



WRT#:1019 Adaptive Cyber-Physical-Human Systems Testbed

Technical Report SERC-2021-TR-002

February 24, 2021

Principal Investigator:

Dr. Azad M. Madni, NAE

University of Southern California

Co-Principal Investigator:

Dr. Daniel A. Erwin

University of Southern California

Research Team:

University of Southern California :

Dr. Ayesha Madni

Edwin Ordoukhanian

Parisa Pouya

Sponsor:

Department of Defense

Copyright © 2021 Stevens Institute of Technology, Systems Engineering Research Center

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 [HQ0034-19-D-0003, TO#HQ003420F0027].

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).

No Warranty.

This Stevens Institute of Technology and Systems Engineering Research Center Material is furnished on an “as-is” basis. Stevens Institute of Technology makes no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material. Stevens Institute of Technology does not make any warranty of any kind with respect to freedom from patent, trademark, or copyright infringement.

This material has been approved for public release and unlimited distribution.

TABLE OF CONTENTS

Table of Contents	iii
List of Figures	iv
List of (Tables, Sequences).....	iv
Executive Summary	1
Personnel.....	4
Research Accomplishments	5
Testbed Ontology and Architecture.....	5
Hardware Design and Development	8
Developing 3D Models	10
3D Printing Components	11
UAV Localization for Indoor Flight.....	13
Testbed Modeling Library.....	16
State-Machine Models.....	16
Probabilistic Models	18
Deterministic Models	28
Testbed Dashboard.....	29
Scenario Creation Tool and Framework	30
Resources Available to Share	31
Summary and Way Ahead	31
Conclusion.....	32
Appendix A: List of Publications Resulted	33
Appendix B: Cited and Related References [Examples].....	34

LIST OF FIGURES

Figure 1 Testbed Ontology	6
---------------------------------	---

LIST OF (TABLES, SEQUENCES)

Table 1 Observation Model Employed for an Exemplar Scenario	23
---	----

EXECUTIVE SUMMARY

The mission of the SERC is to research and analyze advanced and emerging systems engineering practices and relevant technologies to address the full spectrum of DoD systems, from capability areas, enterprise systems, and Systems-of-Systems, down to cyber-physical systems and subsystems, with the goal of ensuring consistency and systems engineering excellence throughout the system's lifecycle. The research focus of the SERC is producing new and improved SE methods, processes and tools to advance the practice of systems engineering within DoD. This particular research task is concerned with the creation of a Model Based Systems Engineering (MBSE) testbed comprising methods and tools that facilitate *demonstration*, *integration*, and experimentation with a variety of models and algorithms that support MBSE.

This report documents the accomplishments in creating a minimum viable testbed for MBSE research. We define a *minimum viable testbed* as the minimum set of functions needed to support demonstration, integration, and experimentation. The testbed that we created comprises components that allow research teams to rapidly initiate activities associated with the engineering of autonomous vehicles (systems), an area of shared interest among a subset of SERC researchers. These components include: scenario builder, scenario-driven dashboard, systems modeling techniques, optimization algorithms, and hardware specifications (for experimental subscale vehicles and related equipment).

The key benefits of the testbed for MBSE researchers are:

- *Easy entry into field*: A research team will be able to start quickly with “zero-dollar” investment by installing the software and running prebuilt simulation scenarios
- *Reduced Risk in Experimentation*: Experimental work can be initiated using known, proven designs, without the need to design new vehicles and become expert “makers”
- *Easy comparison*: It will be possible to compare research results from different research teams in a straightforward way
- *Enablement of best practices*: Rapid dissemination of extensions (new and modified scenarios and vehicles) to the SERC research community

BACKGROUND

Engineering schools at major universities have begun experimenting with different types of system models and digital twin technology as part of their MBSE research. Currently, they spend significant time and resources to develop “one-off” testbeds to conduct analytic studies, evaluate system properties (e.g., adaptability, resilience), and make limited predictions. This is also the case with SERC universities conducting MBSE research. The testbeds of the different SERC universities are based on somewhat different assumptions and technologies. Consequently, their research efforts are difficult to replicate, while their models do not interoperate with those of others.

Against the foregoing backdrop, this research task was concerned with developing a flexible testbed that enables demonstration of new MBSE advances, facilitates hardware-software integration, and supports experimentation with different models and algorithms. SERC researchers working on MBSE can add models to the testbed’s open-source repository, thereby enabling growth and progressively greater utility of the testbed.

The testbed is fundamentally a *cyber-physical-human systems* testbed, in that it supports the modeling, analysis and implementation of both autonomous and human-in-the-loop systems without researchers getting bogged down in implementation details. The testbed is also a methodology-neutral platform to showcase research products on demand to both internal and external customers. It facilitates comparison of models and algorithms, and integration and testing of virtual system models / digital twins. It facilitates interoperability among different models produced by SERC researchers working on unmanned aerial vehicles (UAVs).

This research adds to the SE body of knowledge by creating a minimum viable hardware-software testbed. This research supports the SERC’s research strategy for SE Transformation and Trusted Systems. The research draws on results from RT-183, Next Generation Adaptive Cyber-Physical Human Systems and RT 166/RT210 Formal Methods for Resilient Systems Design Using Flexible Contracts Approach.

RESEARCH GOAL

Our research goal was to create a MBSE testbed comprising modular components that allow research teams to rapidly initiate research in engineering UAVs, an area of shared interest among SERC researchers. To this end, we specified and developed the following testbed components:

- Scenario
- World
- Vehicle
- Agent
- Experiment
- Laboratory
- Project Scope

SCENARIO CREATION TOOL

We developed a software tool that facilitates the creation of new scenarios and modification of existing scenarios. This tool has: a scripting capability that allows scenarios to be described in a simple text-based language; and a graphical layout capability that allows users to rapidly “sketch” simple scenarios using the mouse.

DASHBOARD TOOL

We developed a customizable dashboard program that allows the user to import scenarios, provide default views such as a plan view (or “mission view”) of the world showing all vehicles; provide views from one or more vehicle cameras; present state indicators for all vehicles of interest; and controls to start and stop an experiment. It is possible to customize the dashboard layout either graphically (by moving, scaling, adding and deleting elements), or by modifying the layout script.

STANDARD VEHICLE

We produced a standard physical quadcopter model. This model is an engineering refinement of previous research developed models, with the key differences being clean design, ease of assembly, and crashworthiness (ability to crash without damage). On the software side, a discoverability component is provided to automate vehicle network communication by providing a “plug and play” capability, so that manual setup of vehicle IP addresses are no longer needed.

PREDEFINED SCENARIOS

A key feature of the testbed is a “starter kit” that allows researchers to quickly start using the testbed. The starter kit comprises several predefined scenarios that researchers can use “out of the box” in a simulation environment, and later in the physical laboratory. In the scenarios, multiple vehicle control algorithms are provided, including rule-based techniques and Partially Observable Markov Decision Process (POMDP) model-based algorithms. The scenarios and control algorithms are documented to illustrate how the scenarios and algorithms can be modified, and how new scenarios and algorithms can be created.

The testbed is designed such that researchers will be able to expand its capabilities to eventually support digital threads that can enable inter-university research. The scalable testbed architecture allows extensions to be made over time to include utilities, APIs, analysis and measurement tools, and additional system modeling tools. At that point, researchers will be able to evaluate the benefits of using a commercial cloud service provider for computers and basic services.

PERSONNEL

Research Team: (List all team members, past and present.)

Name	Organization	Labor Category	Contact
Prof. Azad M. Madni	USC	Principal Investigator	Azad.Madni@usc.edu
Prof. Daniel A. Erwin	USC	Co-Principal Investigator	erwin@usc.edu
Dr. Ayesha Madni	USC	Project Manager	amadni@usc.edu
Edwin Ordoukhanian	USC	Graduate Research Assistant (integration)	ordoukha@usc.edu
Parisa Pouya	USC	Graduate Research Assistant (system modeling)	pouya@usc.edu
Shatad Purohit	USC	Graduate Research Assistant (MBSE framework)	shatadtp@usc.edu

Government:

Name	Organization	Role	Contact
Nadine Geier	OUUSD(R&E)	SERC Program Manager	Nadine.m.geier.civ@mail.mil
Rhyan Johnson	OUUSD(R&E)	Support to SERC PM	Rhyan.d.johnson.civ@mail.mil

RESEARCH ACCOMPLISHMENTS

Our research accomplishments are on multiple fronts. To scope the minimum viable testbed and the problem domain, we defined a testbed ontology to support experimentation. Defining this ontology was an iterative process driven by testbed use cases.

We also developed: a scenario builder, : a scenario-driven dashboard, a modeling library and assembled / integrated the requisite hardware. Each of these areas are discussed in subsequent sections.

TESTBED ONTOLOGY AND ARCHITECTURE

With increasing system complexity, interest in ontologies as a means to manage complexity has grown significantly (Madni, 2020; Ordoukhanian & Madni, 2020). Ontologies are being employed in a variety of applications such as Requirement Analysis, Metamodeling, Enterprise integration, Application Interoperability, Architecture development, and Model Based Systems engineering (MBSE) (Madni, 2020).

Ontologies are formal and explicit specifications of a shared conceptualization. Conceptualization includes objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them (Madni, 2020). It is an abstract, simplified view of the world that we wish to represent for some purpose. A well-constructed conceptualization does not change when the world changes.

Figure 1 depicts the ontology for an integrated hardware-software integration and experimentation testbed. Each concept in the ontology is presented in this diagram. Figure 2 presents the ontology that enables a scenario-adaptive system and attendant concepts.

