Risk Considerations for Autonomy Software

Leila Meshkat, Ph.D., Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

Key Words: Autonomy, Risk, Reliability

SUMMARY & CONCLUSIONS

The application of increasingly higher level of autonomy for safety critical systems is inevitable. Ensuring the reliability of such systems requires managing the uncertainties that occur due to the increased autonomy. Our research indicates the increased uncertainty in autonomous software is due to increased discontinuities in the state space and inadequate situational awareness.

This paper summarizes key findings related to methods for the risk considerations of autonomy software. Existing methods for Verification and Validation (V&V) of autonomy software are summarized and a method for the assurance of autonomy software is suggested and demonstrated on a command execution use case. Risk and reliability are defined in the context of autonomy and an approach for risk assessment of autonomy is presented using an example use case. Key insights regarding areas of uncertainty for autonomy are provided, along with a suggested architecture for the systematic consideration of reliability within the context of a given autonomous planner.

1. INTRODUCTION

The last decade has seen an extensive amount of research and development in the area of autonomous software and systems. The advances in this field have been driven by new technologies in sensors and actuators, computing, control theory, modeling and simulation and systems design and engineering. Due to the breadth of the related disciplines, a variety of definitions and classifications have been suggested. This section explains some of these and establishes the context for the paper.

1.1 Definition

Boulanin et. al. [1] define autonomy as the ability of a machine to execute a task, or tasks, without human input, using interaction of computer programming with the environment. They sort out the relative notion of autonomy across and within different disciplines into three categories:

a. The human-machine command and control relationship

b. The sophistication of the machine's decision-making process

c. The types of decisions or functions being made autonomously

Gat [2] defines autonomy software for spacecraft as any software that fits into a part of the spacecraft control process that would normally involve a human. Jonsson et. al. [3] define the necessary conditions for autonomy to be:

a. Autonomy describes a range of behaviors associated with agents, systems that can sense the world around them as well as their own state, can make decisions about what to do, and can carry out their decisions through their own action.
b. An autonomous system can be controlled by commanding it to achieve a set of goals; the system itself transforms the goals into sequences of actions that accomplish each goal.
c. An autonomous system flexibly responds to off-nominal situations by adjusting its activity sequence to attain the high-level goals and maintain system safety.

Pecheur [7] explains that in its simplest form, autonomy software consists in control sequences that allow the controlled system to achieve its particular goal while tolerating a certain amount of uncertainty in its environment.

For the purposes of this paper, we consider as baseline the definition provided by Fesq et. al [10].: Autonomy is defined, as "making decisions and taking actions, in the presence of uncertainty, to execute the mission and respond to internal and external changes without human intervention." This includes the three main steps of perceive, decide and act.

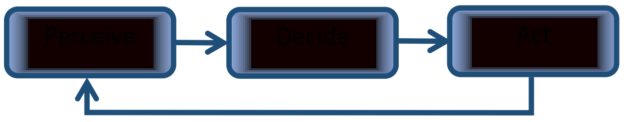


Figure 1: Autonomy Definition [10]

1.2 Classification & Levels of Autonomy

Classification schemes and levels have been suggested for autonomy software in cars [4], UAV's [5] and spacecraft [2]. We consider the three classes of fault protection, planning/scheduling and command execution for spacecraft autonomy, as suggested by Clark et al [12]. The levels across all these applications are dependent on the degree of control of the human operator versus the autonomy software with higher levels of autonomy indicating lower levels of human control. A Subject Matter Expert in Artificial Intelligence whom we interviewed, Dr. Russel Knight, suggested that the level of autonomy is proportionate to the number and complexity of decisions being made. This is shown in figure 2.

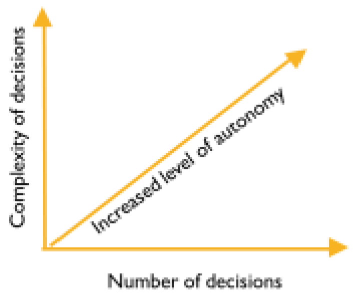


Figure 2: R. Knight Definition for Levels of Autonomy

1.3 Verification and Validation (V&V)

The two main methods for V&V of Autonomy software include testing and analysis. Pecheur [7] describes each of them as follow:

Scenario-Based Testing

"Software is embedded in test harness that connects to the inputs and outputs of that component, and drives it through a suite of test runs. Each test run is an alternated sequence of provided inputs and expected outputs, corresponding to one scenario of execution of the tested component. An error is signaled when the received output does not meet the expected one."

