ASCEND
November 15-17, 2021, Las Vegas, Nevada & Virtual
ASCEND 2021

10.2514/6.2021-4028

A Resource Allocation Algorithm for a Space Habitat ECLSS

Matthew Rines*, Michael Balchanos†, and Dimitri Mavris‡
Georgia Institute of Technology, Atlanta, Georgia, 30332

Jubilee Prasad Rao§, and Jesse Williams¶
Global Technology Connection Inc., Atlanta, GA, 30339

As space habitats increase in complexity and distance from Earth, new methods for maintaining crew safety and system performance are needed to handle unexpected disturbances. This work develops a methodology for implementing a resource allocation algorithm to manage the subsystems of an environmental control and life support system. An algorithm is developed to control the oxygen generation assembly (OGA) while inducing various disturbances to the urine processor assembly (UPA). Habitat system resilience is assessed and evaluated with and without the use of the algorithm.

To test the effectiveness of this methodology, it is implemented in a limited use-case for 100 varying types of degradations in the UPA. The degradations are based on available data from the International Space Station. The optimization process is conducted to find the optimal controls to the OGA. The performance of the habitat with the optimal controls is then compared to a baseline, logic-based controller. To automate subsystem controls to maximize overall resilience of life support systems when a fault is identified, three supervised machine learning algorithms (Gaussian Process, Random Forest, and XG Boost) are trained to the optimized data and compared to each other for accuracy.

Although the UPA degradations had little impact on the overall crew safety, it is found that the optimal OGA controls had a median increase in resilience 14 times greater than the median decrease in resilience seen with the logic-based controller. Additional analysis and comparison of the improvement in resilience based on the failure scenario is conducted. Out of the three machine learning algorithm, XG Boost is identified as the algorithm that performed the best in approximating the optimizer under these circumstances with an R^2 value of 0.84.

I. Nomenclature

$CCAA$	=	common cabin air assembly
CRS	=	carbon dioxide reduction system
$ECLSS$	=	environmental control and life support system
EVA	=	extravehicular activity
GPR	=	Gaussian process regressor
IMV	=	intermodule ventilation
ISS	=	international space station
IVA	=	intravehicular activity
ML	=	machine learning
OGA	=	oxygen generation assembly
PCA	=	pressure control assembly
PCM	=	pressurized core module
PLM	=	pressurized logistics module
$PPRV$	=	positive pressure relief valve
RBF	=	radial basis function

*Graduate Research Assistant, School of Aerospace Engineering, AIAA Student Member
†Research Engineer II, School of Aerospace Engineering, AIAA Member
‡Regents Professor, School of Aerospace Engineering, AIAA Fellow
§Engineer, GTC
¶Chief Technology Officer, GTC

UPA = urine processor assembly
$VCCR$ = variable configuration CO_2 removal
WPA = water processor assembly
WRS = water recovery system

II. Introduction

Space exploration and habitats hosting crews for extended time periods are characterized by increased complexity driven by strict thresholds on environmental conditions, system of system (SoS) complexities, sub-system interactions and human-in-the-loop related challenges. These factors contribute to high levels of operational uncertainty resulting in adverse system states and requiring safety redundancies to avoid mission failures especially during subsystem faults. Successful operation of spacecraft and long-term crewed habitats depends on the optimal operation of subsystems, availability and utilization of resources, as well as on the crew activities. It is important to have system control decisions onsite and operate independently from ground stations located far away on Earth [1].

As the environmental control and life support system is a large, critical, and complex resource generation (water, oxygen, food) and distribution (waste, CO_2, N_2) system, health management becomes crucial to mission success [2]. When subsystem faults are experienced, it may become possible to optimally allocate resources and control the remaining subsystems [3]. The objective of this effort is to explore and test the feasibility of a method for analysis and mitigation of adverse conditions experienced during space habitat operations using machine learning techniques. The goal is to develop an algorithm that autonomously allocates key resources and modifies control settings of subsystems to maintain and/or improve the resilience of a given habitat during such adversities. Such solutions are especially useful to mitigate the effects of never before experienced failure modes.

