
# Formal Methods in Resilient Systems Design using a Flexible Contract Approach - Part 2

Technical Report SERC-2019-TR-015

September 27, 2019

**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**

**Shatad Purohit, 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

As systems and networks continue to grow in complexity and missions become increasingly more challenging, system safety and resilience have become key requirements. As a result, the ability to verify system model correctness and the ability for the system (model) to respond to disruptions are both needed to satisfy safety and resilience requirements. From a system modeling perspective, this means that the system model should be verifiable in terms of correctness*, flexible* enough to work with incomplete knowledge initially and fill in the knowledge gaps during system operation (learning), and *adaptable* to various types of disruptions. These are the requirements of resilient systems. In response to these requirements, we defined a modeling construct called a "resilience contract." A resilience contract (RC) balances verifiability and flexibility, the key prerequisites to safety and resilience. We demonstrated the use of the RC in an illustrative example of DoD relevance. In addition, we developed a rudimentary testbed to support our modeling, analysis and prototyping capabilities. This report summarizes our accomplishments on this project.

## INTRODUCTION

Resilience is a characteristic that 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 disruptions can take a variety of forms depending on the characteristic of the environment and available intelligence (Madni and Jackson 2009, Woods 2006). Resilience implies the ability to circumvent disruptions if they can be anticipated; withstand disruptions that fall within the designed performance envelope of the system; and recover rapidly from the negative effects of disruptions that fall 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 (Goerger et. al 2014, Madni 2018, Madni et. al 2018). 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.

Our research objective on this project was to develop a formal modeling approach for designing resilient systems. To this end, we developed a formal and probabilistic modeling construct called the Resilience Contract (RC). A RC relaxes the fixed assumptions of a traditional Contract and is represented by a Partially Observable Markov Decision Process (POMDP) with in-use learning (Sievers and Madni 2014). On this research effort, we applied RC to the problem of planning and decision making in multi-UAV swarms. This problem is of interest to both the DoD and the civilian sector. This research report presents our accomplishments in the development of a model-based approach based on POMDP to represent resilient systems and verify their designs.

## SYSTEM RESILIENCE

Engineered resilience can be a messy problem for the following reasons (Goerger 2014, Madni et. al 2018): Requirements can be imprecise (especially initially); actions can be unclear (especially initially); system states can be ambiguous (partial observability); incompatible with invariant methods. What is needed is a formal and flexible modeling methodology that provides value even with partial information, has ability to incrementally learn from new evidence, and facilitates a key tradeoff between flexibility (resilience) vs. formality (V&V).

There are many interpretations of system resilience (Woods 2006, Madni et. al 2019). Resilience can be interpreted as recoverability, which is system ability to rebound and return to equilibrium at original, slightly degraded, or better state (Madni et. al 2019). In some literature resilience is interpreted as robustness which is system ability to absorb a disturbance within design envelope without any structural change. If resilience is interpreted as extensibility, then resilience equates to system ability to extend gracefully (i.e., add capacity/resources) to fulfill demand increase. (Madni et. al 2019). System resilience can also mean adaptability, which is system ability to monitor and adjust continually through restructuring or reconfiguring to counter disruptions. (Madni et. al 2019).

Resilient DoD systems need to be able to operate safely in dynamic, uncertain environments, Tolerate / survive systemic faults and failures, accomplish goals (e.g., navigate safely) even with incomplete information (e.g., partial observability), adjust or adapt to environmental disruptions, protect against physical and cyber threats, and reconfigure / restructure to minimize impact of disruptions (e.g., security breaches, loss of sensing node or comm link) (Goerger et. al 2014, Madni et. al 2019, Madni 2018).

## TECHNICAL CHALLENGES

There are several technical challenges that have to be overcome in developing formal methods to engineer 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. (Madni and Sievers 2018). The modeling construct needs to support verification and testing of the system model and SoS network in uncertain environments prone to disruptions. It needs to accommodate changes in structure and behavior to cope with disruptions. Therefore, it is also required to support bi-directional reasoning (evidence, model). The modeling construct needs to provide value even with partial information (not "data hungry") and learn incrementally from new evidence (reinforcement learning).