Analytical Verification

"Analytic verification is the branch of software engineering concerned with establishing, through some mathematically based analysis, that a computer program fulfills a formally expressed requirement. Two main approaches to analytic verification have been developed: Theorem provers build a computer-supported proof of the requirement by logical induction over the structure of the program; Model checkers

Perceive	**Uncertainties**	**Questions for Software Assurance**
System State	Level of accuracy	What is the underlying model of the system?
		How does the spacecraft maintain situational awareness?
		How does the spacecraft update it's state based on its observations?
		What are the possible error modes associated with this?
		How do we ensure that the system remains safe if the system state is not perceived accurately?
Environment	Level of accuracy	What is the underlying model of the environment used?
		How is the command execution dependent on the state of the environment?
		How does the system update its understanding of the state of the enironment?
		What are the underlying erroneous states the system can get into?
		How do we ensure that the system remains safe if the state of the environment is not perceived accurately?
Decide		
Execute sequence	Correct Execution	What are the impediments for the correct execution of sequences?
Change the order of commands in sequence	Correctness of new order	How do we determine if the order in which commands are executed are safe?
Change sequence completely	correctness of new sequence	What are the implications for various orders of commands?
		What is the implication of a wrong sequence?
		How do we ensure the correctenss of the sequence being executed?
Act		
Execute Decision	Ability to execute correctly	What are the impediments for executing each decision?
		Are there interacting sequences involved?
		What are the interections?
		How could we ensure they interact safely?

Table 1: Risk Identification for Command Execution

search all realizable executions of the program for a violation of the requirement. "

He concludes that because of the internal complexity of autonomous controllers, and the huge range of situations that they can potentially address, scenario-based testing provides a very limited coverage. Model checking can help to find the concurrency problems that would be overlooked in testing, and fix them earlier in the development and thus at a cheaper cost.

2. RISK CONSIDERATIONS

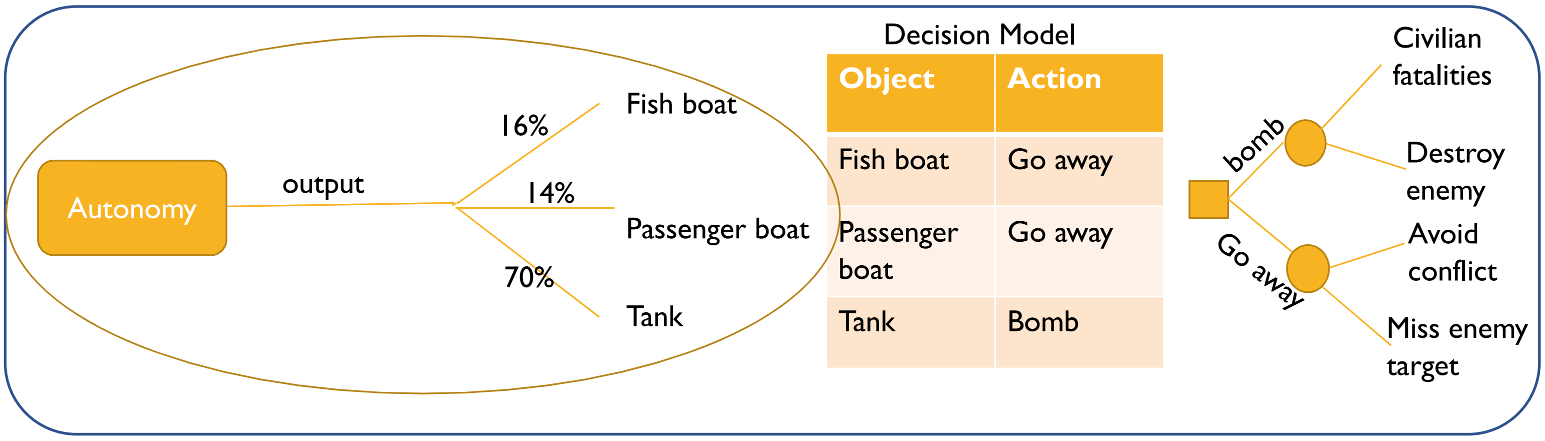
Risk is defined as a triplet <What can go wrong?, What is the likelihood?, What is the consequence?>[7]. By applying this definition to autonomy risk, we derive the triplet: <What can go wring due to Autonomy?, What is its likelihood?, What is its consequence?>. Similarly, system unreliability due to autonomy can be defined as the occurrence of an unreliable state for the system due to autonomy.