In a space setting, multiple systems operate in a coordinated fashion to maintain life supporting resources like drinking water, air quality, humidity levels, temperature etc. Despite having redundant systems, it is very critical to optimally operate them to minimize levels of risk.

III. Proposed Approach

A. Goals and Requirements of Autonomous Resource Allocation

The algorithm being developed seeks to improve the resilience of such life support systems in space during system or component failures. The algorithm needs to mitigate fault related risks or degradations and improve operational effectiveness during single or compound faults. Upon detection of a failure, the framework will extract relevant system and state features and alter timelines of operation and/or controls of different systems to maintain safe environment and resource levels. In doing so, the risk of mission failure is reduced by maintaining the environmental variables like availability of drinking water, partial pressures of oxygen, carbon dioxide etc. The algorithm will be trained on a large number of fault scenario simulations to be ready to react optimally to fault scenarios never observed before. This can autonomously mitigate the cascading effects of failures by altering resource management and utilization and hardware control strategies.

The algorithm has been developed around a space habitat architecture and simulation environment called HabNet [9]. HabNet simulates the various life support systems and subsystems required for survival in a space habitat, their failures and interactions. The algorithm is built and trained using this simulation module for different failure scenarios. The algorithm learns and improves as more failure scenarios are simulated. The performance of the algorithm and the learned resilient strategies depends also on the fidelity of the simulation environment. Once developed, the algorithm can learn from any simulation module and extends beyond HabNet module or just for space habitats. It could be trained with the latest simulation tools that NASA develops or uses in the future. The resilient resource management algorithm built and tested with HabNet could also be implemented to learn in real time from the ISS as well as future lunar or Mars habitats instead of simulation modules.

When new life support system technologies are being considered for integration, the developed algorithm along with the habitat simulation platform can identify potential failures that increase risk for mission failures. Even though individual failures may not be mission or life critical, the interactive effects and cascading effects over time could pose greater risks to mission success. This method will search a design space by inducing many random potential failures in different subsystems and try to learn to mitigate risks involved.

B. Methodology for Autonomous Resource Allocation

The proposed methodology shown in Figure 1 has two main steps. The first is the initial setup and training of the resource allocation algorithm shown in the top row of the figure. The second step is the deployment of the algorithm onto a real system and its continued learning using data from real operation.

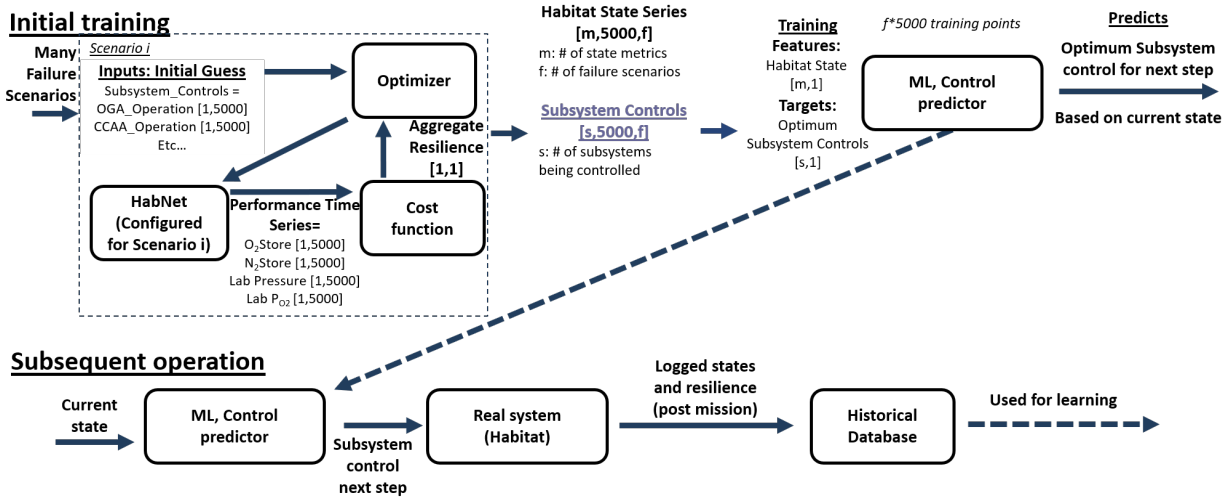

Fig. 1 Overview of the proposed resource allocation methodology.