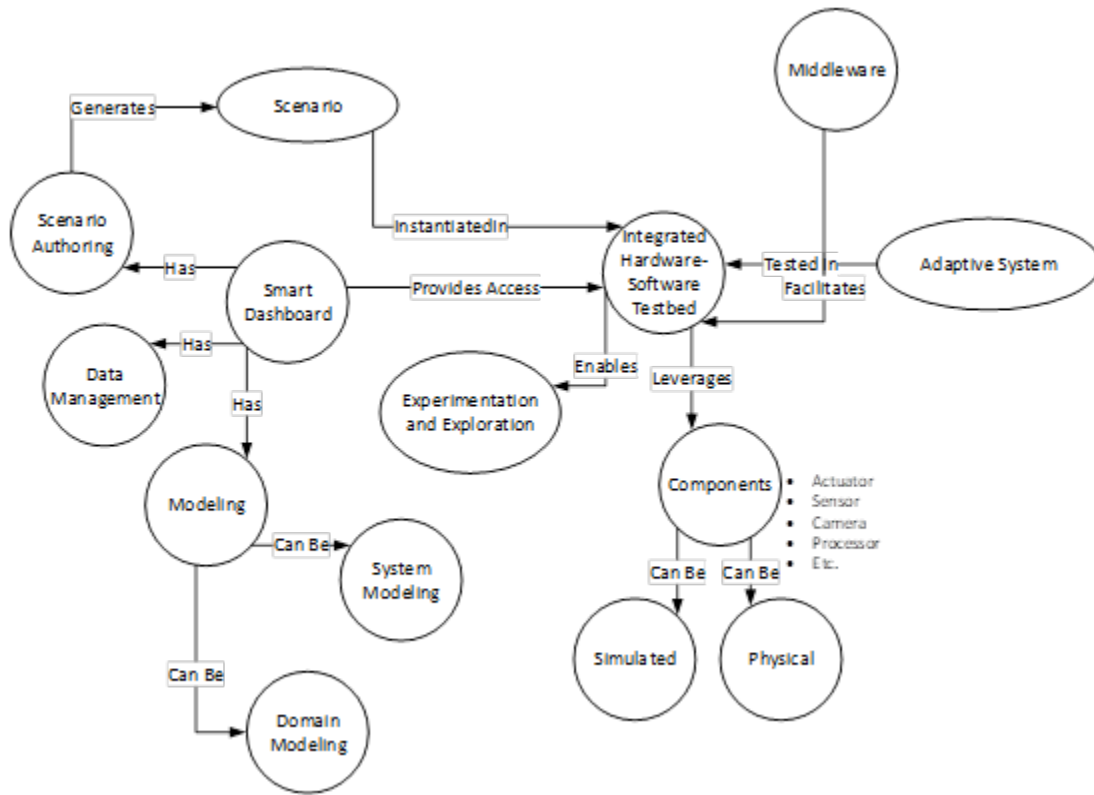


Figure 1 Testbed Ontology

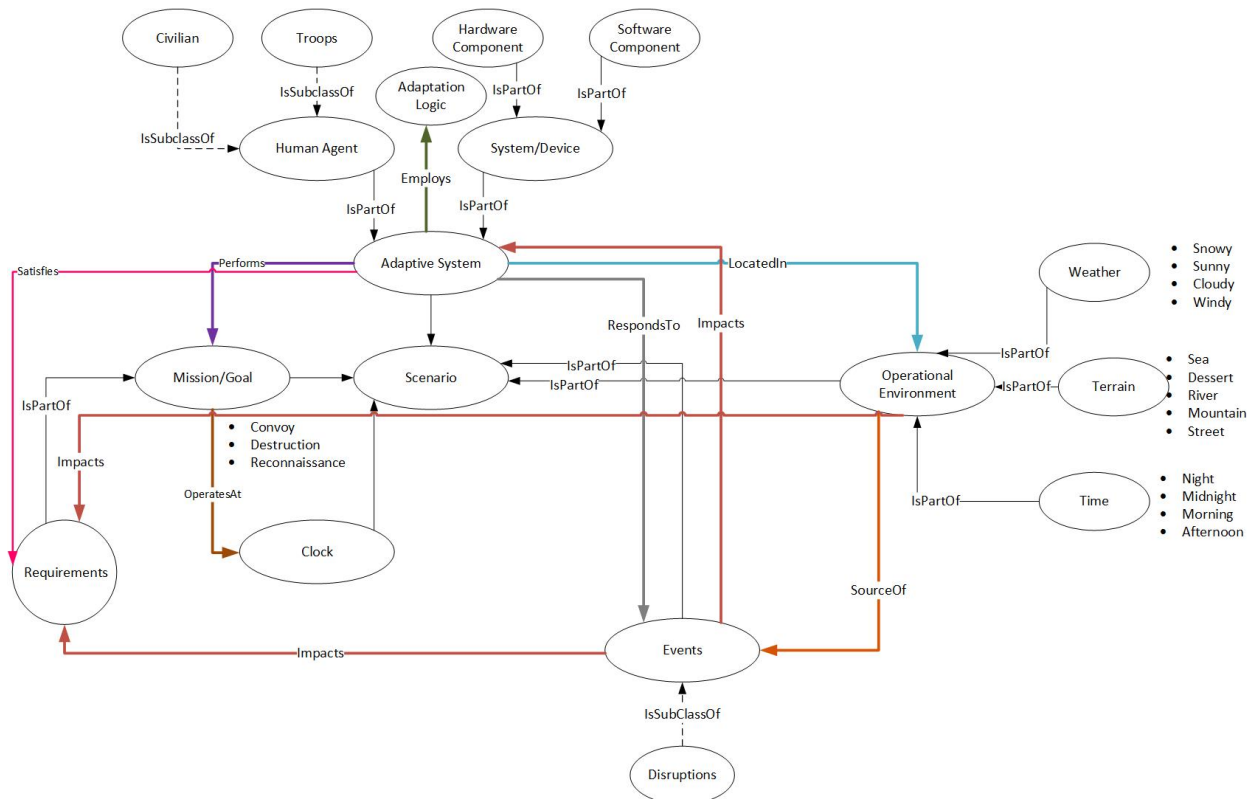


Figure 2 Scenario-Adaptive System Ontology

Figure 3 presents the Testbed Architecture based on the ontology presented in Figure 1. The testbed comprises several components.

- *Multi-purpose dashboard*: allows scenario scripting, experiment execution, system development, system modeling, domain modeling, data analytics, and machine learning
- *Hardware*: includes complete system (UAV & ground vehicle) and sub-systems (actuators & servos)
- *System/Domain modeling*: includes system models (SysML diagrams & state machine), math models (pomdp & dynamic equations), and domain models
- *Virtual environment*: includes simulation environment, simulated sensors, and virtual objects
- *Physical environment*: includes physical obstacles, tracks, environmental objects, and other systems such as vehicles and UAVs
- *Database/Repository*: contains algorithms, scenarios, data, objects, and experiments
- *Configuration management*: keeps track of data based on date and type of experiments

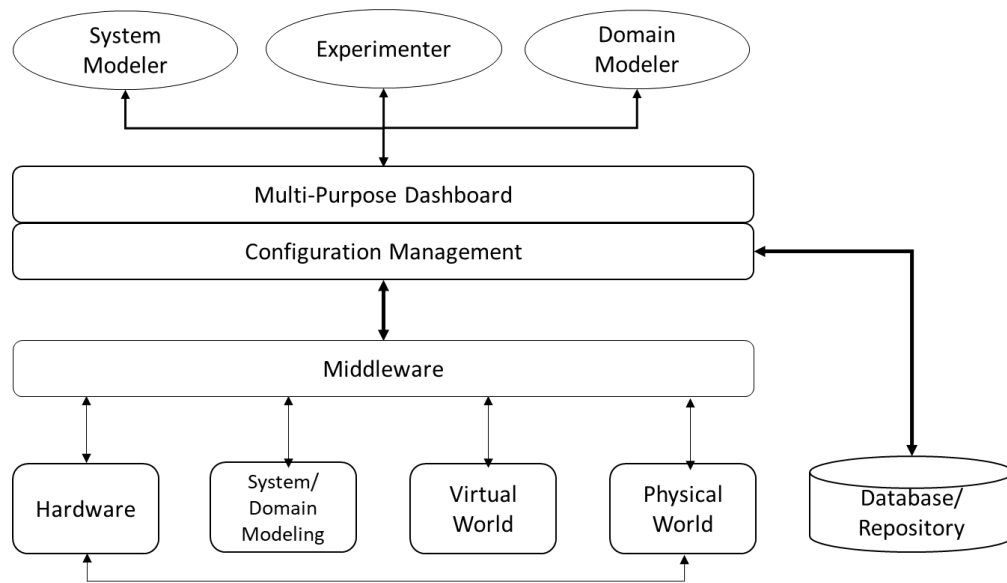


Figure 3 Testbed Architecture

Ontologies are developed using ontology editors such as Protégé. We imported our ontology into protégé. Protégé enables plugins for inserting various reasoning tools to check consistency and completeness of the ontology. In this version, we have selected HermiT reasoner for checking our developed ontology. The reasoner looks for items such as class hierarchy, object property hierarchy, data property hierarchy, class assertions, object property assertions, and data property assertions in the ontology. Figure 4 shows a snapshot of the ontology from Protégé.

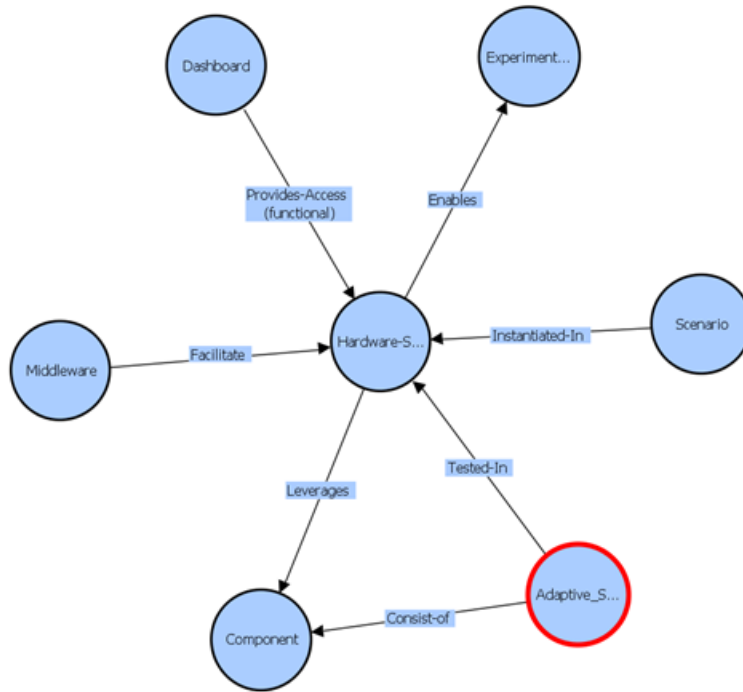


Figure 4 Ontology Snapshot from Protégé

HARDWARE DESIGN AND DEVELOPMENT

One of the goals of the project was to define a process for researchers to rapidly develop and make operational a quadcopter, a small UAV. We defined building blocks for rapid assembly of a standard UAV (Figure 5).

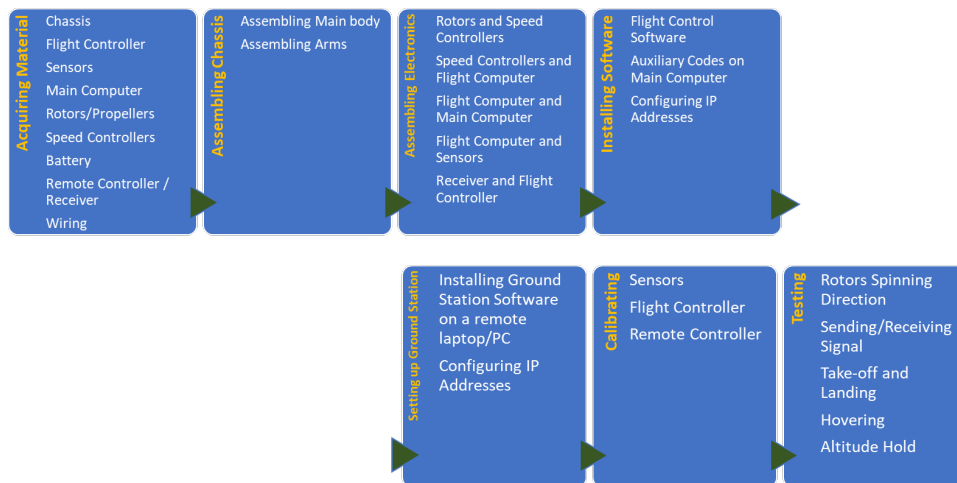


Figure 5 Process for Rapid Assembly of Standard UAV

We leveraged our experience in developing quadcopters in our previous efforts. We focused on creating a robust, crash-proof quadcopter suitable for indoor flight. As shown in the process above, the first step in assembling a quadcopter begins with a chassis.

When testing quadcopters indoors, one runs the risk of unexpected quadcopter behaviors leading to crashes, that result in broken components and/or damaged electronics. Depending on the severity of the damage, expensive replacements and/or delay in flight testing can result.

Therefore, a quadcopter needs to have a suitable frame and appropriate safeguards for indoor testing. To this end, we developed a quadcopter frame suitable for indoor flight. The following were the main requirements that we addressed in the new design:

- *Rugged*: Can suffer severe crashes without the need for repairs or replacement of electronics
- *Modular*: Can be adapted to a wide range of hardware for different teams or test cases
- *Accessible*: Can be created using a standard 3D printer and standard hardware

We reviewed typical quadcopter failure points during severe crashes and compared these against the relative time and cost of replacement. Scouring design forums and talking to pilots, we determined that propeller damage was by far the most likely to occur, while broken arms and motor mounts were both likely in a crash. Figure 6 summarizes possible damage categories and categorizes them based on the severity of the crash.