2.1 Risk Identification

Our suggested approach for risk identification for autonomous systems is to determine the key perceptions, decisions and actions involved, their corresponding uncertainties and questions that can help to identify the risks and unsafe states associated with those uncertainties. We demonstrate this approach using command execution as our use case for a class of autonomy.

2.2 Use Case: Command Execution

The first column in Table 1 includes a set of perceptions, decisions and actions that a given command execution software conducts. The second column high-lights some of the uncertainties involved in these decisions and based on these uncertainties the third column identifies questions that are addressed for software assurance purposes. In our example, the system perceives the state of the spacecraft as well as the state of the environment. It then decides to execute a given sequence, change the order of commands in the sequence or change the sequence completely and in each of these cases there's uncertainty about the correctness of the sequence and its corresponding order of commands. The action taken is the execution of the decision. The key uncertainty involved in the perception of the system state is the level of accuracy of this perception. To determine the hazards and possible failure modes that could lead to an unsafe system state as a result of this uncertainty, some of the questions that arise are about the underlying system model, the situational awareness of the spacecraft, the method it uses to update its state based on its observations, the possible error modes associated with it, and finally probing into the method that the system uses to stay safe if the system state is not perceived accurately. The expectation is for this method to be used to facilitate discussions between software developers and software assurance managers for the purposes of identifying and managing the risks associated with autonomous software.



Object	Action
Fish boat	Go away
Passenger boat	Go away
Tank	Bomb

Figure 3: Example Object Detection and Decision Making by Earth Orbiting Satellite

2.3. Distinguishing Features

The increased uncertainty in autonomous software is due to increased discontinuities in the state space and decreased situational awareness.

In non-autonomous control systems, there are typically a sequence of activities or scenarios that are implemented. Autonomous software can jump between sequences in satisfying goals. Even though this behavior is still in existence for non-autonomous software, the scope of the behavior increases for autonomous software. Therefore, there's less determinism in the state of the software for autonomous software than there is for non-autonomous software. The risk posed by this can be mitigated by additional V&V of the software.