**Experimentation Platform.** The experimentation platform for this effort needs to support SoS specification and visualization, deterministic and probabilistic modeling, and integration with analytics and reporting modules.

## INNOVATIVE MODEL-BASED APPROACH

Our innovative approach combines formal and probabilistic modeling with heuristics within a two-level architectural framework spanning planning and decision making (top level) and control (bottom level). This is a layered architecture comprising planning and decision-making layer and control layer. Decisions and information flow from planning and decision-making layer to control layer and execution constraints flow from control layer to planning and decision-making layer. In case of conflicts, satisfaction of global objectives has precedence over satisfaction of local system goals.

Formal modeling introduces rigor in system verification, testing, and reasoning (Madni 2019, Madni et. al 2019). 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 our approach and 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 (Monahan 1982, Williams and Young 2007). 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 1 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.

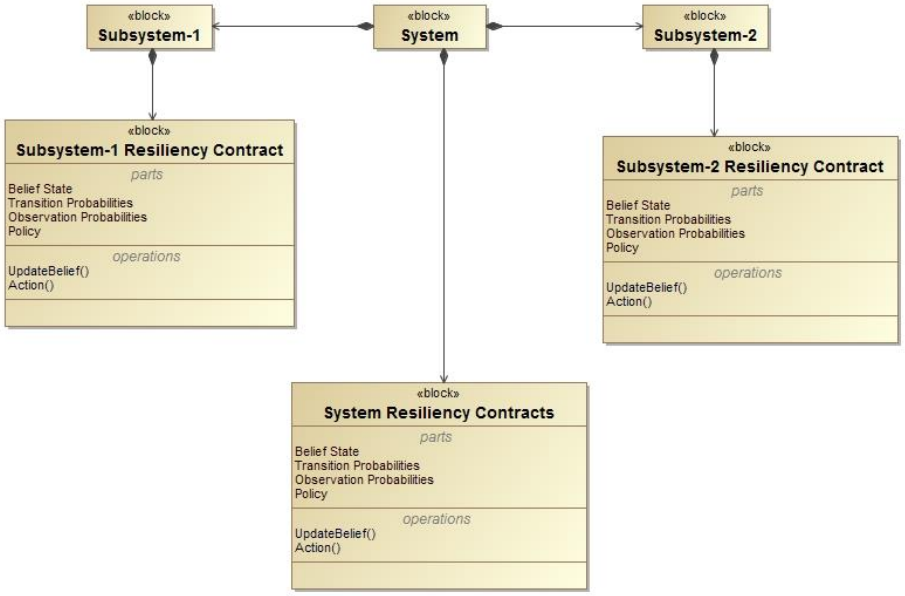


**Figure 1. Resiliency Model**

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. Key characteristics of resilience contract is summarized in Table 1.

**Table 1. Resilience Contract: Key Characteristics**

| |
|---|
| ■ relaxes assertions in traditional contract to "belief-reward" (flexibility)<br>■ Partially Observable Markov Decision Process (uncertainty handling)<br>■ in-use reinforcement learning (hidden states, transitions) |

