulted in that belief vector. Finally, the value function returns a sequence of actions that produces the highest policy value at that episode. Table 6 presents the pseudo-code for POMDP Value Function calculations.

Table 7: Pseudo-Code for the POMDP Value Function

$Function\ POMDP\_Value(b^t, Z, T, R)$:
$Initialize\ v\ = [\ ], index1 = [\ ]$
$For\ action\ =\ 0\ to\ \#actions$:
    $Initialize\ gamma\ =\ 0.9;$
    $Initialize\ prob\_temp = [\ ]$
    $Initialize\ v\_temp = []$
    $For\ o = 1\ to\ \#observations$:
        $b_{temp} =\ update\ belief\ stateas\ (b^t, Z, T, action, o);$
        $For\ a = 0\ to\ \#actions$:
            $V\_temp[o, a]\ = (sum(Z[:, action, o] * b^t)) * (\ R[:, a] * b_{temp})$
        $End\ for$
        $prob\_temp \leftarrow\ sum(Z[:, action, o] * b^t)\ /*$
$probabilities\ of\ observations\ in\ b^t\ */$
    $End\ for$
    $ind\_\max \leftarrow argmax(prob\_temp)\ /*\ most\ probable\ observation\ in\ b^t\ */$
    $V\ \leftarrow\ R[:, action] * b^t +\ gamma *\ max(v\_temp[ind\_max\ , :])$
    $index1 \leftarrow [action, ind\_max]$
$Return\ (index1[argmax(V)])$

Figure 8 represents an example of the quadcopter's physical readiness POMDP model in a simulated scenario. In this example, the UAV's physical readiness POMDP starts with equal beliefs over the states, because it has not received any observations at that moment. Thus, the value function decides to "wait" until it receives an observation. After performing the "None/Wait" action, the quadcopter receives an observation, which implies that the quadcopter is in a good health status and has enough battery, "Good to Go", that results in increasing the belief of S_0. The value function evaluates the updated belief and calculates the value of performing various actions using the rewards and penalties that are provided in the reward matrix, and the action "Asking the Lower Level POMDP" receives the highest value, 7.41, based on the updated belief. This action enables the lower level POMDP, so the quadcopter can perform the assigned mission. After performing the "Ask" action, the quadcopter's physical readiness POMDP receives another observation, "Good to Go" and updates its belief vector. At this point, S_0, receives a higher probability (0.95) compared to the previous belief vector, in which the probability of S_0 was 0.83. The value function evaluates the updated belief vector and the action "Ask the Lower Level POMDP" with value of 11.6 becomes the next decision. However, after performing the "Ask" action, the POMDP receives an observation that means the battery voltage is in a yellow range,

"Battery Yellow" and increases the probability of state "Battery Yellow" to 0.71. The value function evaluates the current belief and decides that the "Go Safe" action is preferred with a value of 7.39. The POMDP then receives another observation that shows there is an issue with the quadcopter that implies "Health Issue". As shown in the last bar chart in Figure 3, the probability of the "Health Issue" state is increased to 0.68 after updating the belief vector. Thus, the POMDP decides to immediately perform the "Land" action and abort the mission because of the health issue.