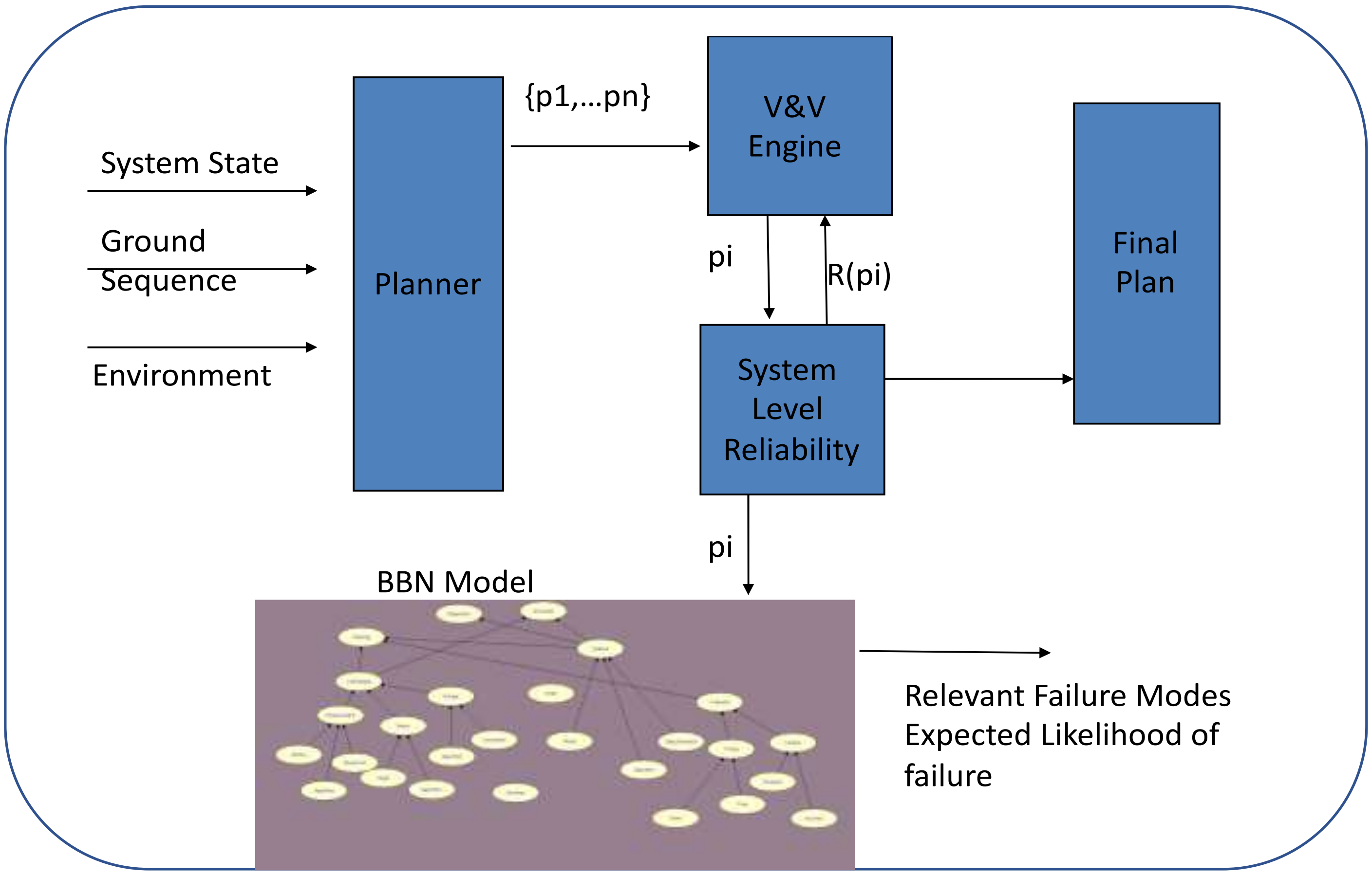


Figure 4: Proposed Architecture for Incorporating System Reliability into V&V Considerations

Situational Awareness

One of the most cited models of situational awareness has been suggested by Endsley M. [14]. In this model, situational awareness is defined as the perception of environmental elements and events with respect to time or space, the comprehension of their meaning, and the projection of their future status.

Situational awareness has been recognized as a critical, yet often elusive, foundation for successful decision-making across a broad range of situations. Lacking or inadequate situational awareness has been identified as one of the primary factors in accidents attributed to human error [15].

The situational awareness of an autonomous system is different as compared to humans. In the case of autonomy for space applications, the knowledge required for comprehending the perceptions and projecting into the future may be difficult to anticipate and code into the system a priori as there is uncertainty about the environment. This poses a risk which is typically mitigated by initially including the ground teams in the loop and gradually increasing the level of autonomy, for instance, in the case of the Mars rovers [16].

2.4 Risk Analysis Use Case

Sugawara [8] presents an example of an earth observing satellites that detects ships and identifies the type of ship as a fish boat (with a likelihood of 16%), a passenger boat (14% likely) and a tank (70% likely) of the type. They suggest that if the detection is incorrect, it could lead to a failure.

We elaborate on this example by assuming a decision model for this satellite where if the object is a fish boat or a passenger boat, the satellite goes away but if it is a tank, it bombs the tank. As seen on the right-hand side of Figure 3, the decision is to either bomb or to go away. If it bombs, there’s a chance that there may be civilian fatalities if in fact it were a fish boat or a passenger boat. If the object was a tank, then the decision is correct and the enemy has been destroyed. If the satellite makes the decision to go away, then if it were a fish boat or a passenger boat, it has made the correct decision and avoided conflict. If, however, it was actually a tank, then it has missed the enemy target.