The methodology to develop the resource allocation algorithm begins by optimizing the subsystem controls during a defined mission with a subsystem failure scenario. The habitat and mission configuration as well as the effects of the failure scenario need to be configured in the simulation environment. The optimizer varies the subsystem controls for the duration of the defined mission and calls the simulation environment to evaluate the performance with those subsystem settings. The subsystem controls are the amount each subsystem in the habitat should be operating at for every time step. These values can range from 0 to 100%. The optimization begins by initializing the subsystem controls with reasonable values.

The optimizer is seeking to maximize the resilience of the habitat and will change the subsystem controls values until it finds an optimum. The mission performance data needs to be aggregated in order to inform the optimizer whether a certain set of controls is better or worse than another set of controls. To do this, a resilience metric is calculated for different system performance attributes and used to establish the cost function [4]. Four primary principles that a system needs to be able to do with regards to disturbances are monitor, respond, anticipate, and learn [5]. Tran et al. define resilience as the ability of a system to absorb, adapt, and recover to system degradations [4]. They also developed a formulation to quantify resilience. This is shown in Equation 1, with the symbols defined in Table 1.

$$R = \begin{cases} \sigma\rho \times [\delta + \xi + (1 - \tau^{(\rho-\delta)})] & \text{if} \rho >= \delta \\ \sigma\rho \times (\delta + \xi) & \text{otherwise} \end{cases} \tag{1}$$

Table 1 Breakdown of Tran et al. resilience formulation[4].

Resilience Factor	Description	Equation
σ	Overall capability	$\sigma = \frac{\sum_{t=t_0}^{t_{final}} y(t)\Delta t}{y_{desired} \times (t_{final} - t_0)}$
ρ	Restorative capacity	$\rho = \frac{y_{recovered}}{y_{desired}}$
δ	Absorptive capacity	$\delta = \frac{y_{min}}{y_{desired}}$
ξ	Volatility	$\xi = \frac{1}{1+exp[-0.25(SNR_{db}-15)]}$
τ	Recovery time	$(\tau = \frac{d^*}{n})$

For this project, it was decided to begin with this formulation and then simplify and adapt it as necessary. In terms of simplification, the volatility term of the original metric was eliminated as this was not deemed highly important for the objectives of the problem. Additionally, the original metric was formulated to reward any increase in performance, but an alternate version has been added that rewards targeting a specific performance based on mean squared error shown in Table 2. So for instance, if performance is measured by the amount of O_2 in the habitat, then it makes sense that targeting an optimal level could make the habitat more resilient than maximizing it.

Table 2 Comparison of equations for minimum performance and target performance.

Description	Desired Minimum	Desired Target
Overall capability	$\sigma = \frac{\sum_{t=t_0}^{t_{final}} y(t)\Delta t}{y_{desired} \times (t_{final} - t_0)}$	$\sigma = 1 - \frac{\sum_{t=t_0}^{t_{final}} (y_{desired} - y(t))^2}{y_{desired}^2 \times (t_{final} - t_0)}$
Restorative capacity	$\rho = \frac{y_{recovered}}{y_{desired}}$	$\rho = 1 - (\frac{y_{desired} - y_{recovered}}{y_{desired}})^2$