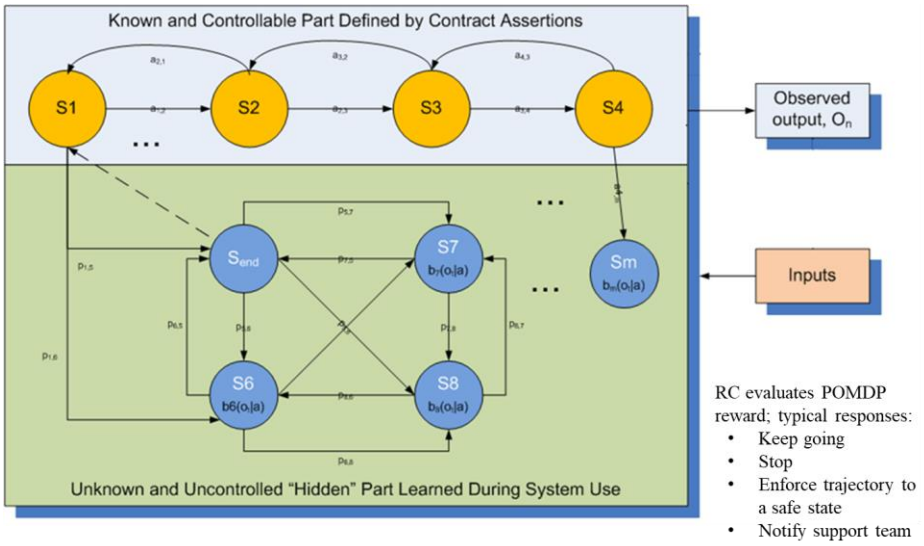


**Figure 2. Resilience Contract (RC)**

## ACCOMPLISHMENTS

### EXPERIMENTATION TESTBED

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 already incorporated it into the flight software.

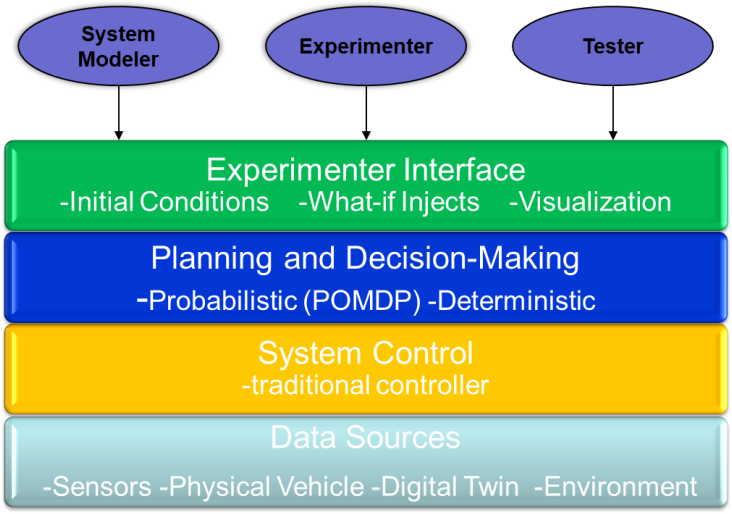


**Figure 3. Experimentation Testbed: Layered Architecture**

The testbed is critical for exploring modeling, verification, simulation, and analysis option in a controlled environment. Concept exploration and model verification are critical for exploring different CONOPS and verifying system models through static correctness analysis and simulation-based evaluation in a controlled experimentation environment. The modeling approach needs to support verification and testing of the AV system model and network in uncertain environments prone to disruptions. Additionally, it should accommodate changes in structure and behavior and support bi-directional reasoning. The modeling construct should be scalable and extensible and provide value even with partial information. As the system deals with uncertainty, the system model must learn incrementally from new evidence. The testbed is well-suited for incrementally developing and demonstrating system and SoS capabilities.

The testbed also offers an ideal platform for developing, maintaining, and applying Digital Twin technology throughout the system lifecycle. Digital twin technology produces significant time and cost savings. In the last few years, digital twin technology has emerged as a promising cost-effective, approach for replicating complex physical systems as virtual models for use in simulation as well as for system verification and testing. The basic idea of a digital twin is quite simple – create a specific virtual replica of a physical or human system to test-drive their behaviors in simulation to uncover and fix deficiencies. This idea can be exploited in the current hardware-software experimentation testbed as well. For example, the autonomous vehicle physical system can be represented within the simulation as a digital twin. As such, digital twin technology can become an integral part of a research testbed for developing and evaluating various models, CONOPS, and use cases. Lastly, the experimentation dashboard serves as a demonstration platform ideal for incrementally developing and demonstrating system and SoS capabilities. Figure 4 presents the key components of the experimentation testbed. These include quadcopters of two different sizes, a manual controller, and an experimentation dashboard.