Considering the three elements of risk, we can assess the risk associated with the autonomous object detection and decision making as follow:

1. *What can go wrong?* It can make the wrong decision
2. *What is the likelihood of making the wrong decision?*

Using the Law of Total Probability, we find the likelihood of an incorrect decision as follows:

P (incorrect decision) = P (incorrect decision| select fish boat) x P (select fish boat) + P (incorrect decision | select passenger boat) x P (select passenger boat) +P (incorrect decision |select tank) x P (select tank)

Let's assume that the Decision Maker has much higher value for not missing a tank than for accidentally bombing a fish boat or passenger boat so decides to bomb the object with this given probability distribution.

P (incorrect decision) = P (incorrect decision| select tank) * 1= P (not being tank| 70% tank prior likelihood)

3. *What is the consequence?* Civilian casualties associated with the fish boat or passenger boat.

3. SYSTEM LEVEL RELIABILITY

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [13]. Software has a greater level of decision-making authority in autonomous software and these decisions may lead to system level unreliability even in the absence of software faults. System reliability is time dependent and even though decisions that cause higher levels of system degradation may appear optimal in the short term, they could prevent the system from achieving its goals over the entire life cycle of the mission.

While stochastic modeling methods for the determination of the optimal sequence of activities to be performed by a rover on Mars have been developed [9], they do not address the overarching system unreliability caused by autonomy software. Typically, on Mars rovers, there are safety constraints that need to be met. These constraints ensure that the rover does not run out of power or make any movements that lead to damage of the hardware. These constraints are binary – the rover action is either deemed safe or unsafe. Reliability, however, is a likelihood and therefore can have a range of values in the interval [0,1]. Each course of action takes a given toll on the reliability of the rover and has different implications for its performance over time. For instance, one action may cause a larger degradation in the wheels than another. Or one course of action may require a more complex software procedure than another and hence an increased likelihood of error.

We therefore propose an architecture that takes into account, not just the autonomy software, but the system in which the autonomy software is operating. This proposed architecture is shown in Figure 4 and the context is an autonomous planner.

As seen in this figure, the planner takes as input the system state, ground sequence and the information obtained from its sensors regarding the environment. Based on these inputs, it generates a set of plans. These plans are run through the V&V engine to test it for completeness and optimality. A complete plan is a plan that satisfies all the necessary constraints in terms of the activities that the rover needs to perform in order to be safe. An optimal plan is one that can achieve all the activities with the least amount of resources. This plan can then be assessed to determine how it affects the system reliability. The system Reliability, which is nominally shown using a Bayesian Belief Network (BBN) model assesses the risks involved for the system for a given plan and their expected likelihood of occurrence. Based on this assessment, the plan may or may not satisfy the reliability constraints for the system. If not, there is a need for another iteration between the planner and the V&V engine until a plan is developed which is not only complete and optimal but also safe. That is the final plan.

4. RISK MANAGEMENT

The Continuous Risk Management Process used at NASA is shown in Figure 5 [11]. This process entails the identification, analysis, planning, tracking and control of risks. We have explored several methods for risk identification for autonomy software. The approach explained in section 3 calls for deliberating on the perceptions, decisions and actions that are taken by the autonomous system and software and the uncertainties associated with them. Another approach for identifying the risks is to deliberate on the key areas of uncertainty associated with autonomy software, in terms of the non-determinisms in the state space and the situational awareness of the autonomous system. Two methods for the analysis of these risks has also been suggested. One method is finding the likelihood and consequence of the risk elements identified. This is demonstrated with a use case in section 4. Another method is to determine the system level reliability of the system that is operating with autonomy software. Once the risks are identified and analyzed, the risk engineers or managers can use this information for planning purposes. This planning would involve making decisions about whether or not the risks are acceptable and finding an approach for mitigating or reducing the likelihood or consequence of these risks. One approach for mitigating the risks is to use testing or analytical model checking techniques, as suggested in section 2. Tracking and controlling the risks is an ongoing activity that is performed over the lifecycle of the autonomous software and system.