Additionally, some performance metrics associated with a space habitat do not easily fit into this resilience formulation. For example, in the case of crucial resource store levels such as water and oxygen, it is not obvious what a desired performance level would be. Also, for stores that do only deplete over time and do not have a means of refilling, the concepts of recovery time and the restorative capacity also lose meaning. For metrics such as these, it has been decided to create a more applicable resilience metric based on mean squared error, shown in Equation 2. Since the more filled the storage tanks the better, this formulation penalizes any capacity less than maximum.

$$R_{store} = 1 - \frac{1}{t_{final} - t_0} \sum_i (\frac{y_{initial} - y_i}{y_{initial}})^2 \tag{2}$$

Habitat O_2 partial pressure is just one example of a performance metric important to the life support system. Since there are many such metrics, there needs to be a way to aggregate them all into a single value that can then be used to calculate an overall resilience value. Although there are many ways to aggregate objective values, the task of determining the best way of doing this has been left for future work. In this project, it is simply the average of all values, as shown in Equation 3. So for example, resilience could be calculated for the amount of O_2, N_2, and CO_2 in the habitat, and then these three resilience values could be averaged to give an aggregate resilience value.

$$R_{agg} = \frac{1}{n} \sum_i^n R_i \tag{3}$$

The outputs of the optimization step are the set of optimal subsystem controls for all failure scenarios as well as mission performance data for each failure scenario. This data is then post-processed and formatted to train a supervised machine learning algorithm to predict the optimal subsystem controls based on the habitat state. The machine learning algorithm is crucial because it avoids the need to run computationally expensive optimization anytime a decision needs to be made for how to operate a subsystem. The data is organized into pairs of features and targets, where the features are the relevant habitat state data separated for each time step across all failure scenarios, and the targets are the corresponding optimal subsystem controls for those time steps.

A supervised learning regression algorithm can then train on this data and results in an algorithm that can use current habitat state data to determine how to control subsystems and allocate resources to them for the next time step in order to maximize the resilience of the habitat.

Ideally this algorithm would then get used on a real habitat as shown in the bottom row during subsequent operation of the figure. A state of the real habitat would get input to the algorithm and subsystem controls would be output and sent to the ALS system of the habitat to execute subsystem operation. The performance of the habitat would also be recorded and used to allow the control predictor algorithm to learn over time. This is a necessary feature both to tune the algorithm to any discrepancies between the habitat model and the real system when it comes online as well as allowing the algorithm to make better decisions in light of failure scenarios and habitat state combinations that it did not train on.

IV. Setup of Methodology Demonstration

The setup and implementation of the methodology up to the deployment on a real system step will be demonstrated. To do this, the various components will be defined in this section.

A. Identification of the Baseline Habitat ALS Mission Scenario

The modeling and simulation environment selected for use in this project is HabNet. HabNet was developed by Sydney Do as part of his dissertation at MIT in 2015[6]. Table 3 shows the review that was completed in comparing different modeling and simulation environments. HabNet was selected because it models the advanced life support system of a surface habitat using relevant technology descriptions based on the ISS, is quick to run, and is well-documented and flexible enough that modifications could be made to tailor HabNet for the purposes of this project.

Table 3 Evaluation of the existing M&S environments to the established requirements. ✓ indicates that the environment nearly or completely satisfies the requirement. ~ means that the requirement is not entirely neglected but is not nearly satisfied. × translates to the requirement being neglected by the environment.

Requirement	BioSim	V-Hab	HabNet	ELISSA
ECLSS Modeling	✓	✓	✓	✓
Crew Modeling	✓	~	✓	✓
Definable Habitat Architecture	~	✓	✓	~
Definable Mission Architecture	~	✓	✓	✓
Disturbance Modeling	✓	~	~	~
Subsystem Operational Control	✓	×	×	×
High Speed	~	×	✓	×

The HabNet environment also comes with a baseline scenario already set up. The habitat architecture seen in Figure 2, is based on the architecture of NASA's Habitat Demonstration Unit [7] and the Scenario 12.1 Lunar Outpost concept that was part of NASA's Constellation Program [8]. The baseline scenario comes with initial stores of key resources such as food, drinking water, oxygen, nitrogen, among others. It also has key technologies loaded in such as an oxygen generation assembly, carbon reduction system, common cabin air assembly for dehumidification, pressure control assemblies for oxygen and nitrogen injection into the habitat, among others. Table 4 highlights some of the key statistics for the baseline scenario.