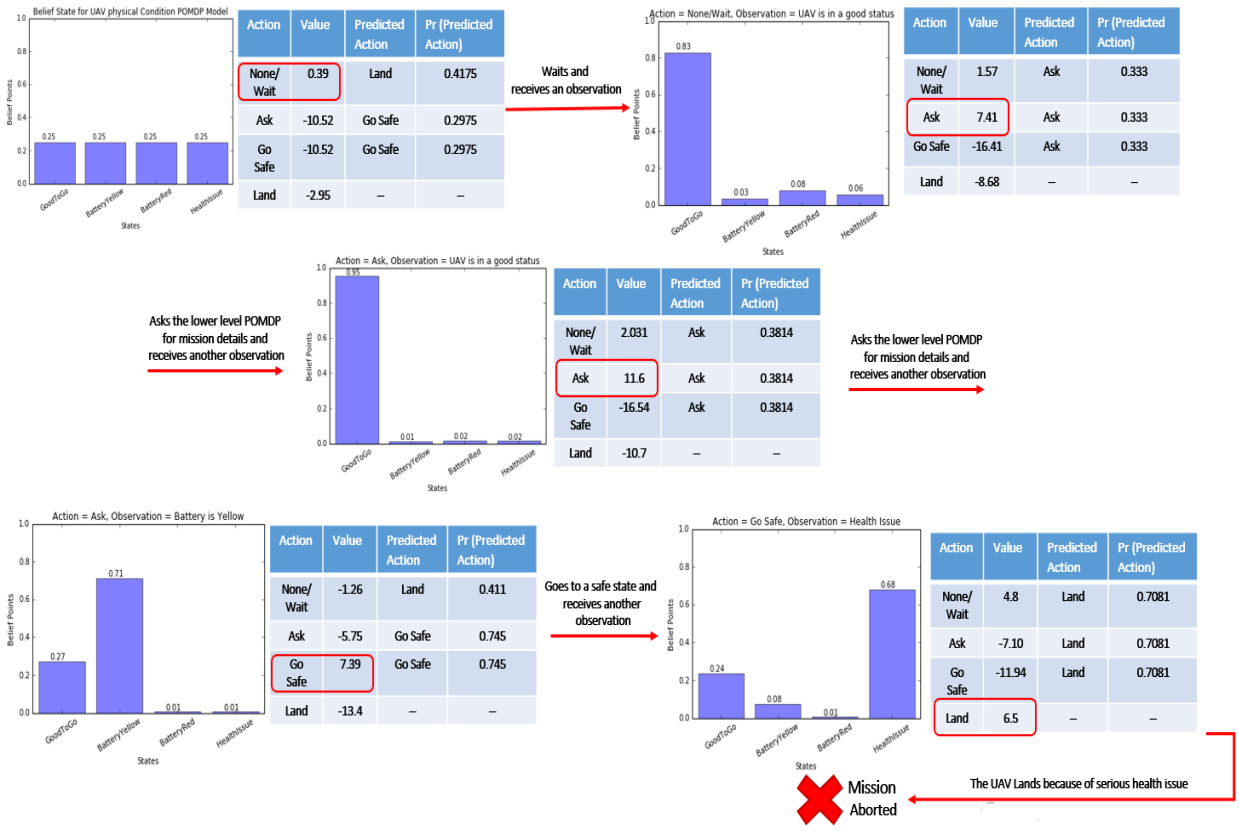


Figure 8: An example of UAV's Readiness POMDP Including Belief Vectors and Value Function Evaluations

HIDDEN STATES IN THE UAV'S POMDP MODEL

As noted before, a POMDP accounts for imperfect knowledge of both system and environmental states. This implies that the POMDP can plan and make decisions even if it receives an observation that it cannot interpret using its observable states. In other words, a POMDP can change non-Markovian state information into Markovian through incorporation of hidden states. The hidden states and their transition and emission probabilities are not known initially. This implies that there are no transition and emission probabilities associated with the hidden states in the pre-defined transition and emission matrices. If a hidden state needs to be introduced in a POMDP, the POMDP model should expand by initializing a new state in its belief vector, transition, emission, and reward matrices. In the quadcopter's POMDP model, a set of functions

are implemented to initialize hidden states with appropriate probabilities in the POMDP model. For instance, if the POMDP receives an observation that it cannot explain using its available states, the POMDP expanding functions add a new state into the POMDP matrices and belief vector and assign them small probabilities (e.g. 0.001). Later, these probabilities are refined and updated in the light of data collected from scenario simulations.