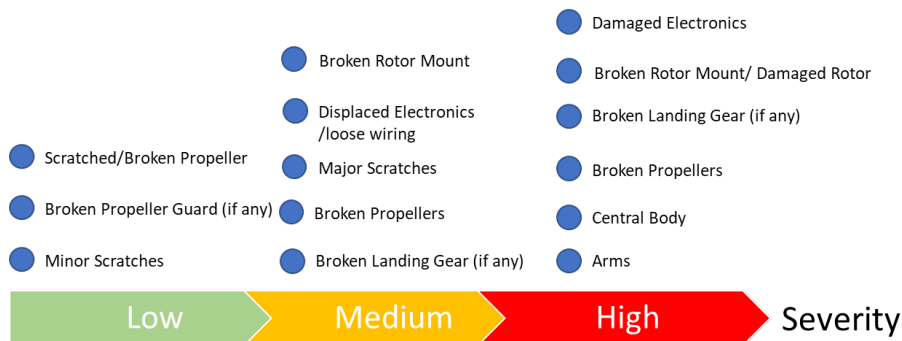


Figure 6 Quadcopter Damage and Severity

We conducted further research to evaluate the work completed at the intersection of 3D printed quadcopter frames, and those that are specifically designed with durability in mind. Typical quadcopters, with durability as the focus, tend to be too small, too expensive, and not modular enough for research purposes. One design alternative stood out: [the EPFL quadrotor](#) that utilizes a combination of flexible arms and magnets such that the vehicle is rigid during flight, yet under the loads of a crash, becomes flexible and dissipates impact energy. While this frame meets the durability aspect of our design goals, a larger and more modular design was necessary.

To 3D print the quadcopter frame, we researched suitable 3D printing filaments. With a combination of rigid and flexible components to be printed, generic TPU and PETG filaments were selected for flexible and rigid components, respectively. They were selected because of their durability and ease of printing in comparison to alternatives such as TPE and Nylon.

The final design focused on utilizing a combination of rigid and flexible components on a scale small enough to be printed on a 20 x 20 cm build volume. Quadcopter arms were made of flexible TPU filament in a honeycomb pattern to facilitate rigidity in the vertical direction and flexibility for lateral impacts. Each arm was printed in two pieces. Both sides were attached to the rigid body and motor mount via bolts for a permanent connection. A combination of magnets and elastic fillaments can be used to allow further flexibility and impact absorption in later versions. The center body mount for electronic components can remain rigid with the exception of flexible bumpers.

Developing 3D Models

Through iterative development, we created a quadcopter fulfilling the following criteria: reduce the mass of the finished product and increase part manufacturability. Modifications throughout the effort included reducing the size of pieces, changes in infill density, redesigns of piece geometry to reduce necessary support material, and various other changes. The modifications were centered around three areas, frame body, propeller guards, and arms. Figure 7 shows final design of the quadcopter rendered in Fusion 360 software.



Figure 7 Final Design of Quadcopter

The body, or center of the quadcopter frame was shrunk and cut into smaller, buildable pieces. The result of this change is a decrease in mass and decrease in print time. There are a few components that must currently be mounted to the body of the frame and require the most protection in the event of a crash, including (in our case) a Raspberry Pi, flight controller board, a power distribution board, speed controllers, and a battery. Thus, mounting space required is minimal and mounting holes are only generated for the fore-mentioned components. However, the original CAD files will be provided for easy modification to specific needs with the use of the widely available Fusion 360 software.

3D Printing Components

We printed quadcopter components using our PRUSA MK3S 3D printer. The design focuses on the use of rigid mounting pieces made of PETG, and flexible components (such as arms) made of TPU. Figure 8 shows the quadcopter arm parts.

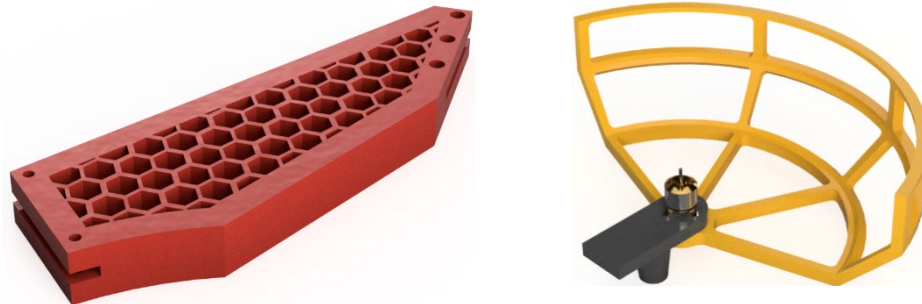


Figure 8 Quadcopter Arm Parts

Figure 9 shows 3D printed components with PETG filaments.

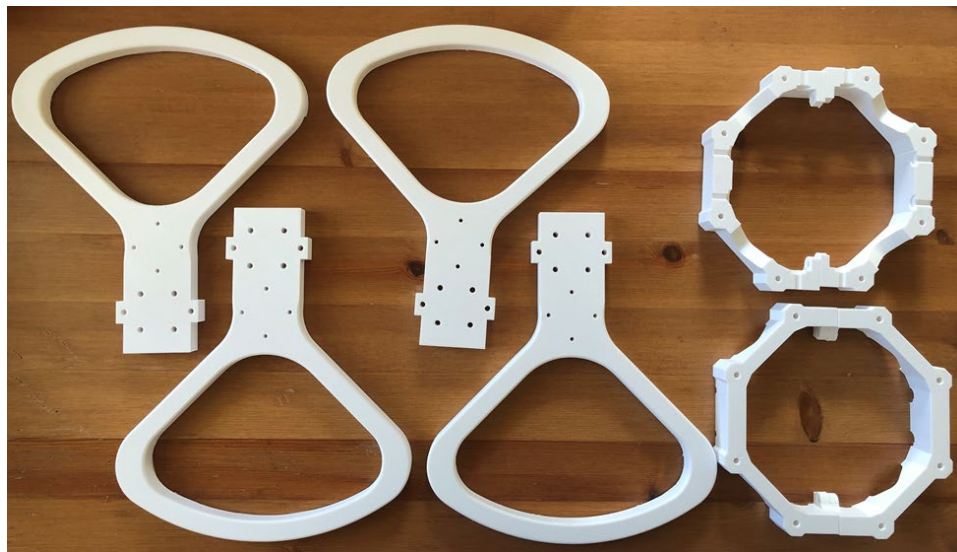


Figure 9 PETG Printed Components

The propeller guards, which also serve as motor mounts, were completely redesigned throughout the project. Landing gear and vertical bars were removed so that the pieces could be printed completely flat on the print bed. While this sacrificed some protection, this change strikes the right balance between protection, mass, and ease of printing. Prop guards are essential for indoor flight for both vehicle and operator safety and should not be completely removed.

The frame arms are made of tough, semi-flexible TPU filament printed in an extruded honeycomb pattern. The arms must be designed with enough rigidity to be stable in the air, while still flexible enough to bend without breaking in the event of a crash. This

balance is achieved not by changing the material, but instead by adjusting the thickness and size of the honeycomb walls. By printing arrays of test samples, we are able to fine-tune the part to our specifications. Figure 10 shows results of test prints to find suitable flexibility with honeycomb pattern and TPU filament. First column on the left has the most flexibility. First column on the right has the least flexibility. The middle column has more flexibility from the right column but less flexibility from the left column.

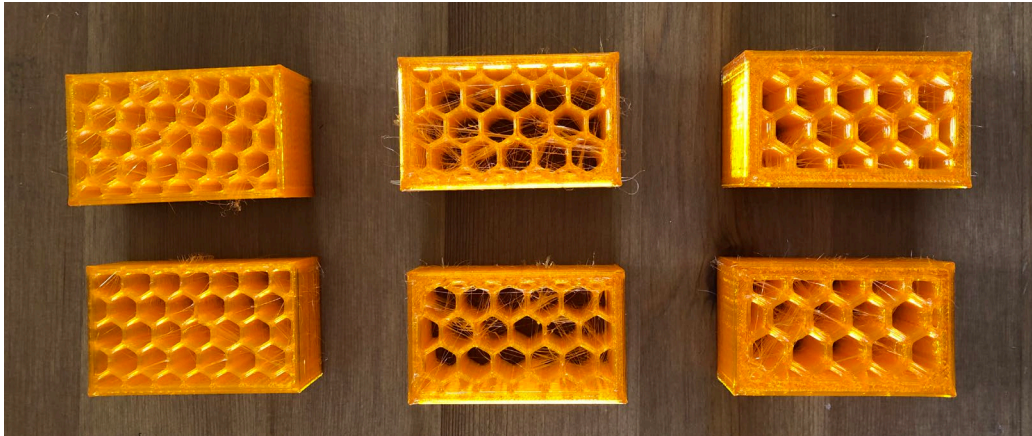


Figure 10 Flexible Honeycomb Pattern Test

Significant effort has been put into dialing in the correct settings for the Fused Deposition Modeling (FDM) printing of PETG and TPU test components. Print speed, layer thickness, bed temperature, nozzle temperature and more were adjusted to produce successful prints with our chosen filaments. Figure 11 shows the final print of a quadcopter arm using the flexible (TPU) filament.



Figure 11 3D Printed Quadcopter Arm using Flexible Filament

UAV LOCALIZATION FOR INDOOR FLIGHT

This section discusses the development of an indoor replacement for GPS for sensing absolute location of the vehicle. Our approach was to use upward-looking vehicle-mounted cameras which image fiducial markers mounted on the laboratory ceiling. Image

processing results in pose estimation (measurement of location and orientation with respect to the fixed markers) and thus yields absolute location which can be used to replace GPS. The goal was to be able to place the fiducial markers in any convenient manner, and determine their precise locations using a calibration procedure. We overcame this challenge with the following process.

A series of images of the ceiling was recorded from locations on the laboratory floor at the corners of one-foot tiles. The camera orientation was maintained fairly consistent with the use of an alignment jig (Figure 13) on which the camera was mounted. Direct determination of ceiling marker pose from these images cannot be done accurately, because the marker locations depend very sensitively on the camera orientation which is not known precisely. Both systematic and random variations of camera orientation are expected to be present.

Figure 12 shows our laboratory ceiling and the fiducial markers. The markers are sheets of paper printed with Aruco markers, black and white patterns which can be uniquely identified in a camera image.