**Figure 4. Key Components of Experimentation Testbed**

## POMDP SOLUTION ALGORITHM

Solving a POMDP model is a challenging problem and has been the focus of various research communities (e.g. computer science and robotics) for many years. Most of the available literature focus on solving the POMDPs offline, which means that they tend to find the optimum policy similar to solving MDPs. Various algorithms and techniques, such as dynamic programming, can be used for this purpose. Offline algorithms find the optimal policy for a POMP model prior to execution, which implies that the policy will not change if the model or problem objective is changed during the execution. Moreover, offline algorithms become non-trivial when applied to models with large state-space. Thus, because of the available limitations in the offline algorithms, we employ online algorithms that tend to find the optimal policy only for a belief state (current belief state) at a time step in a finite time-horizon. Basically, online algorithms, in contrast with offline algorithms, do not consider all possible belief states, but instead investigate only the belief states resulting from one specific belief state, which is the most recent belief updated during the execution.

The "N-Step Look-Ahead" algorithm, Table 2, is an online algorithm that finds the optimal policy for the current belief state. At every time step that the belief state is updated, the algorithm constructs a belief tree with depth equal to $N$ and estimate the long-term value for traversing different paths in that tree. Finally, the action that leads to the maximum long-term reward is considered the optimal policy for that belief state. It is important to note that since the number of the belief nodes increases with $N$, so the execution time of the algorithm, we prune the paths or plans with very low probabilities or very small expected long-term values.

**Table 2. N-Step Look-Ahead Pseudo Code**

TABLE I
PSEUDO CODE FOR "N-STEP LOOK-AHEAD" ALGORITHM

```
0  Function NstepLookAhead (b_t, N, γ):
1    X: Number of reachable observations
2    Z: Emission matrix
3    R: Reward matrix
4    value ← 0
5    If N is equal to 1:
6      For all a in A:
7        O' ← {X (ObsWithHighProbability) | b_t , a}
8        For all i in O':
9          b_{t+1} ← BeliefUpdate(b_t, a, i)
10         value ← value + Reward(b_{t+1}, R)
                   * Σ_{s∈S} b_t(s) * Z(s, a, i)
11       EndFor
12     EndFor
13   If N > 1:
14     For all a in A:
15       O' ← {X (ObsWithHighProbability) | b_t , a}
16       For all i in O':
17         b_{t+1} ← BeliefUpdate(b_t, a, i)
18         If b_{t+1} not a TerminalNode:
19             value ← value + γ
                 * NstepLookAhead( b_{t+1}, N − 1, γ)
                     + Reward( b_{t+1}, R)
                     * Σ_{s∈S} b_t(s) * Z(s, a, i)
20         Else:
21           value ← value + Reward(b_{t+1}, R)
                   * Σ_{s∈S} b_t(s) * Z(s, a, i)
22       EndIf
23       EndFor
24     EndFor
25   EndIf
26   Return value
```

## ILLUSTRATIVE EXAMPLE: NAVIGATING THROUGH HOSTILE ENVIRONMENT

As part of a larger mission, a quadcopter is given a specific destination to travel to. During its flight the quadcopter is going to travel through a hostile environment. The environment includes static obstacles and dynamic actors. These actors can be friends (e.g., other members of SoS) or enemy quadcopters that can interfere with the mission. Figure 5 presents an illustrative example of navigation through a hostile environment.