EXPERIMENTAL TESTBED DEVELOPMENT

In principle, the proposed R&D could be carried out using simulated UAVs. However, demonstration of the work using an experimental testbed is important, for several reasons:

- The real world always presents unanticipated difficulties which challenge an approach in significantly greater depth than do simulations;
- Actual devices (sensors and actuators) are noisy and require more careful analysis and control than do simulated devices; and
- Experimental demonstrations tend to be more convincing than simulations.

The goal of the experimental testbed development was to demonstrate a prototype UAV whose actions could be controlled by a decision-making algorithm such as POMDP. The conditions (requirements/constraints) included:

- Ability to fly in an indoor laboratory as well as outdoors
- Large enough to carry a powerful onboard computer with a full suite of sensors (camera, GPS, IMU) which can run autopilot software as well as POMDP
- Support for open source software

Flying indoors meant that airplanes were ruled out as well as gasoline-powered motors. Battery-powered quadcopters were thus the clear choice. We selected a class of quadcopters approximately 24-inches in diameter, with 1000 KV motors and 10-inch propellers, taking a LiPo battery with capacity of order 3000-5000 mAH. There is a wide variety of kits and parts for this class of vehicles. Furthermore, since these are widely used by hobbyists, they are generally quite inexpensive.

For the onboard computer, a combination of Raspberry Pi 3 single-board computer and Navio2 flight controller was selected. The Raspberry Pi is a little smaller than a deck of cards but is a quad-core 1-GHz 64-bit CPU with 1 GB RAM, costing about $35. It runs a flavor of Debian Linux and so supports essentially all open-source software.

The Navio2 is a flight controller board that connects to the CPU via the GPIO pins. It carries the GPS, IMU, and magnetometer, as well as the PWM controllers for the motors. It is the most expensive component of the entire quadcopter but is essential for autonomous flight.

For the autopilot, we selected Ardupilot, an open source program, because of its support for our hardware and because there is a wide variety of modes of operation as well as supporting modules. We particularly required *guided mode*, in which the UAV responds to external

commands such as moving to a specified position, setting a specified velocity vector, or holding at a specified location and attitude. (An external command is one which originates outside the autopilot program. It can come from a ground station computer over a wireless communications link or from a different program running on the UAV CPU. Thus, guided mode is useful for fully autonomous maneuvers as well as centrally controlled operation.)

Ardupilot supports both simulated and physical quadcopters. Our prototype demonstration employed this capability to control of 3 quadcopters, two simulated and one actual. At present we have two complete operational quadcopters including flight controllers.

Indoor flight in our laboratory presents a special challenge because autonomous flight requires a solid GPS lock in the unmodified Ardupilot software. However, the GPS satellite signals are too weak in our laboratory to achieve this lock. Accordingly, one of our current tasks is to modify Ardupilot so that we are able to use position and attitude information obtained from camera observations of multiple Aruco markers positioned on the walls and ceiling of our lab. We have experimentally demonstrated good accuracy with this technique but have not yet incorporated it into the flight software.

SUMMARY AND CONCLUSION

This research project has developed a model-based approach for the design of resilient systems. The approach combines formal modeling and probabilistic modeling to ensure requisite system verifiability and flexibility. The approach combines traditional contract from Contract-Based Design (CBD) with flexible assertions and Partially Observable Markov Decision Process (POMDP) to create a hybrid modeling construct called a Resilience Contract (RC). A RC is well-suited to modeling complex systems such that both system model verification and system flexibility can be simultaneously addressed. The approach enables both system and SoS model verification and offers the requisite flexibility to respond to disruptions. The approach is shown in the context of multi-UAV swarm control with several simplifications that do not limit the feasibility of the approach. For example, system state is a multi-faceted term that includes system's location, health, fuel status, sensor status, etc. For simplicity, we have used system location as system state in the figures presented. However, the approach is sufficiently general to be applied to a variety of SoS including autonomous vehicle networks and smart grids.