Figure 12 Laboratory Ceiling and Mounted Aruco Markers

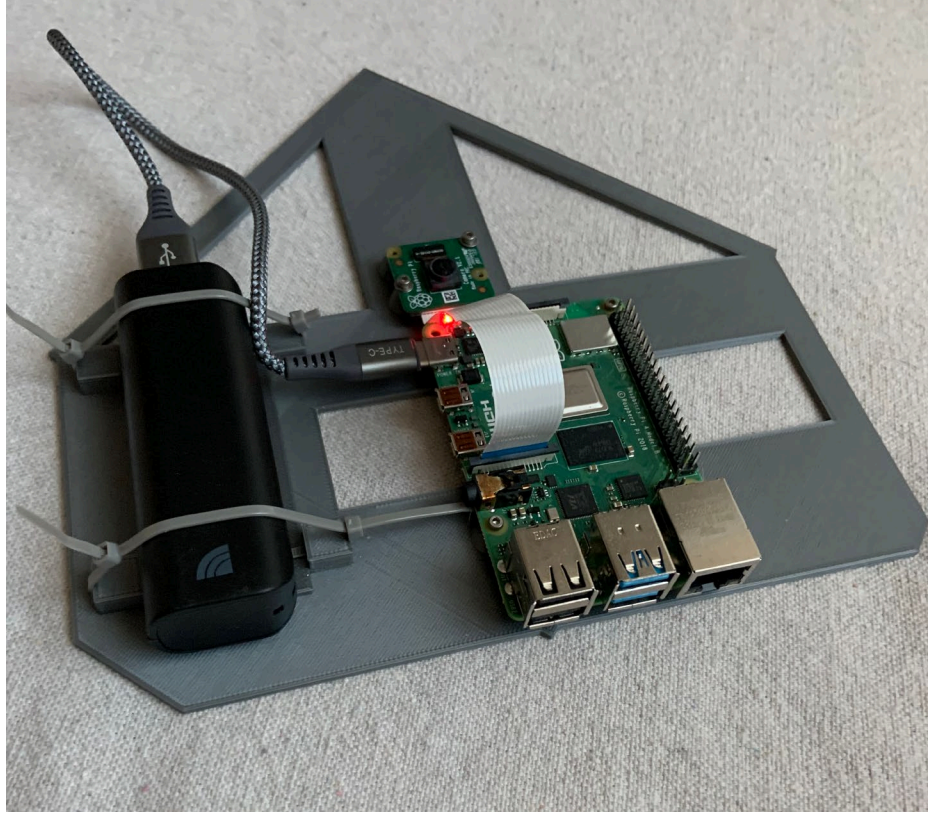


Figure 13 Calibration jig for Measuring the Locations and Orientations of Ceiling-mounted Aruco Markers

The calibration method consisted of analyzing pairs of calibration images taken at locations separated by two or four feet. Each image was analyzed using OpenCV's Aruco module to determine the pixel coordinates of each marker within the field of view. Using the camera projection model (intrinsic and distortion matrices, measured using standard checkerboard calibration), the marker corner locations were converted to rays (i.e., directions in 3D space). The ray directions are relative to the camera image plane, which is uncertain because of the uncertainty in camera orientation in each image.

For each marker that is seen in both images, each of the 4 corners is located at the intersection of two rays, one from each of the two camera locations. However, the rays computed from the corners' pixel locations do not in general intersect, but pass each other at a minimum distance of order an inch or so depending on assumed camera orientations. This nonzero distance (the triangulation error) is due primarily to the uncertainty in the camera orientation.

We implemented a procedure for determining the camera orientation for each of the two images. While varying six angles (the unknown roll, pitch and yaw angles for each of the two camera orientations, each varied between -5 and 5 degrees), the root-mean-square of the triangulation errors for all corners of all markers common to both images was minimized. To perform the minimization, the minimize function of the `scipy.optimize` Python package was used. In most cases this optimization results in an RMS triangulation

error of 0.01 to 0.03 inches, equivalent to an orientation error of about 0.005° to 0.015° for a ceiling height of 10 feet.

It should be noted that the optimization procedure determines the absolute 3D locations of the marker corners, in the same coordinates used to define the floor tile corner locations. For our purposes, to emulate GPS, the x-direction was measured to the north in inches from laboratory center; the y-direction was to the east; and z was straight down and was zero at floor level. The known marker dimensions were not used in the procedure, and in fact there is no need to have markers all the same size; this may be useful later. For example, if markers are mounted on walls as well as ceilings, the distance from a vehicle to a marker may vary by a great deal.

Results of the calibration method are shown in figure 14. The method gives absolute location accuracy of about 0.1 inch for most of the calibrated area, though the error increases to about half an inch in a couple of areas. Even the lowest accuracy is far better than that of GPS.

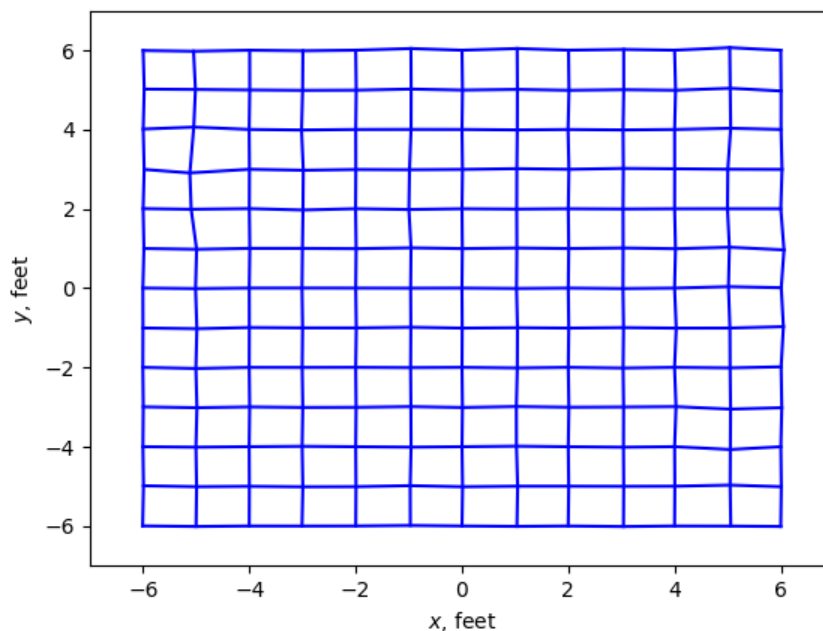


Figure 14 Accuracy of Measured x- and y-locations

The intersections of the line segments are the measured locations of the cameras for each of the calibration images. Since the images were taken at one-foot corner locations, perfect measurement would result in a rectilinear grid aligned with one-foot lines; the experimental results are close to this.

As discussed, the testbed includes various libraries. One of the libraries we had focused on during this effort was the modeling library. We have developed various models which can be used by researchers to jump start their research effort.

State-Machine Models

To address possible issues and challenges associated with the system's (quadcopter) physical health in the testbed, we focused on designing and developing a state-based model, where different states in the model represent specific system-health related situations. For this purpose, we addressed three different sources of health information that can affect the quadcopter's overall mission/goal in its environment. These sources are: 1- Battery voltage, 2- GPS (or location-related) signal, and 3- CPU load. To describe various states associated with the battery voltage, we used three qualitative variables: 1- Battery Normal, 2- Battery Low, and 3- Battery Dead. Both the GPS signal and CPU load were described using binary variables. Thus, the states of the GPS signal are: 1- No GPS (there is no GPS signal) and 2- GPS (GPS signal exists). Similarly, the states associated with the CPU load are defined as: 1- CPU Busy (there is a heavy load on the CPU or onboard computer) and 2- CPU Normal. Based on these variables and whether these variables are independent or dependent, 8 states associated with the system health were defined:

- S_0 – [Battery dead]
- S_1 – [Battery low, GPS]
- S_2 – [Battery low, No GPS, CPU busy]
- S_3 – [Battery low, Nor GPS, CPU normal]
- S_4 – [Battery normal, GPS]
- S_5 – [Battery normal, No GPS, CPU busy]
- S_6 – [Battery normal, No GPS, CPU normal]
- S_7 – Terminal state

The rationale for choosing the above-mentioned variable combinations for defining various states was the priority and importance of these variables in identifying the overall health state of the quadcopter. The battery life/voltage of the quadcopter has the highest priority in identifying the overall health-related state of the quadcopter. In other words, the quadcopter will be able to perform mission-related computations and calculations if there is power in the system. The next important factor in determining the health status of the quadcopter is the GPS signal. In our scenarios, we assumed that the quadcopter (system) was able to observe its location either using GPS (for outdoor flights) or Aruco markers (for indoor operations). However, there may be situations where the signal associated with the system's location/position is lost and, in these cases, the quadcopter needs to estimate its location/position in the environment. To achieve this goal, the quadcopter may use various approaches, such as motion equations or Kalman filter, but this may result in an additional computational load on the CPU or onboard computer. Thus, before proceeding with these computations, we needed to measure and evaluate the baseline computational load on the onboard computer.

After identifying various health states for the quadcopter, we also defined a set of high-level, heuristic actions that the quadcopter needs to perform when transitioning into a specific state in the model. The priority of these high-level actions is higher than mission-related or control actions because these actions are directly associated with the physical safety of the quadcopter. These actions are described as follows:

- A_0 – [Save data, Communicate location to central monitoring station, Land]
- A_1 – [Go home]
- A_2 – [Terminate POMDP, Estimate location, Go home]
- A_3 – [Estimate location, Go home]
- A_4 – [Maintain status quo]
- A_5 – [Reduce N (POMDP steps look ahead) or sampling rate, Estimate location]
- A_6 – [Estimate location]

After defining the states and actions for the state-based model, the model was defined as a Finite State Machine (FSM) as shown in Figure 15.

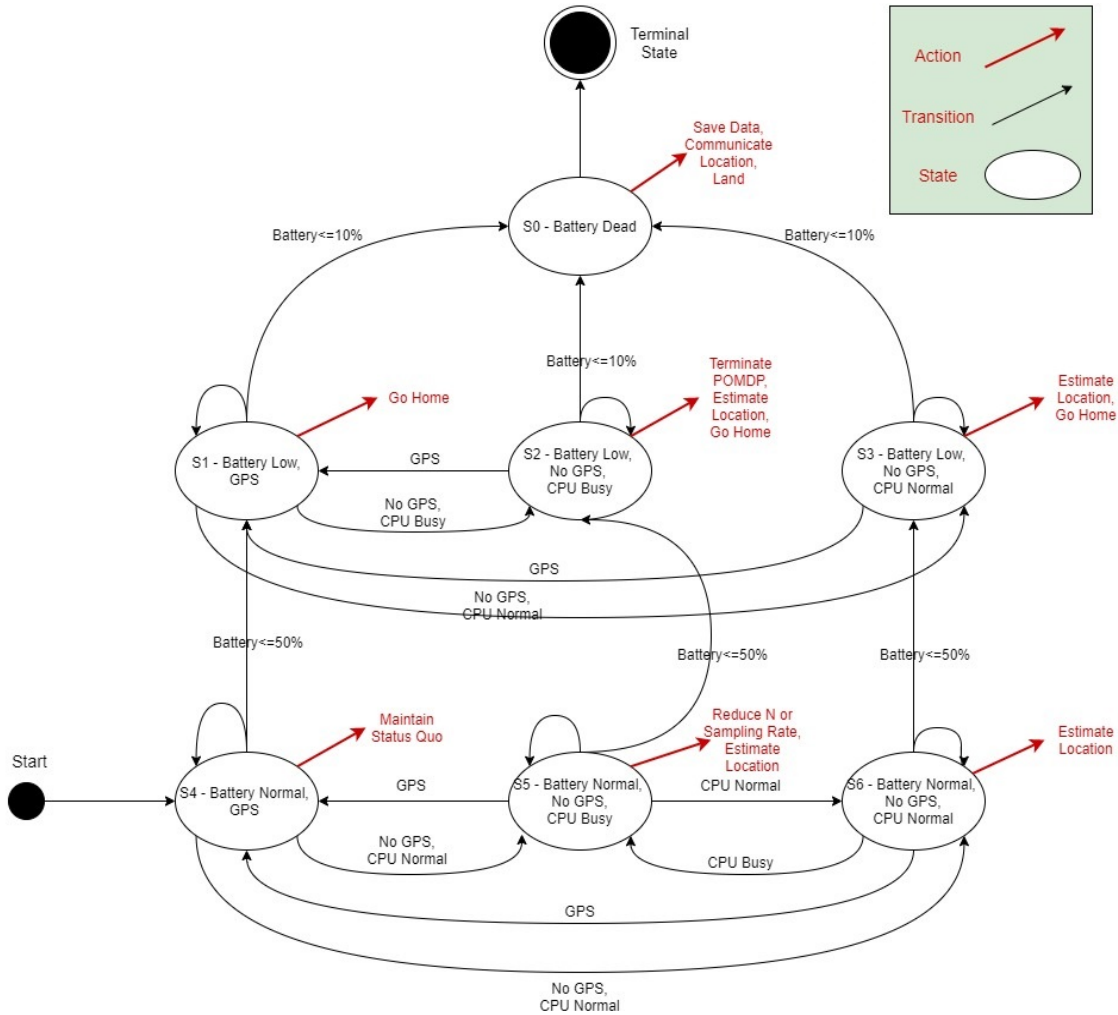


Figure 15 Quadcopter's Health Finite State Machine with Embedded Heuristic Actions

While there are no actions to control/reduce the battery usage in the quadcopter, we can reduce the computational load on the onboard computer by adjusting the sampling rate or planning and decision-making model parameters (such as N). To evaluate and measure the effect of model parameters and sampling rate on the computational load, we measured the CPU percentage for running a POMDP model with different N values and sampling rates in a simulated environment and sampled the CPU percentage during the simulation. The results of our simulated evaluation are demonstrated in Figure 16.