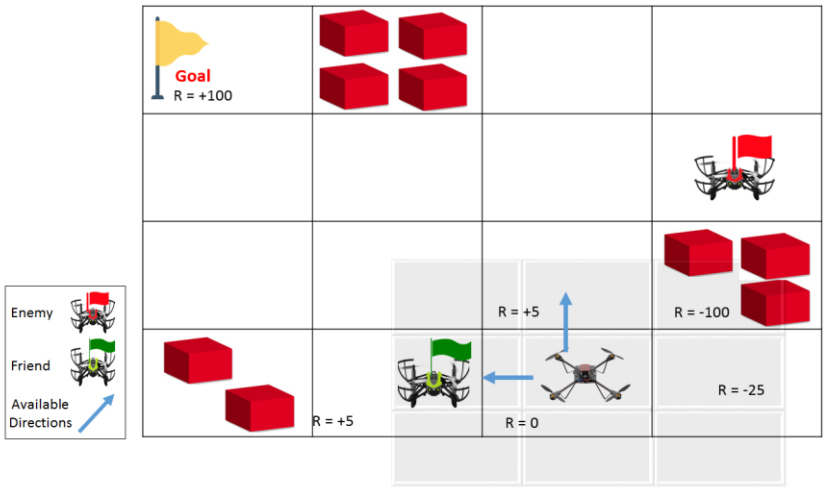


**Figure 5. Illustrative Example: Navigation Through Hostile Environment**

The problem of moving towards a new destination or navigation can be as simple as generating low level control actions, such as control commands without no object or obstacle detection for the quadcopter until it arrives at the destination. However, the quadcopter can only use the low level control commands to fly safely towards the destination if there are no fixed (e.g. trees or buildings) or dynamic obstacles (e.g. other quadcopters) on the way. In our scenario, we classify the problem of navigation with obstacles to the following categories: 1) Navigation in presence of fixed obstacles, and 2) Navigation in presence of dynamic obstacles.

### NAVIGATION IN PRESENCE OF FIXED OBSTACLES

Similar to other navigation problems, the overall goal of navigation in our scenario is to find a safe and considerably short path and travel towards a pre-identified destination. In our scenario, we assume that the quadcopter is only capable of observing a limited area, which only allows the quadcopter to recognize the objects inside that area. Thus, the quadcopter cannot plan its path all the way to its destination.

To navigate towards the destination with only limited knowledge about the environment, we turn the quadcopter's field of view into a grid. The grid is constructed in a way that the quadcopter is always located in the center of the grid and its adjacent cells are: North, North-East, East, South-East, South, South-West, West, and North-West (Figure 6).

| | | |
|---|---|---|
| North-West | North | North-East |
| West |  | East |
| South-West | South | South-East |

**Figure 6. Local Navigation Grid for a Quadcopter**

To make decisions and plan a path towards the destination while avoiding the static obstacles, a POMDP model is designed that the cells of the grid are considered as the states ($s_0$: $Center, s_1$: $North, \ldots, and\ s_8$: $North - West$ ) assuming that the quadcopter does not move diagonally and the actions are associa—ted with moving the quadcopter in one of these directions. The observations for this POMDP model are associated with the location of the quadcopter and its surrounding grid after taking an action. The most important factor in designing this POMDP model is the reward matrix. The rewards and penalties associated with the decisions in the POMDP model are dynamically assigned based on the following:

- The rewards matrix is initialized with 0s for all of the possible states.
- Each adjacent cell in the grid of the quadcopter is assigned with $r \in \{-5, 0, +5\}$ based on the Manhattan distance measured for the quadcopter's current cell to the destination compared to the adjacent cell's distance to the destination.
    - $R\{current\ cell\} = 0$
    - $if\ D_M\{Current\ cell, Destination\} = D_M\{Adjacent\ cell, Destination\}\ \ svx$
      $$R\{Adjacent\ cell\} = 0$$
    - $if\ D_M\{Current\ cell, Destination\} > D_M\{Adjacent\ cell, Destination\}$
      $$R\{Adjacent\ cell\} = +5$$
    - $if\ D_M\{Current\ cell, Destination\} < D_M\{Adjacent\ cell, Destination\}$
      $$R\{Adjacent\ cell\} = -5$$
- The reward of the adjacent cell is added by -100 if any fixed obstacle is recognized at that location
- To avoid unnecessary loops and find a short path to the destination, the reward value of a cell is added by -20 if it has been visited before.