Future work will focus on solving the UAV navigation problem using POMDP. To this end, the first step will be to employ belief states along with sensed information from the environment to update the pre-defined reward and penalty values for the grid. In other words, the probabilities associated with the belief state (array) will be combined. Specifically, the pre-defined grid value and the new values will be used as new rewards/penalties to determine the best policy at that time. This will result in the UAV's action (movement) within the operational environment. The process of updating the belief state and grid values (rewards/penalties) will continue, until the UAV reaches its destination.

REFERENCES

Bahill, T. and Madni, A.M. *Trade-off Decisions in System Design,* Springer, Dec. 2016.

Bellman, R. A. "Markovian decision process," *Journal of Mathematics and Mechanics*, 1957, Jan 1:679-84.

Bellman, R. A. *Dynamic Programming*, Princeton University Press, ISBN- 0-486-42809-5, 1957.

Browne, C., Powley, E., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., and Colton, S., "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, 2012, 4(1):1-49

Carlson, J.M. and Doyle, J. "Highly optimized tolerance: Robustness and design in complex systems," *Physical Review Letters*; 2000, 84 (11):2529.

Cassandra, Anthony R., Leslie Pack Kaelbling, and Michael L. Littman. "Acting optimally in partially observable stochastic domains." *AAAI*. Vol. 94. 1994.

Cassandra, Anthony, Michael L. Littman, and Nevin L. Zhang. "Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes." *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1997.

Cheng, Hsien-Te. *Algorithms for partially observable Markov decision processes*. Diss. University of British Columbia, 1988

Csete, M.E. and Doyle, J.C. "Reverse engineering of biological complexity," *Science*, 2002, 295(5560):1664-9.

Doshi, Finale, Joelle Pineau, and Nicholas Roy. "Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs." *Proceedings of the 25th international conference on Machine learning*. ACM, 2008.

Foka, Amalia, and Panos Trahanias. "Real-time hierarchical POMDPs for autonomous robot navigation." *Robotics and Autonomous Systems* 55.7 (2007): 561-571.

Goerger, S.R., Madni, A.M. and Eslinger, O.J. "Engineered Resilient Systems: A DoD Perspective," *Procedia Computer Science*, 20 28:865-72.

Golovin, D., Krause, A. "Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization," *Journal of Artificial Intelligence Research*, 2011, 42:427-486

Madni, A.M. and Jackson, S. "Towards a Conceptual Framework for Resilience Engineering," *IEEE Systems Journal*, 2009, 3(2):181-91.

Madni, A.M. and Sievers, M. "A Flexible Contract-Based Design Framework for Evaluating System Resilience Approaches and Mechanisms," *IIE Annual Conference and Expo*, 2015, May 30 - June 2 Nashville, Tennessee.

Monahan, George E. "State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms." *Management Science* 28.1 (1982): 1-16

Morrison, D., Jacobson, S., Sauppe, J., Sewell, E. "Branch-and-bound algorithms: A Survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, 2016, (19):79-102

Neches, R. and Madni A.M. "Towards affordably adaptable and effective systems," *Systems Engineering*, 2012, 16(2):224-34.

Sievers, M. and Madni, A.M. “A flexible contracts approach to system resiliency,” *Systems, Man and Cybernetics* (SMC), 2014 IEEE International Conference, 2014, IEEE.

Sievers, M and Madni, A.M., “Contract-Based Byzantine Resilience in Spacecraft Swarms,” AIAA *SciTech*, 2017.

Smallwood, Richard D., and Edward J. Sondik. "The optimal control of partially observable Markov processes over a finite horizon." *Operations research* 21.5 (1973): 1071-1088.

Spaan, Matthijs TJ. "Partially observable Markov decision processes." *Reinforcement Learning*.

Springer, Berlin, H Sondik, Edward Jay. *The optimal control of partially observable Markov processes*. No. SU-SEL-71-017. STANFORD UNIV CALIF STANFORD ELECTRONICS LABS, 1971 eidelberg, 2012. 387-414.