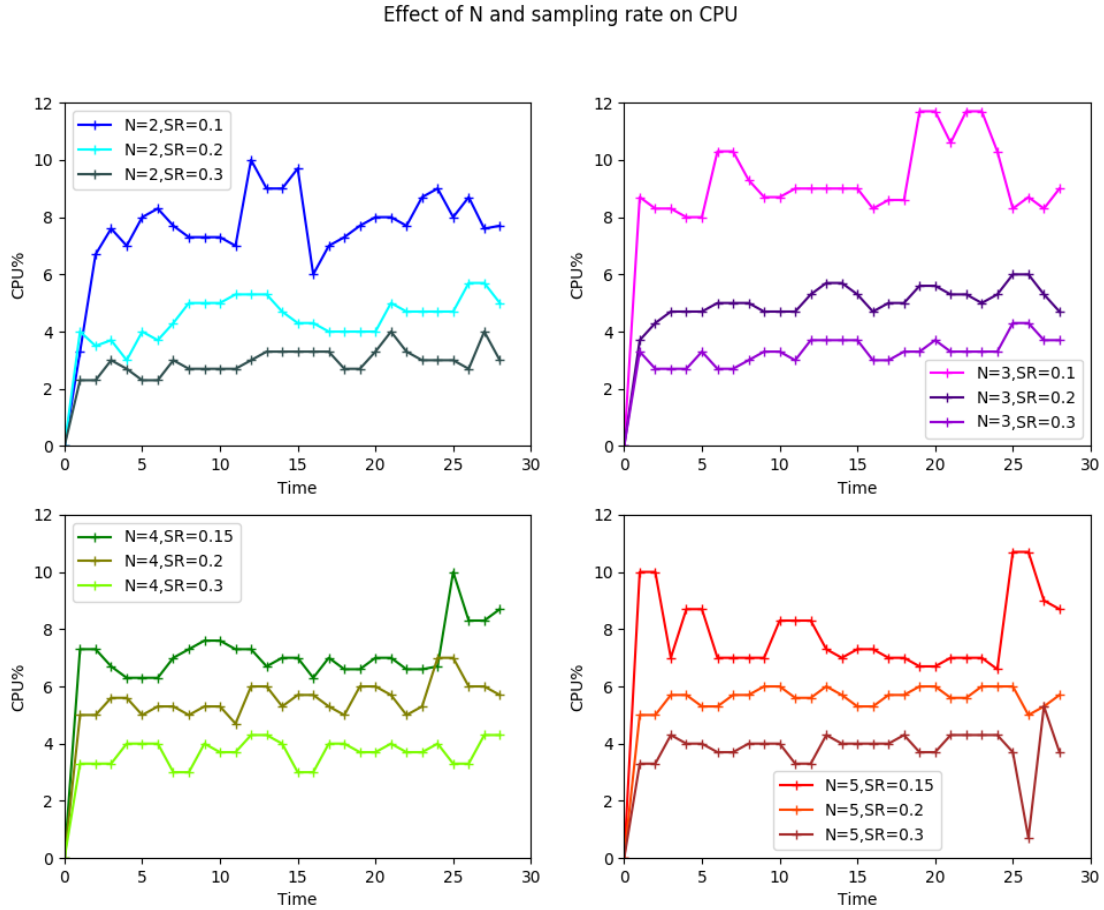


Figure 16 Evaluating the Effect of Various N Values and Sampling Rates on Computational Load

As shown in this figure, changing the sampling rate and N can dramatically influence the computational load and this is the rational for performing action A_5 when the quadcopter transitions to state S_5 .

Probabilistic Models

We created an exemplar POMDP probabilistic model to represent a UAV search and rescue mission scenario. The POMDP model can be adapted to other UAV scenarios, such as package delivery, and surveillance.

Search and rescue operations can greatly benefit from the use of autonomous aerial vehicles (UAVs)/quadcopters to survey the environment and collect information about any possible missing object (e.g., person) in the environment. There are various hardware-related limitations and environmental constraints and conditions that can add challenges to the problem. Limited sensor (e.g., camera) capabilities, noise in observations, and limited battery power are examples of undeniable hardware limitations. In addition, time constraints forced from the environment, wind gusts, different weather conditions, and shading effects caused from large objects (e.g., rocks and trees) on the ground in the environment are examples of challenges posed by the environment. The aforementioned limitations and constraints lead to a partially observable and uncertain environment. To address these challenges, we modeled the interaction between the quadcopter and its environment using a POMDP model. We created a belief map of the object's location in the environment, and updated the belief map continually based on collected evidence provided by the quadcopter. Figure 17 presents a description of the search and rescue mission.

Description of search and rescue scenario

- Scenario: Search and Rescue
- Mission/Goal: Survey a designated area for a missing object (e.g. person) and transmit object's location to a control unit
- Operational Environment:
 - Terrain: Roads + Mountains + Rocks + Trees + Missing object
 - Weather: Sunny
 - Time: TBD
- Events:
 - Disruptions: Low battery + Uncertainty in object detection and identification due to partial observability caused by shading affects of mountains, rocks, and trees
- Hardware-Software Components:
 - Software Components:
 - Model: Partially Observable Markov Decision Process (POMDP), observation model
 - Function: Belief update function, transition function, reward/penalty function, state to location transition function, location to state transition function
 - Algorithm: Online POMDP policy estimator (e.g. N-Step Look-Ahead), Coverage estimation
 - Packages: Pandas, NumPy (from Python)
 - Data Repository: Initial belief map, Initial UAV location, Initial object location, Observation model confusion matrix

Figure 17 Search and Rescue Scenario Description

Modeling the search and rescue scenario as a POMDP problem required a model of environmental factors that impact the system as well as a model for the interactions between the quadcopter and the environment.

The environment model for our application was a two-dimensional grid map, where the region/environment to be surveyed was divided into smaller segments (e.g., squares). Each segment (i.e., square) provided a unique location on the grid map. For instance,

given a map of the environment/area to be surveyed, the map can be divided into 100 segments as shown in Figure 18.

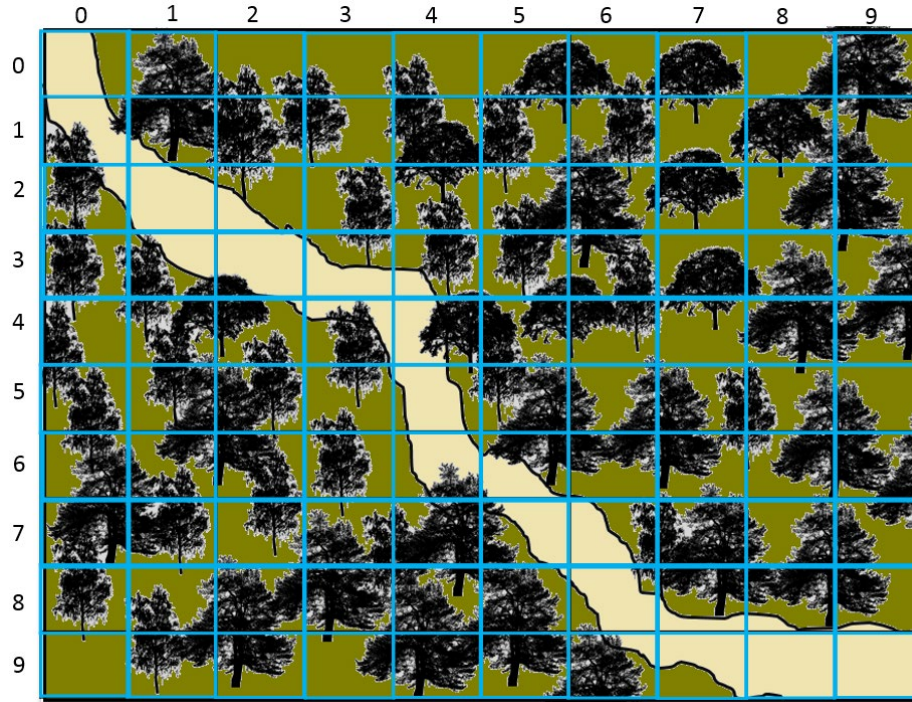


Figure 18 Example Grid Map

We assume that the target object (e.g., a missing person) is located in one of the segments on the grid map, and the quadcopter needs to surveil the area to locate that right cell. The grid map presents the state-space of the search and rescue POMDP model. To maintain the information on the probability of the target object location, the quadcopter maintains a grid-based probabilistic map (belief map) composed of the segments shown on the map. In the belief map, each cell contains the probability that the target object is located in that cell. At every time-step, when the quadcopter surveils a section of the grid map, the belief map gets updated based on the quadcopter's observations of that section.

To initialize the belief map ($b_{t=0}$), we assume that with 0.5 probability, the target object is located on/nearby the road regions, and with 0.5 probability it is located in the other regions. Based on this assumption, the initial belief map was constructed as shown in Figure 19.

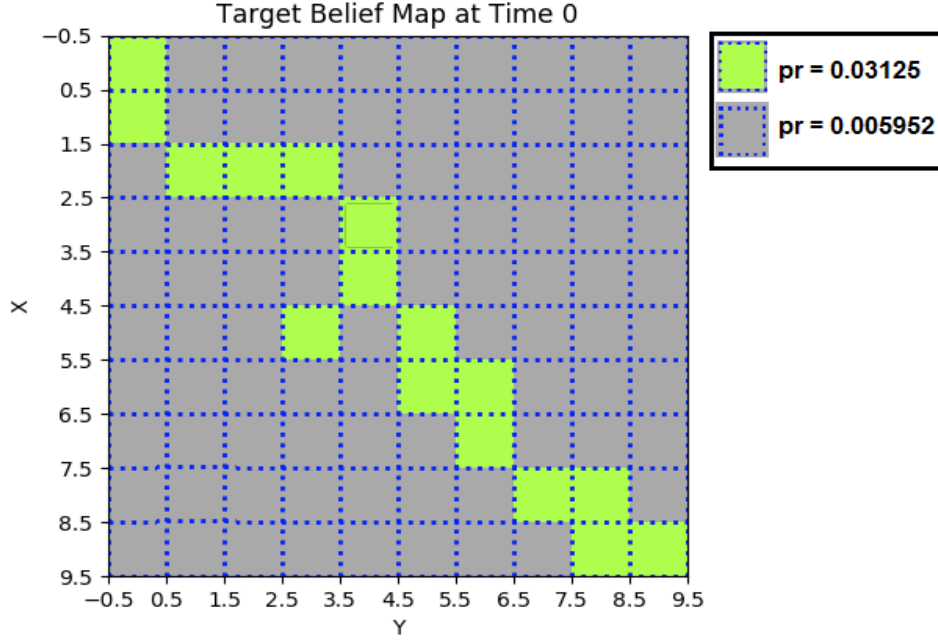


Figure 19 Initialized Belief Map

(at time = 0, based on road and forest regions)

The quadcopter-environment interaction depends on the following: 1- Quadcopter's action in the environment; 2- Quadcopter's observation from the environment; and 3- feedback from the environment, which depends on the objective/goal of the mission.