After constructing the reward matrix associated with a specific grid, the POMDP model initializes its belief as $b_0 = \{0.92, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01\}$. Since, the quadcopter is assumed to always be in the center of the grid ($s_0$), this state is initialized with a higher belief compared to the others. Finally, the POMDP model uses the reward matrix, and finds the action that changes the grid and quadcopter's location in a way that the state

with the highest reward has the highest belief after the associated action is taken. If there are more than 1 action that has a high value and the values are close to each other, a set of decisions (actions) are available for the quadcopter to randomly pick from to proceed.

---

**NAVIGATION IN PRESENCE OF DYNAMIC OBSTACLES**

Dynamic obstacles become highly important when the quadcopter is assigned with a mission and has to maneuver in the field and travel toward its destination. This scenario gets more complicated when there are enemy quadcopters in the field in addition to friend quadcopters. To successfully perform an assigned mission and navigate safely in the field, the quadcopter should consider other flying quadcopters in the field, especially if they are located in the direction that the quadcopter is moving. The problem of decision making based on detecting the flying objects (e.g. quadcopters) and distinguishing friends from enemies is a partially observable problem due to the following reasons:

- The received observations (communication signals, image data, or motion data) from the other quadcopter(s) in the field can be partially available, because of especial weather conditions and limited sensor (e.g. camera) capabilities.
- The observations can be modified, replicated, or hacked by enemy quadcopters. In other words, enemy quadcopters can appear as/pretend to be a friend quadcopter.

The existing limitations and issues within the problem, does not allow for full recognition of the current system/ environment state. In other words, because of any lost, modified, or miss-interpreted information in the received signals (observations) from the environment, the state of the quadcopter cannot be updated deterministically. This implies that the state update is probabilistic, which leads to a belief state rather than a state description. Thus, a POMDP model is required for decision making and planning.

For decision making and planning in presence of active enemy and friend quadcopters in the field, a POMDP model is designed to *evaluate the identified path provided for the navigation*.

This POMDP model is described below:

States:

- $s_0$: view direction is clear (R = +10)
- $s_1$: view direction is occupied by a friend quadcopter (R = -1)
- $s_2$: view direction is occupied by an enemy quadcopter (R = -1)
- $s_3$: mission failure/ crash (R = -20)

Actions:

- $a_0$: randomly pick a direction from the available optimum directions
- $a_1$: pick the direction that is not occupied
- $a_2$: move away/ move in the opposite direction

- $a_3$: maintain status quo

Observations:

- $o_0$: nothing is observed on the view direction
- $o_1$: captured ArUco marker pattern identifies a friend on the view direction
- $o_2$: captured ArUco marker pattern does not match the pre-identified friend patterns
- $o_3$: failure data

In this POMDP model, the friend and enemy quadcopters are assigned with specific ArUco markers for identification purposes. It is possible to identify an enemy quadcopter as a friend based on the initially received observations from that quadcopter (if it is far away or the captured image of the ArUco marker is unclear) and the environment. However, as the quadcopter collects more evidence about the enemy quadcopter and updates its belief, the belief of seeing a friend will gradually reduce while the belief for observing an enemy will increase, so the decision of the quadcopter will change with respect to the changes in the belief.

The plot in Figure 7 shows the changes in the belief vector of a quadcopter as it makes decisions and receives feedback for its actions.