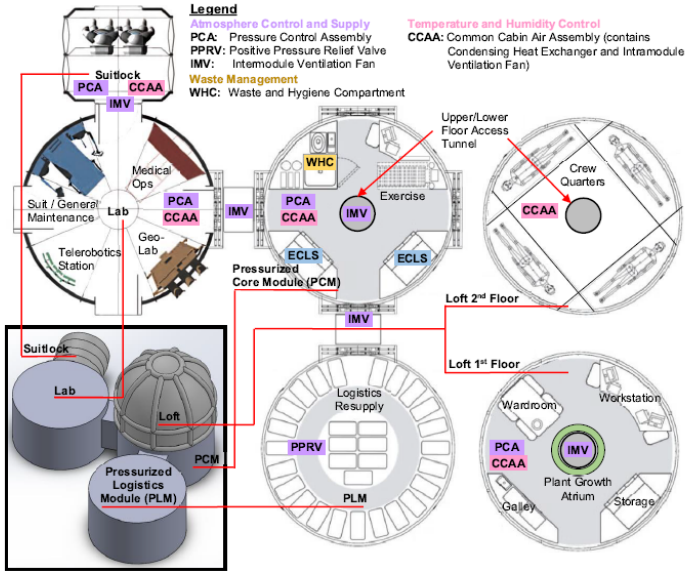

Fig. 2 HabNet Baseline Habitat Architecture.[9]

The baseline mission architecture assumes a crew of 4 on a 5000 hour stay without any resupplies, in-situ resource utilization, or crop growth.

Table 4 Summary of HabNet baseline habitat and mission architectures.

Habitat Configuration	Mission Configuration
Volume	Crew Size
$8300 ft^3$	4 - 2 females, 2 males
Type/Location	Duration
Surface habitat on Mars	9 months
Layout	Crew Schedule
See Figure 2	8 hours of sleep/day
Resource Capacity	2 hours of exercise/day (unless EVA)
O_2: $670kg$	EVAs occur in pairs for 8 hours
H_2O: $2960L$	
H_2: $5kg$	
N_2: $38kg$	
Subsystems	
Oxygen Generation Assembly (OGA)	
Carbon Dioxide Reduction System (CRS)	
Variable Configuration CO_2 Removal (VCCR)	
Pressure Control Assembly (PCA)	
Positive Pressure Relief Valve (PPRV)	
Urine Processor Assembly (UPA)	
Water Processor Assembly (WPA)	
Common Cabin Air Assembly (CCAA)	
Intermodule Ventilation Fans (IMV)	

Crewmembers are assigned a schedule consisting of extravehicular activities (EVAs), intravehicular activities (IVAs), exercise, and sleep. Their actions relate back to the amount of resources they are consuming and producing at a given time. For example, exercise requires more oxygen to be consumed and more carbon dioxide to be produced in the habitat than sleep. The crew schedule by default is not deterministic when a HabNet simulation is run. The exact time of EVAs is randomized within a certain tolerance (5 EVAs are to be conducted each week) and the two astronauts selected to perform a given EVA is also randomized. The randomness introduced in the crew schedule introduces some slight variability in the results of a given mission. Figure 3 shows the results of 1000 baseline simulations that were run to measure baseline resource allocation. The oxygen partial pressure of the lab module is plotted for the length of a single mission. The darker green represents a higher frequency of simulations. As can be seen, there is little variability in the amount of oxygen in the lab module from one simulation to the next. The sudden increase in oxygen around timesteps 1800 and 3500 is due to the operation of the oxygen generation assembly. Also plotted is the minimum tolerable oxygen pressure before the crew would start experiencing hypoxia. The baseline scenario maintains oxygen levels well above this limit. The baseline scenario assumes that no systems in the habitat experience any failures, but if faults were to occur, this could likely decrease the performance of the habitat to maintain tolerable conditions. Table 5 shows the six ways that conditions can deteriorate to a point at which crew members can die. If a crewmember dies, then the mission is considered a failure and the simulation ends.