We assume that the quadcopter can maneuver in the environment by executing one of the following 8 actions: A_0 : Maintain status quo, A_1 : Go to left, A_2 : Go to right, A_3 : Go Forward, A_4 : Go backward, A_5 : Go up, A_6 : Go down, and A_7 : Transmit object location. Thus, the POMDP action-space is defined as: $A = \{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$. In addition, we assume that the velocity and position of the quadcopter is fixed, and when the quadcopter executes one of the $\{A_1, A_2, A_3, A_4\}$ actions, it travels to the adjacent cell located in the direction of the action. The actions A_5 and A_6 change the height of the quadcopter where the location of the quadcopter on the 2-D map, i.e. (X, Y), is fixed. We also assume that the visible area from the quadcopter's camera (mounted on the bottom of the quadcopter) depends on the quadcopter's height. If the quadcopter is located 5 meters above the ground, the view angle of the camera only covers one cell; if the height is 10 meters, the camera covers 4 cells; and finally, if the height is 15 meters, the camera covers a 3x3 grid consisting of 9 cells, where the quadcopter is in the middle of the 3x3 grid. Furthermore, we suppose that the view angle of the quadcopter camera can change to cover different regions when the quadcopter is located 10 meters above the ground. Figure 20 shows cell coverage for different height levels.

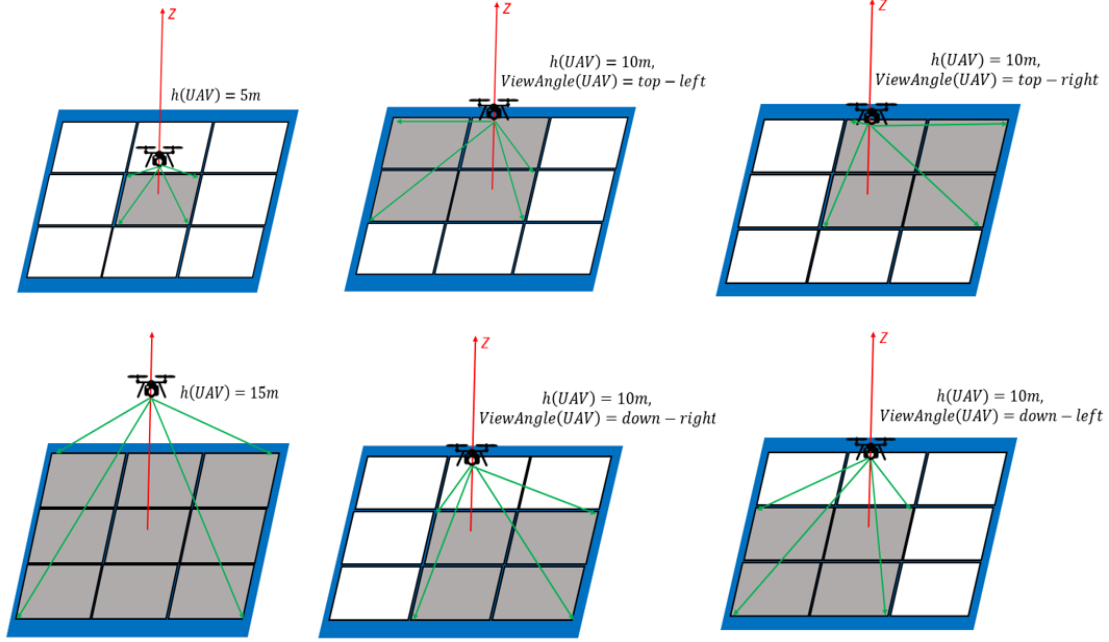


Figure 20 Cell Coverage Based on Quadcopter Height

To speed up the search process and maximize coverage, we implemented a function to control the view angle of the camera when the quadcopter is located 10 meters above the ground. This function evaluates all possible covered cells provided by 4 different viewing angles and selects the view that covers the cells with high belief probability. For instance, if the belief probability of cells/segments obtained from the belief map at time t based on different view angles are {top-left: [0.001, 0.001, 0.001, 0.001], top-right: [0.0005, 0.0005, 0.0001, 0.0005], down-left: [0.21, 0.001, 0.001, 0.001], down-right: [0.0012, 0.14, 0.015, 0.0001]}, the function will tilt the camera towards the down-left corner as the sum of belief probabilities for this region is larger when compared to the other three regions.

For this scenario, we assumed that a target detection and identification model that is trained based on a series of images captured from different locations, regions, and height levels, is integrated with the quadcopter. However, due to partial observability and the fact that the quality of images (i.e., resolution) depends on camera height, the object detection and identification model can make mistakes in detecting objects. We assumed that the resolution of the images improves as the quadcopter descends, so the probability of missed-detection also reduces. In other words, there is a relationship between the height of the camera (or the quadcopter) and the probability of missed-detection of objects from images. Thus, we consider an observation model that accounts for false positives (mistakenly detecting an object from an image when there are no actual objects) and false negatives (not being able to detect objects from images that contain actual objects), where the false positive and negative rates depend on the height of the camera (or quadcopter) (Symington et al., 2010; Waharte & Trigoni, 2010).

- Probability (sensing target at height h – target) or True Positive: $1 - \beta_h$
- Probability (not sensing target at height h – target) or False Negative: β_h
- Probability (not sensing target at height h – no target) or True Negative: $1 - \alpha_h$
- Probability (sensing target at height h – no target) or False Positive: α_h

where $0 \leq \alpha_h, \beta_h \leq 1$ represent the false alarm and missed detection probabilities and vary depending on the height level. Table 1 provides an example of observation model employed for an exemplar scenario.

Table 1. Observation Model Employed for an Exemplar Scenario

Height level (h)	α_h	β_h
h = 5 meters	0.052	0.18
h = 10 meters	0.0629	0.2
h = 15 meters	0.082	0.25

To update the belief map based on quadcopter observations and keep track of the target state probability, we employed a Bayesian approach. The belief probability of a region/cell at time t ($p(b_t)$) is calculated based on the belief probability at time $t-1$ ($p(b_{t-1})$) and probability of observation as shown in eq. 1.

$$b_t(s) = \begin{cases} \frac{(1-\beta_h)b_{t-1}(s)}{(1-\beta_h)b_{t-1}(s) + \alpha_h(1-b_{t-1}(s))} & obs = 1 \\ \frac{\beta_h b_{t-1}(s)}{\beta_h b_{t-1}(s) + (1-\alpha_h)(1-b_{t-1}(s))} & obs = 0 \end{cases} \quad (1)$$

The objective of this scenario i.e., finding a target object (e.g., a person) and transmitting its location to the control unit) is captured in the reward function. At every time-step that the quadcopter makes a decision and takes an action within the environment, it receives feedback from the environment, in addition to an observation, which determines the value of the quadcopter's action relative to the goal to be achieved.

In this scenario, we defined various reward and penalties to address coverage, time, and effort associated with rescuing a missing object in the environment. To address time, we assigned a high reward to cells/segments with high initial belief probability. If the action of the quadcopter results in moving to these regions and covering the associated cells on the grid map, the quadcopter will receive $r_1 = \sum_{s \in R_1 \cap C} b_t(s) \times 10$, where R_1 contains all states with initial high belief probability and C is the area covered by the quadcopter. Otherwise, it will receive a small reward of $r_2 = \sum_{s \in R - R_1 \cap C} b_t(s) \times 1$, where $R - R_1$ contains all states with initial low belief probability. Based on these reward values, the quadcopter initially focuses on surveying the areas (cells/segments) with high belief probabilities, rather than executing a random path. For instance, the immediate reward for transitioning into the segments marked by dashed lines in Figure 20 is: $r = 3 \times 0.03125 \times 10 + 6 \times 0.005952 \times 1 = 0.973212$.

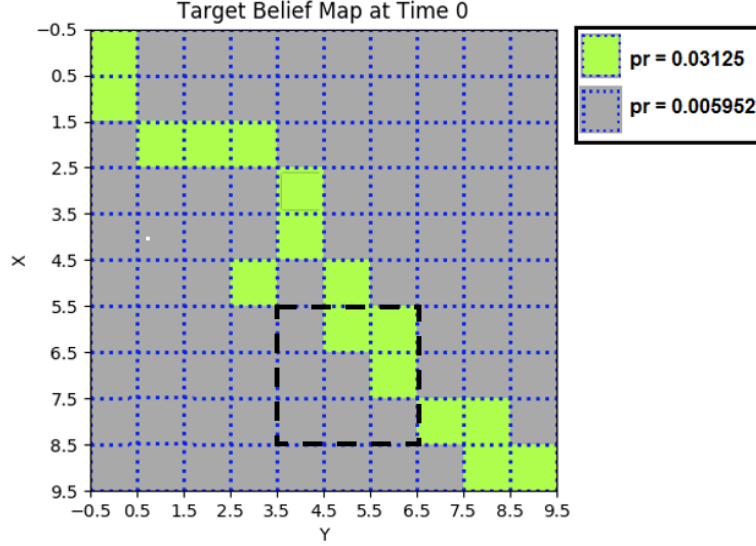


Figure 21 Example Belief Map

(for calculating the immediate reward associated with transitioning into the segments marked by dashed lines)

- To make the quadcopter focus on the regions where target state has high probability after detecting the object in that region, we assigned a penalty to the actions that moves the quadcopter away from that area.

$$penalty(distance) = -b_t(s = s_{obj}) \times (2 \times sum(|[x_{quad}, y_{quad}] - [x_{obj}, y_{obj}]|) + \frac{z_{quad}}{5}) \quad (2)$$

- Finally, to minimize the effort associated with transmitting a signal containing object's location on the grid map to the control unit, we defined a reward term for transmitting the signal when object is found, and a penalty for transmitting the signal with no objects found. Thus, the cost associated with transmitting a signal can be calculated as follows:

$$cost(signal\ transmission) = b_t(s = s_{obj}) \times reward_{transmit} + (1 - b_t(s = s_{obj})) \times penalty_{transmit} \quad (3)$$

Based on this function, the quadcopter transmits the signal to the control unit if the belief probability of the object's state (based on the belief map) is high enough, which results from consecutive successful target detections and identifications in a specific cell on the grid map.

There are various techniques for choosing the next action for the quadcopter based on the belief map. Greedy techniques that focus on coverage can survey the area only based on the belief map, without requiring any reward function. Based on the greedy techniques, the quadcopter's decision in the environment completely relies on evaluating the neighboring regions. For instance, the quadcopter chooses an action that transitions the quadcopter to an area with the largest sum of belief probabilities based on cell coverage.

On the other hand, when the problem is modeled as a POMDP problem, the quadcopter's actions at each time-step are identified based on an optimal policy depending on the belief map and reward function. For this purpose, we employed our online, N-Step Look-Ahead policy estimator algorithm.

The code/implementation for this scenario and model is fully modularized and does not depend on a specific map or observation model.

- *Environment Map Supportability*: Various maps can be plugged into the code to be “gridified” based on the required number of states (e.g., 10x10, 20x20). Belief map initialization is also supported based on a priori.
- *Observation Model Supportability*: Various observation models that are trained based on different maps or scenarios can be plugged into the code to replace our observation model.
- *Scenario and Model Execution*: Users can execute their scenario and model either using the POMDP model or the greedy technique. The N-Step Look-Ahead algorithm employed for POMDP solving is an online algorithm that calculates the optimal policy based on the reward function and belief map; it does not require any offline calculations. Therefore, it can be applied to any belief map created based on our environment model and scenario implementation.