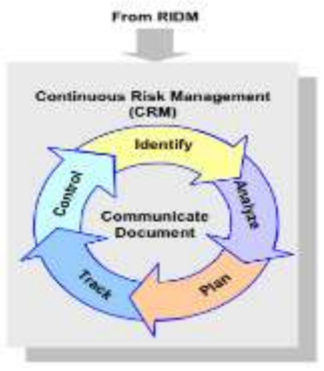


Figure 5: NASA Continuous Risk Management – NPR 8000.4B

5. SUMMARY & CONCLUSIONS

In this paper, we have provided an overview of autonomy software, in terms of how it is defined, classified and the methods used for its verification and validation. Using the definition for autonomy and some of its distinguishing features, we have suggested methods for identifying risks related to autonomous system. We have further extended the definition of risk and reliability to autonomy software and suggested an approach for risk assessment of autonomous software and an architecture for the consideration of system level reliability implications of autonomy software. The risk assessment is demonstrated using an example use case. Finally, each of the proposed methods and the connection between them are clarified in the context of the NASA Continuous Risk Management Process.

ACKNOWLEDGEMENT

The research reported in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA. The author gratefully acknowledges the sponsorship of the Software Assurance Research Program (SARP).

REFERENCES

1. Vincent Boulanin, “Mapping the Development of Autonomy in Weapon Systems; A primer on Autonomy”, Stockholm International Peace Research Institute, 2016
2. E. Gat, “Autonomy Software Verification and Validation Might Not Be as Hard as it Seems”, 2004 IEEE Aerospace Conference Proceedings
3. Jonsson, R. Morris, L. Peterson, “Autonomy in Space Current Capabilities and Future Challenges”, AI Magazine, Volume 28, No. 4. Winter 2007
4. https://www.techrepublic.com/article/autonomous-driving-levels-0-to-5-understanding-the-differences/
5. Eric Sholes, Mike Cole, Jason Rupert, Tony Colquitt, Matt Davis, and Justin Williams, “Analysis of UAV Behaviors via Simulation (and Live Flight)”, AIAA-20056199, 2005 AIAA Modeling and Simulation Technologies Conference and Exhibit, 2005.
6. C. Pecheur, “Verification and Validation of Autonomy Software at NASA”, The NASA STI Program Office NASA/TM-2000-209602
7. Stanley Kaplan and B. John Garrick, “On the Quantitative Definition of Risk”, March 1981, Journal of Risk Analysis.
8. Sugawara, K. “JAXA’s activity for assurance of autonomy spacecraft”, Assurance of Autonomy for Robotic Space Missions Workshop, August 2018, Pasadena, CA.
9. Wayne Chi, et. al., “Optimizing Parameters for Uncertain Execution and Rescheduling Robustness. ICAPS 2019: 501-509
10. Lorraine Fesq, et. al., “JPL Strategic Plan”
11. Agency Risk Management Procedural Requirements, Office of Safety and Mission Assurance, NASA Procedural Requirements NPR 8000.4B
12. Clark, Paula Pingree, Garth Watney, Autonomy & Control Section 345, Jet Propulsion Laboratory, “Levels of Autonomy Technical Implementation Peer Review (TIPR)”
13. Michael R. Lyu , *Handbook of Software Reliability Engineering*. McGraw-Hill publishing, 1995, ISBN 0-07-039400-8.
14. M. Endsley, “Situatioal Awareness Misconceptions and Misunderstandings”, Journal of Cognitive Engineering and Decision Making 2015, Volume 9, Number 1, March 2015, pp. 4-32
15. Hartel, Smith, & Prince, 1991; Merket, Bergondy, & Cuevas-Mesa, 1997; Nullmeyer, Stella, Montijo, & Harden, 2005
16. Estlin, Tara, et al. *Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution*. September 15, 2006.

BIOGRAPHIES

Leila Meshkat, Ph.D.
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA, 91109
leila@jpl.caltech.edu

Leila Meshkat is a Senior Engineer at the Jet Propulsion Laboratory (JPL)) and an adjunct lecturer at the Astronautics Department at the University of Southern California. During the course of her career at JPL she has conducted and led numerous Risk and Systems engineering tasks. She holds a Ph.D. in Systems Engineering from the University of Virginia, an M.S. in Operations Research from the George Washington University and a B.S. in Applied Mathematics from the Sharif University of Technology.