Szer, D., Charpillet, F., Zilberstein, S. “*MAA*: A Heuristic Search Algorithm for Solving Decentralized POMDP*s,” Jul 2012, arXiv:1207.1359.

Tao, Zongwei, Ross B. Corotis, and J. Hugh Ellis. "Reliability-based structural design with Markov decision processes." *Journal of Structural Engineering* 121.6 (1995): 971-980.

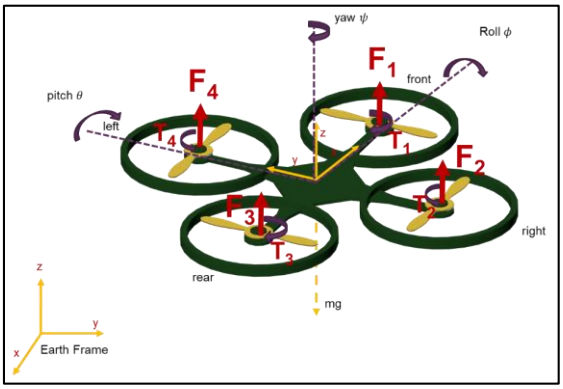
Williams, Jason D., and Steve Young. "Partially observable Markov decision processes for spoken dialog systems." *Computer Speech & Language* 21.2 (2007): 393-422.

Woods, D.D. “Essential characteristics of resilience,” *Resilience engineering: Concepts and precepts*, 2006, 21-34.

Wymore, A. W. *Model-based systems engineering*, CRC, Press, Boca Raton, FL, 1993.

APPENDIX A – VEHICLE PHYSICS MODELING

For vehicle physics models, we need "just enough fidelity" to accept action commands from probabilistic model and drive various visualization on the dashboard for situation awareness. Since the number of UAVs can grow, we need a sparse representation for vehicles. To this end, we chose quadcopters for our research because of their relative simplicity. Quadcopters in general are under-actuated systems in that 6 degrees of freedom (X, Y, Z, roll, pitch, and yaw) are controlled by only 4 rotors. These vehicles are nonlinear systems that require two non-linear controllers at the physics level, one for controlling attitude and the other for controlling position. Figure below shows a quadcopter for a particular orientation.



The physics model makes the following assumptions: a) the quadrotor is a rigid body with symmetric mass distribution; b) propellers are rigid; c) center of gravity and body fixed frame origin are co-located; d) Earth's gravitational field (g), quadrotor's mass (m) and body inertia matrix (J) are constants; e) thrust factor and torque factor of motors are constants; f) inertia of motors and rotors is negligible; g) aerodynamic drag force is proportional to translational velocity; and h) rotation of the Earth relative to distant stars is negligible. With these assumptions, the model becomes simpler, but requires further simplification to reduce computation.

Waypoint-waypoint path generation is done using analytical function called Wymore's Standard Scoring Function (Wymore, 1993). Since the vehicle in the illustrative example is in a near hover mode, and errors in X, Y, and Z directions are negligible, the vehicle's position can be predicted by calculating path coordinates without having to run a full dynamics model. Wymore's function used for path generation takes the following form. This function can be used to estimate vehicle location.

$$Position\_X = \frac{1}{1 + \left(\frac{\mathrm{B} - \mathrm{L}}{\mathrm{t} - \mathrm{L}}\right)^{2*\mathrm{S}*(\mathrm{B}+\mathrm{t}-2*\mathrm{L})}}$$

where t is simulation time; B is midpoint time between two waypoints; S is minimum speed (S=1/(B-L)); and L is required time from a waypoint to keep X, Y, Z bounded.

APPENDIX B – CSER 2019 PAPER

Madni, A; Sievers, M. "Combining Formal and Probabilistic Modeling in Resilient Systems Design" CSER 2019, Washington DC, April 3-4, 2019.