We defined an exemplar scenario based on the environment map provided in Figure 21. In this scenario, we place a target (e.g., a person) in the middle of the map and launch the quadcopter from one of the cells located in the bottom-left corner of the map. The results for both greedy and POMDP (N-Step Look-Ahead with $N = 2$) approaches are provided in the figures below.

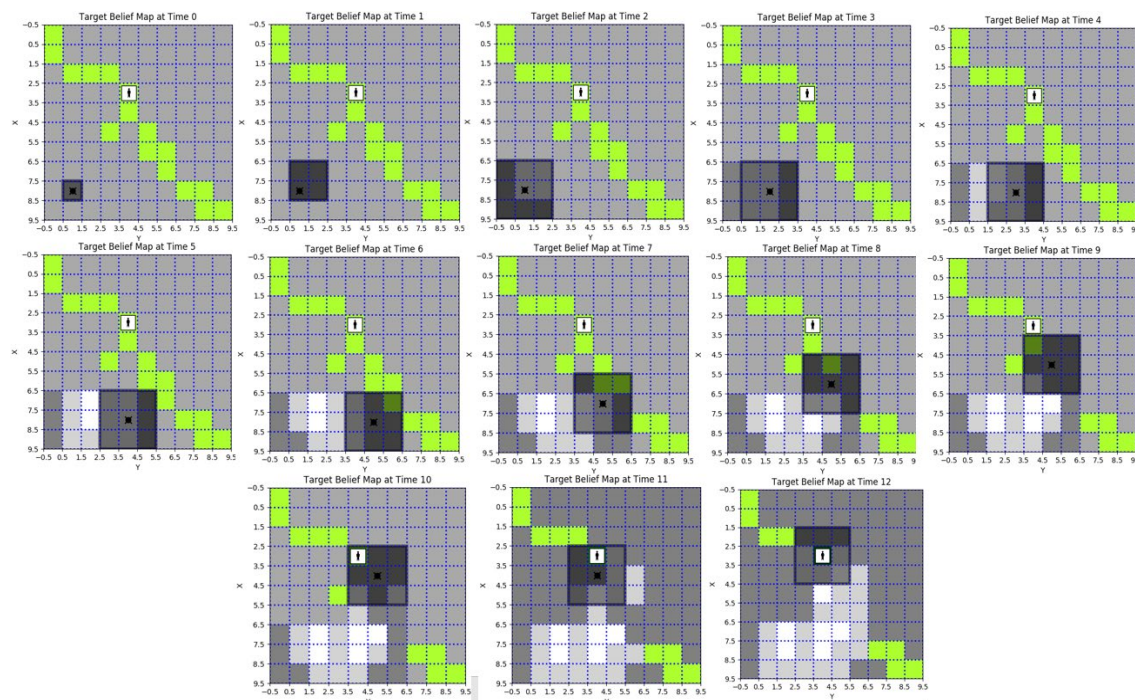


Figure 22 Updates in Belief Map
(based on quadcopter's actions and observations from the environment (Greedy approach))

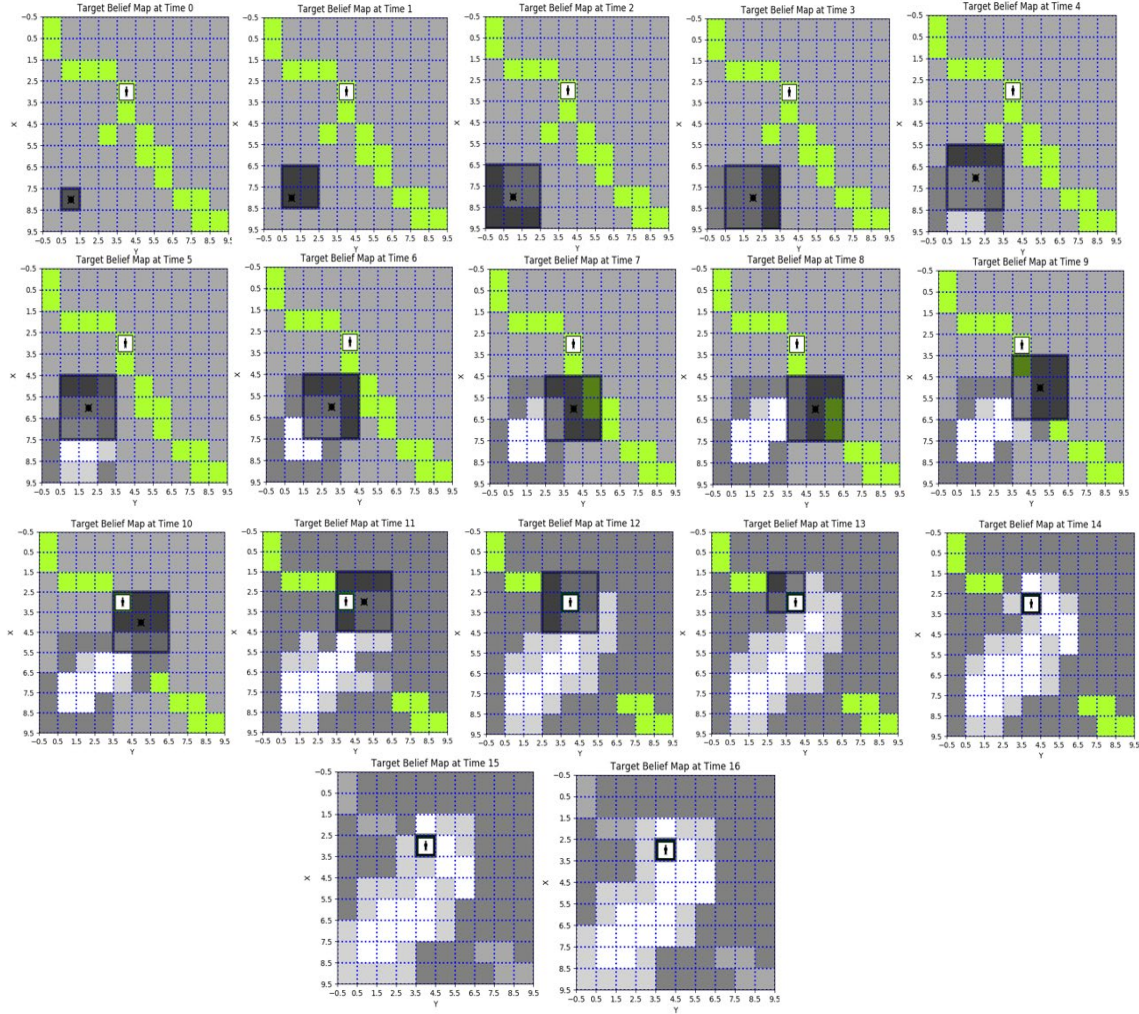


Figure 23 Updates in Belief Map
(based on quadcopter's actions and observations from the environment (POMDP approach))

We defined and prototyped scenarios associated with different missions. We created an exemplar scenario that can be generalized to various missions/goals including surveillance, search and rescue, and package delivery using quadcopters/UAVs.

- Assuming a specific operational environment with known actors (e.g., 1 quadcopter, 1 target object, and 5 non-target objects), states of the environment and/or system, actions of quadcopter, and observations may remain the same, which implies that the same state-space, action-space, and observation-space can be used for different missions/goals.
- Changing the goal/mission (e.g., search and rescue to package delivery) and underlying mission-related constraints (e.g., time constraints) for the same operational environment affects evaluation of taking different actions in the environment (i.e., reward function).
- Changing the goal/mission for the same operational environment implies changing the rewards/penalties associated with taking an action in a specific state.

- Changing the goal/mission for the same operational environment does not affect the outcome of actions in terms of transitioning to a different state or receiving an observation.
- Size of the state-space, transition and emission probabilities are fixed and defining a larger/smaller state-space (and underlying probability distributions) requires recalculating the transition and emission probabilities manually.

Figure 24 illustrates two different scenarios (package delivery and search and rescue) in the same operational environment with pre-defined actors.

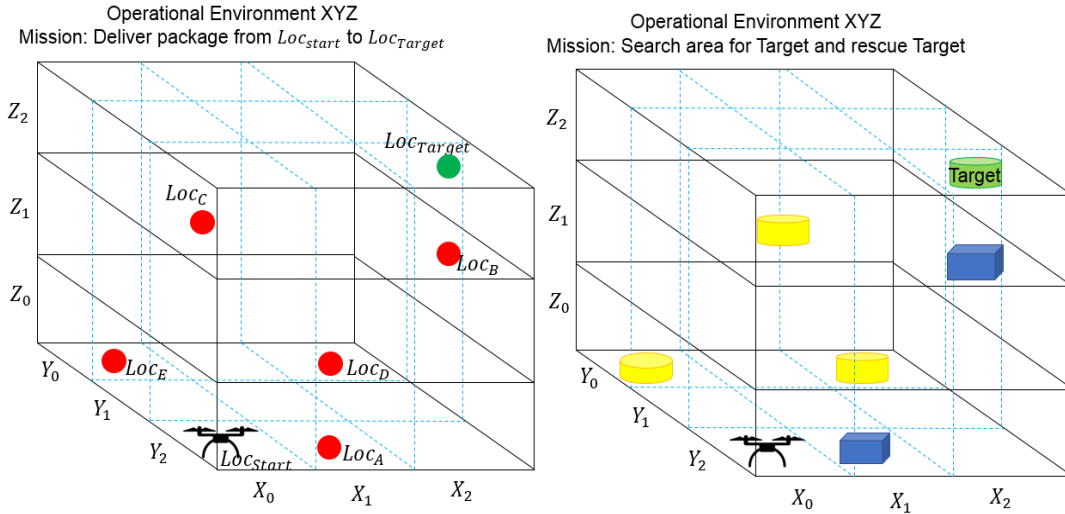


Figure 24 Generalized Illustration of Operational Environment XYZ with Pre-defined Actors

Model for target detection scenario:

- Total 37 states – 1 terminal state and 36 other states that are different combinations of regions (R_i), heights (H_h), and observations (O_j): $S_s = (R_i, H_h, O_j)$
 - Observations at each state are: no object, object detected not identified, object A, object B, ...
 - Total 7 actions – go left, go right, move forward, move backward, go up, go down, and land
 - There is a reward $+R$ associated with detecting the target object, landing next to it, and transitioning to the terminal state, and also a penalty $-P$ associated with landing next to a non-target object.
 - Performing different actions have different costs
 - C_1 for changing region (left, right, forward, and backward)
 - C_2 for changing height
 - C_3 for transitioning to the terminal state
 - $|C_3| \gg |C_2| > |C_1|$ are fixed negative values
 - We will update the fixed costs with a cost function that calculates the cost associated with each action with respect to battery usage

Deterministic Models

For a deterministic physics model to work as intended, a user must define a desired set of waypoints. These waypoints are then converted to a defined path by the algorithm. The path information at each time step is then sent to the “Position Controller.” The position controller is then responsible for generating control signals to stabilize the quadcopter and keep the vehicle on track with respect to X, Y, and Z coordinates. Since quadcopters move in X, Y, Z direction by changing roll, pitch and yaw angles, the position controller is also generating desired roll, pitch, and yaw angles to move the quadcopter along the path. This information is then sent to the “Attitude Controller.” The attitude controller is then responsible to hold those angles as long as necessary to reach desired X,Y, or Z coordinates. The control signals take the following form:

$$\begin{aligned} U_1 &= F_1 + F_2 + F_3 + F_4 \\ U_2 &= F_4 - F_2 \\ U_3 &= F_3 - F_1 \\ U_4 &= T_2 + T_4 - T_1 - T_3 \end{aligned} \quad (4)$$

Where U_1 is, total force required to be generated by all 4 rotors to lift the vehicle up vertically and keep the desired altitude. U_2 is required force to generate desired roll angle. U_3 is required force to generate desired pitch angle. U_4 is required torque to generate desired yaw angle. The controllers are designed based on a Backstepping approach. The detailed steps are discussed in (Bouabdallah, 2005; Suicmez, 2014).