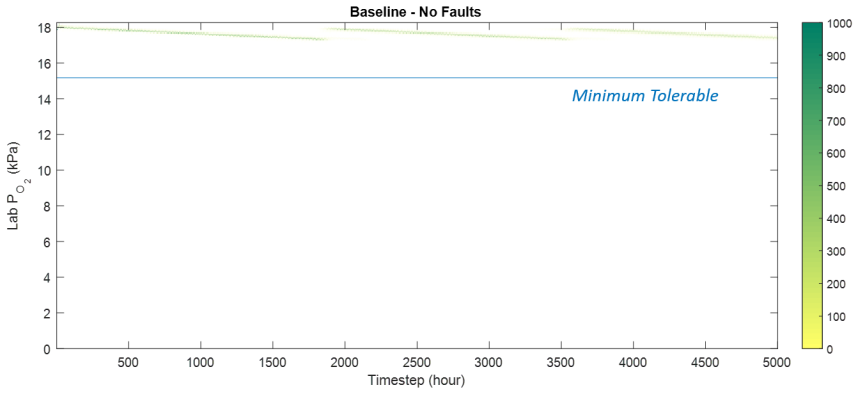

Fig. 3 O_2 Partial Pressure in the Lab Module for 1000 simulations.

Table 5 Summary of possible conditions in HabNet that result in mission failure

Failure Conditions	Model Implementation
Crew Starvation	Crew caloric consumption requirement is greater than calories available within food store
Crew Dehydration	Crew water requirement is greater than potable water available within potable water store
Crew Hypoxia	Partial pressure of O_2 within crew environment is less than 15.168kPa
Crew Hyperoxia	Molar fraction of O_2 within crew environment is greater than 60% (Corresponds to the hypoxic limit for the 55kPa atmosphere assumed)
Crew CO_2 Poisoning	Partial pressure of CO_2 within crew environment is greater than 0.482kPa (0.07psi)
Cabin Under-pressure	Total cabin pressure is less than 20.7kPa (3psi)

B. HabNet Expansion

The baseline HabNet environment has been updated to improve fault creation, detection, and propagation. Figure 4 shows how the additions fit into the existing HabNet environment and how they add functionality.

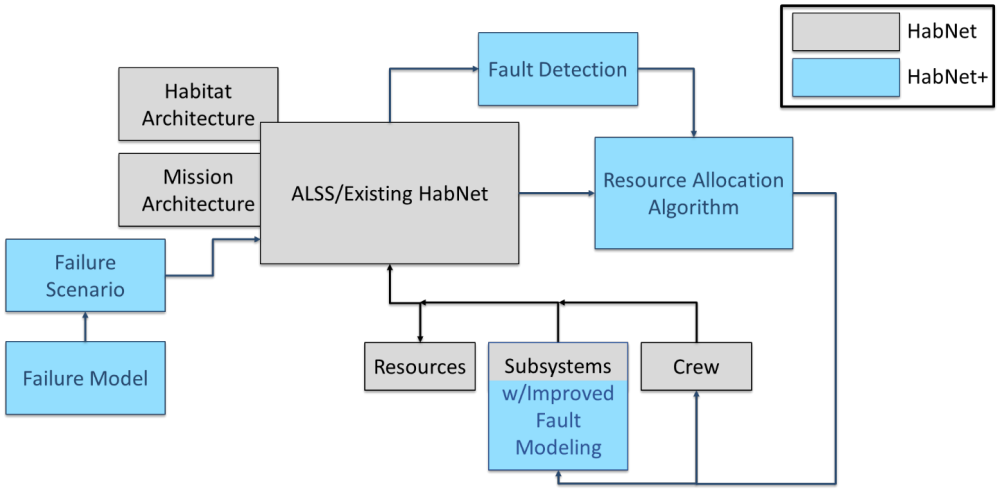

Fig. 4 Diagram showing how the existing HabNet environment was modified.