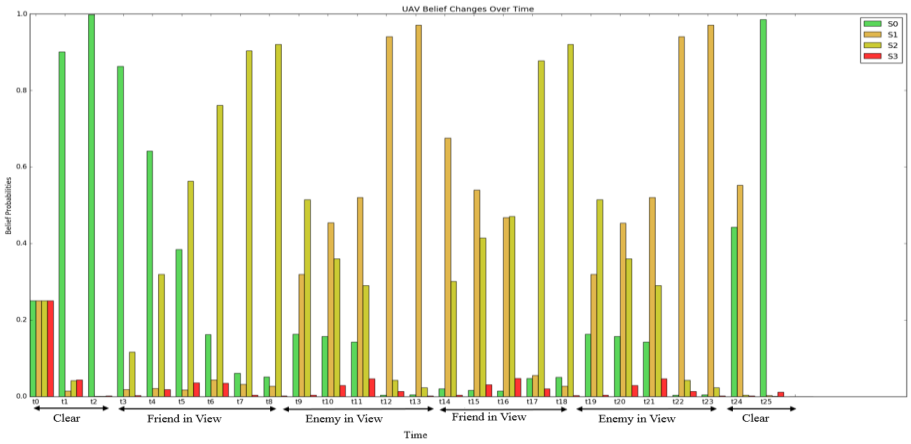


**Figure 7. Example Changes in Quadcopter Belief Vector**

## EXPERIMENTATION: INTEGRATING HARDWARE AND SOFTWARE

To show the applicability of our approach in the real world, we integrated our POMDP model discussed in the previous section with our hardware. We then experimented in our lab. This section summarizes our experimentation results.

The first experimental task was to test the performance of the POMDP obstacle avoidance algorithm on the experimental hardware (the Raspberry Pi 3 quadcopter flight computer). The code was separated from the dashboard (the graphical interface) so that it could run as a

standalone process with no user input. It was configured to communicate with the quadcopter autopilot using socket-based communication. (In these tests, the quadcopter was flying in simulated mode, i.e., the autopilot was running using simulated dynamics and sensor inputs.) The result was that the POMDP ran on the quadcopter with no loss in performance, even with the autopilot software also running. This demonstrates that the POMDP guidance algorithm is efficient enough to be practical for real-time use on autonomous vehicles.

The next task was to demonstrate the quadcopter actually flying and avoiding obstacles under POMDP control. The primary challenge here was to achieve indoor autopiloting with good location accuracy in a lab without GPS. We succeeded in developing and integrating a custom GPS driver into the Ardupilot software. This driver determines the quadcopter's location in the lab using the video stream from an upward-looking camera mounted on top of the quadcopter avionics module and analyzing the pose relative to a set of ceiling-mounted ArUco markers. The resulting location is converted into a standard NMEA GPS data record which is read by the autopilot.

Using this configuration, we were able to fly the quadcopter indoors in the autopilot mode. However, excessive motor vibration prevented stable autonomous operation for long enough to run the obstacle avoidance algorithm. This is purely a vehicle issue, unrelated to the POMDP, and is expected to be fixed shortly.

## FINDINGS AND LESSONS LEARNED

During the conduct of this research we accumulated several valuable lessons. First, when implementing a hybrid model, it is important to separate planning and decision-making from vehicle control by assigning them to two different layers. This separation allows experimentation with different models in the planning and decision-making layer without disrupting the integrity of the control layer. Second, constraint violations and other mismatches are likely to arise between the two layers. These mismatches can be resolved by ensuring that the propagated commands from the planning and decision-making layer to the controller do not violate physics or regulatory constraints. In the same vein, it is essential that the control layer propagates execution constraints to the planning layer for consideration when the planning layer issues commands. In case of conflicts between these two layers, some heuristics can be incorporated.
Third, the POMDP (i.e., vehicle planning and decision-making model) and controller work on different time scales. The system's dynamic model and low-level control algorithms run at high frequencies (e.g. 100Hz). In sharp contrast the POMDP model runs at a lower frequency when making decisions and issuing high level commands. Fourth, the ideal sampling period for POMDP can be determined experimentally with the objective of minimizing overall response time to action (a priority). Translation of POMDP commands into executable actions are performed by the control layer. The POMDP model will continue to give the same command to the controller until it realizes that a transition in vehicle's actual state has occurred based on observations. Fifth, a POMDP model is equivalent to a rule-based system for simple scenarios in which there is complete observability (i.e., partial observability is not a concern. Sixth, POMDP model states should be defined and created based on various conditions that the system/SoS can potentially