The control signals generated by controllers are then sent to the “Physics Model.” The physics model is the mathematical model of the quadcopter. It is a system of 12 differential equations that mimic the quadcopter’s dynamic behavior. These equations can be written in the form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \\ \dot{x}_9 \\ \dot{x}_{10} \\ \dot{x}_{11} \\ \dot{x}_{12} \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \dot{\theta} \\ \ddot{\theta} \\ \dot{\psi} \\ \ddot{\psi} \\ \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \\ \dot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} x_2 \\ x_4x_6a_1 + b_1U_2 \\ x_4 \\ x_2x_6a_3 + b_2U_3 \\ x_6 \\ x_4x_2a_5 + b_3U_4 \\ x_8 \\ \frac{u_xU_1}{m} - \frac{k_x x_8}{m} \\ x_{10} \\ \frac{u_yU_1}{m} - \frac{k_y x_{10}}{m} \\ x_{12} \\ -g + \frac{\cos(x_1)\cos(x_3)U_1}{m} - \frac{k_z x_{12}}{m} \end{bmatrix} \quad (5)$$

Where,

$$\begin{aligned} a_1 &= (I_y - I_z)/I_x \\ a_3 &= (I_z - I_x)/I_y \end{aligned}$$

$$\begin{aligned}
a5 &= (I_x - I_y)/I_z \\
b1 &= L/I_x \\
b2 &= L/I_y \\
b3 &= 1/I_z \\
u_x &= (\cos(x1)\sin(x3)\cos(x5) + \sin(x1)\sin(x5)) \\
u_y &= (\cos(x1)\sin(x3)\sin(x5) - \sin(x1)\cos(x5))
\end{aligned} \tag{6}$$

L , I_x , I_y , and I_z are defined by the experimenter for the specific quadcopter which will be used. L is the length of the quadcopter arm and I_x , I_y , I_z , are moments of inertia for x , y , and z respectively. Currently, this model is implemented in Java and has been tested many times. This library can be easily linked to the rest of the testbed using socket communication or can be used standalone. For ease of integration this library can be re-implemented in python language in the future.

TESTBED DASHBOARD

Our goal was to develop a general dashboard which is scenario-driven. The design goals for the dashboard included:

- Ability to analyze an arbitrary scenario which is represented by one or more descriptive files
- Ability to utilize and compare automated control algorithms
- Ability to repeat scenario simulations in systematic batch-mode ensembles of experiments

We wanted the dashboard to allow for flexible scenario creation, implementation, and multiple-window display. The dashboard opens a scenario file, which can be changed and updated. The rendering and multi-view display is supported by an open-source python-first gaming engine and a full featured GUI. Specifically, the dashboard graphics were updated for both speed and quality replacing the older Visualization Toolkit (vtk) with the modern Python-based game engine Panda3D. We integrated Panda3D with the frontend (wxPython) graphical user interface and imported higher-fidelity artwork generated by the high-end design program Blender (free open-source). We updated the data repository format from subversion to mercurial to allow for pure implementation and best version control. We developed a scenario construction mode which allowed the capability of specifying scenario actors and control algorithms as well as their interrelationships. The dashboard has an Export to SysML capability which generates the key SysML diagrams representing the current scenario.

Another key design goal of the dashboard was that existing algorithms (e.g., intelligent UAV flight sequencing code) could run as is, without need for translation to Python to incorporate into the dashboard. This capability results from the dashboard being able to communicate with external programs using network socket communications. In the past, we have demonstrated this capability with algorithms running on a different machine than the one running the dashboard.

Critical to analysis of a scenario is the ability to run a sequence of experiments, to test different algorithms, different conditions, different probabilistic outcomes (e.g., various failure modes which occur with varying probabilities), or a combination. The dashboard is therefore designed with a batch capability to run ensembles of such experiments, maintaining detailed logs to facilitate post-processing of the results. To this end, we also:

- Defined a language in which to write scenario and terrain description files
- Created a mechanism through which users may supply external code (e.g., for terrain and model definition) and invoke it using the Python import mechanism. For example, the airport from our earlier C-130 perimeter surveillance scenario is now defined in a text file with the line `Airport: python airport CreateAirport` which imports the module defined in `airport.py` containing the function `CreateAirport`, which returns a set of graphical models representing the airport.

Scenario Creation Tool and Framework

We devised a scenario description framework consistent with our testbed ontology. Scenarios are specified and initialized based on this framework and include the following elements:

- *Operational Environment (Time, Terrain, and Weather)*: The physical area or volume in which the scenario occurs, when the scenarios occur, and what the weather is like at that time (i.e., sunny, windy etc.) For UAV scenarios the terrain is 3D, although the horizontal extent is typically much larger than the vertical. The terrain can be fictitious, in which case it is defined by a set of graphics models, or actual, meaning that the terrain is a region of Earth's surface with an altitude bound. The terrain is simply connected but may have one or more regions of interest (ROIs). For instance, in a search and rescue mission, the staging area and the primary search area may be two ROIs.
- *Actors*: The agents that carry out the scenario. These are typically UAVs. Actors may work in one or more teams; the teams may work together or may be antagonistic (e.g., enemy troops/UAVs).
- *Events*: and/or varying conditions, typically adverse, that may occur. For instance, UAVs may be subject to random wind gusts as well as to occasional engine or other failures.
- *Mission*: The goal or set of goals of the scenario. The mission may be time-independent (e.g., maintain surveillance of a perimeter) or time-dependent (e.g., a series of time-specific waypoints). The mission may have a discrete goal (e.g., blow up a bridge) or a quantitatively defined goal such as to maximize a fitness function.
- *Dashboard display/interface*: The set of display elements and controls to be provided to the user. A plan view of the terrain (mission view) is typical; separate higher-resolution views of ROIs may also be important.

The scenario description framework also includes information associated with scenario display elements, such as:

- Current camera view
- Thumbnails of all cameras
- Thumbnails of actors (e.g., quadcopter, soldier, airplane, and ground vehicle)

When visualizing and testing a specific scenario using the dashboard, the required information for the scenario, such as: mission/goal, operational environment (weather and terrain), environment constraints (e.g., time), software components (e.g., underlying model, functions, algorithms, required libraries, and packages), and scenario-related events (e.g., disruptions, partial observability, and uncertainty) is retrieved from the scenario description and plugged into the dashboard.

If automated control of the agent is desired, then a control algorithm is needed. The algorithm is not part of the scenario. In fact, it will often be the case that different algorithms will be compared to determine the best approach to a given scenario. Human-computer cooperation may be important; in that case operation may be partly automated and partly manual.

RESOURCES AVAILABLE TO SHARE

As a result of this effort, the following resources can be shared with the respective transition sites identified in our initial proposal.

- Testbed Library
 - 3D Models and 3D printing settings
 - Testbed Modeling Library
- Testbed Dashboard
 - Scenario Creation Tool
 - Localization Algorithm

SUMMARY AND WAY AHEAD

Because of COVID-19, we had requested a no-cost extension on the project to integrate, test, and user-harden the testbed software before transition. We were asked to submit a new proposal to perform these activities with the remaining funds in this effort. Upon receiving the approval for this activity, we will finish these integration and test activities and deliver the testbed to two SERC universities.

CONCLUSION

This report has presented our accomplishments in creating a minimum viable MBSE testbed for engineering cyber-physical-human systems. We developed all core components for this testbed, including scenario builder, scenario-driven dashboard, deterministic and probabilistic system modeling techniques, integration components and connectors, and vehicle hardware with onboard instrumentation to support data collection. The testbed is intended to allow SERC researchers working on MBSE for UAVs to provide demonstration on demand to both internal and external customers, to experiment with different modeling approaches, and share results. The testbed employs a low-cost platform and has a near-zero cost of entry and a minimal learning curve. It offers users a starter-kit to get going on modeling and experimentation. The initial configuration could be a software-only setup which costs next to nothing to get started.

APPENDIX A: LIST OF PUBLICATIONS RESULTED

Madni, A.M. (2021). MBSE Testbed for Rapid, Cost-Effective Prototyping and Evaluation of System Modeling Approaches, *Applied Sciences Journal*. Accepted for publication.

Madni, A.M. (2020, November 10-13). MBSE Research Testbed for Rapid and Flexible Experimentation. *NDIA 2020 22nd Annual Systems and Mission Engineering Conference*, Virtual.

Madni, A.M., Sievers, M., Purohit, S., and Madni, C. (2020, October 11-14). Toward a MBSE Research Testbed: Prototype Implementation and Lessons Learned. *IEEE SMC International Conference*, Toronto, Canada.

APPENDIX B: CITED AND RELATED REFERENCES

Bouabdallah, S., & Roland S. (2005) "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor." *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE.

Madni, A.M. (2020). Minimum Viable Model to Demonstrate Value Proposition of Ontologies for Model Based Systems Engineering. *Conference on Systems Engineering Research*, Virtual.

Madni, A. M., Erwin, D., Madni, A., Ordoukhanian, E., Pouya, P., & Purohit, S. (2019). *Next Generation Adaptive Cyber Physical Human Systems*. Stevens Institute of Technology, Systems Engineering Research Center Hoboken, United States.

Madni, A., Erwin, D., Madni, A., Ordoukhanian, E., & Pouya, P. (2018). *Formal Methods in Resilient Systems Design using a Flexible Contract Approach*. SYSTEMS ENGINEERING RESEARCH CENTER HOBOKEN NJ HOBOKEN, United States.

Madni, A. M., Sievers, M., Madni, A., Ordoukhanian, E., & Pouya, P. (2018). Extending Formal Modeling for Resilient Systems Design. *INSIGHT*, 21(3), 34-41.

Ordoukhanian, E., & Madni, A.M. (2020). Ontology-Enabled Hardware-Software Testbed for Engineering Adaptive Systems. *Conference on Systems Engineering Research*, Virtual.

Pouya, P., & Madni, A. M. (2020). Leveraging Probabilistic Modeling and Machine Learning in Engineering Complex Systems and System-of-Systems. In *AIAA Scitech 2020 Forum* (p. 2117).

Pouya, P., & Madni, A. M. (2020). Expandable-Partially Observable Markov Decision-Process Framework for Modeling and Analysis of Autonomous Vehicle Behavior. *IEEE Systems Journal*.

Pouya, P., & Madni, A. M. (2020, October). A Probabilistic Online Policy Estimator for Autonomous Systems Planning and Decision Making. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 2933-2938). IEEE.

Sievers, M. M., Madni, A. M., & Pouya, P. (2019). Assuring spacecraft swarm byzantine resilience. In *AIAA Scitech 2019 Forum* (p. 0224).

Sievers, M., Madni, A. M., Pouya, P., & Minnichelli, R. (2019, October). Trust and Reputation in Multi-Agent Resilient Systems. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)* (pp. 741-747). IEEE.

Suiçmez, E. C. (2014). (2014) Trajectory Tracking Of A Quadrotor Unmanned Aerial Vehicle (UAV) Via Attitude And Position Control. *Diss.* Middle East Technical University.

Symington, A., Waharte, S., Julier, S., & Trigoni, N. (2010, May). Probabilistic target detection by camera-equipped UAVs. In *2010 IEEE International Conference on Robotics and Automation* (pp. 4076-4081). IEEE.

Waharte, S., & Trigoni, N. (2010, September). Supporting search and rescue operations with UAVs. In *2010 International Conference on Emerging Security Technologies* (pp. 142-147). IEEE.