experience when interacting with its environment. Seventh, the ability to acquire new knowledge incrementally through reinforcement learning and to expand the model as required, makes POMDP models especially attractive for complex scenarios with partial observability. Eighth, the POMDP value function and the time horizon required for estimating the online policy are important parameters that affect system and SoS network behaviors. There is a trade-off between the finite time-horizon in the online look-ahead search and accuracy of the estimated long-term reward for a plan. In other words, as the online policy estimator algorithm explores further down the possible estimated belief tree, the accuracy of the estimated policy is increased. However, additional time is required to perform this calculation. Ninth, the POMDP reward/value function should be designed in a manner that considers the physical aspects of the vehicle. This practice will remove potential mismatches between the high-level plans and low-level control commands. Tenth, the POMDP model(s) are designed to include both goal and failure states in their state-space. The probabilities of transitioning from one state to another are captured in a transition matrix. Based on the states, probabilities, and rewards/penalties associated with the states, the online policy estimator selects the action that increases the probability of achieving the goal while maintaining a safe state. Eleventh, the concurrent creation of the testbed with the system model, facilitates experimentation and data collection. We are currently able to switch between simulation model and physical system. In the future, we will be able to acquire and incorporate operational data and maintenance history from physical systems into virtual models, thereby transforming virtual models into Digital Twins. Finally, the monitoring, control and execution dashboard turned out to be a valuable aid for understanding and debugging vehicle behaviors.

## SUMMARY AND CONCLUSION

This research project has developed a model-based approach for the design of resilient systems. The approach combines formal and probabilistic modeling to balance system verifiability and flexibility. This hybrid modeling construct, called a Resilience Contract (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 planning and decision making. During the conduct of this research, we contributed to constructing and solving POMDP models, specified scalable and extensible architecture for hardware-software integration, and implemented a rudimentary testbed for experimentation. This testbed along with the testbed components created on RT-183 are being transitioned to The Aerospace Corporation in keeping with our technology transition goals. In transferring this technology, we acknowledge the Government's intellectual property rights; contract clauses DFARS 252.227-7013 and 252.227-7014.

## REFERENCES

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

Madni, A.M. and Sievers, M. “Model Based Systems Engineering: Motivation, Current Status, and Opportunities,” *Systems Engineering*, 20th Anniversary Special Issue, vol. 21, issue 3, pp. 172-190, 2018.

Madni, A.M. Formal Methods in Resilient Systems Design Using a Flexible Contract Approach, 21st Annual Systems Engineering Conference, Tampa, Florida, October 22-24, 2018.

Madni, A.M., Sievers, M., Ordoukhanian, E., and Pouya, P., and Madni, A. “Extending Formal Modeling for Resilient Systems,” 2018 INCOSE International Symposium, July 7-12, 2018.

Madni, A.M. “Formal Methods for Intelligent Systems Design and Control,” AIAA SciTech Forum, 2018 AIAA Information Systems, AIAA InfoTech@Aerospace, Kissimmee, Florida, Jan 8 12, 2018.

Madni AM, Sievers M. A Flexible Contract-Based Design Framework for Evaluating System Resilience Approaches and Mechanisms. InIIE Annual Conference. Proceedings 2015 (p. 2982). Institute of Industrial and Systems Engineers (IISE).

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

Madni, A.M., Erwin, D., Sievers, M., “ Architecting for Systems Resilience: Challenges, Concepts, Formal Methods, and Illustrative Examples”, unpublished manuscript, 2019

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.

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

Sievers M, Madni AM. A flexible contracts approach to system resiliency. In2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC) 2014 Oct 5 (pp. 1002-1007). IEEE.

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.