The types of faults that could be created was limited by how subsystems were modeled to be able to fail; they could

only be turned off. In order to add flexibility to the types of faults that could be modeled, the subsystem models needed to be changed. Subsystem faults are now able to be enacted at a percent degradation instead of either working perfectly or not working at all. For example, the pressure control assembly, which is responsible for adding oxygen to the habitat from the storage tank when necessary, could experience a 47% degradation, which would reduce the maximum flowrate of oxygen into the habitat by 47% of its nominal value. The ability to simulate variable degradations was added to all nine of the major subsystems in HabNet.

A module for fault detection was also added along with a basic algorithm for detecting faults, which acts as a placeholder for more sophisticated algorithms in future work. The current implementation of fault detection, shown in Figure 5, is set to recognize a fault after two operations of the degraded subsystem. To do this, every subsystem was modified with a property to keep track of how many times it has run since it degraded as well as a property to signal whether its fault has been detected yet or not.

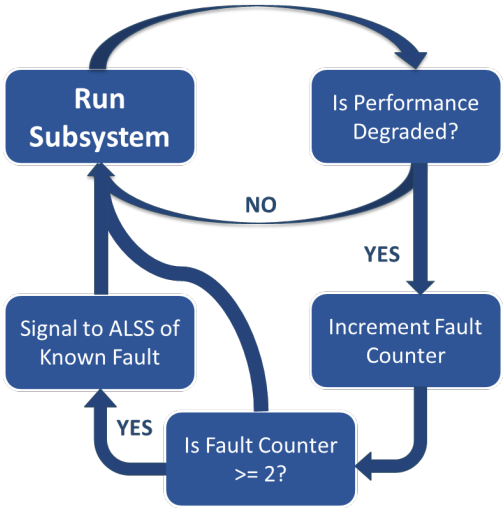

Fig. 5 Diagram of the fault detection module.

The main HabNet environment also needed to be modified to incorporate and make better use of the new features added to subsystem files and the fault detection module. HabNet now allows for an associated impact or degradation parameter to be given along with the subsystem that will experience it. HabNet then passes the degradation parameter onto the corresponding subsystem to take effect. HabNet also now calls the fault detection module, which determines whether any existing faults would be known to the health monitoring system and crew of the habitat. This interaction is shown in Figure 4.

Additionally, HabNet was further modified to be able to input the duration of a fault. Previously, HabNet would only trigger at what timestep a fault would start, but the fault would have to continue until the end of the simulation. The addition of a fault duration allows the environment to simulate a degraded component being replaced after a reasonable amount of time. HabNet is now able to take in any number of subsystem failures as well as their corresponding start times and durations, and dynamically implement those into the simulation by propagating that information to the subsystems when they are performing their functionality.

C. Defining Failure Scenarios

Data from the International Space Station's urine processor assembly (UPA) was used to develop realistic failure scenarios that could be provided as an input to a HabNet simulation [10]. The UPA data can be seen in Figure 6 and includes information on the dates of component failures and when the component was able to be replaced in the UPA.

The time from one failure to the next allowed a model to be made for time between failures, which could be used for coming up with failure start times for a given failure scenario input to HabNet. Using the time between component failure and component replacement similarly allowed the development of a model to approximate failure duration, which could also be used for the failure scenario input to HabNet. For both of these cases, Weibull distributions were proposed to approximate the ISS data. A Weibull probability plot was generated, where the data points, shown as + in Figure 10, should follow the line if a Weibull distribution is a good representation. Additionally, a Lilliefors test was conducted

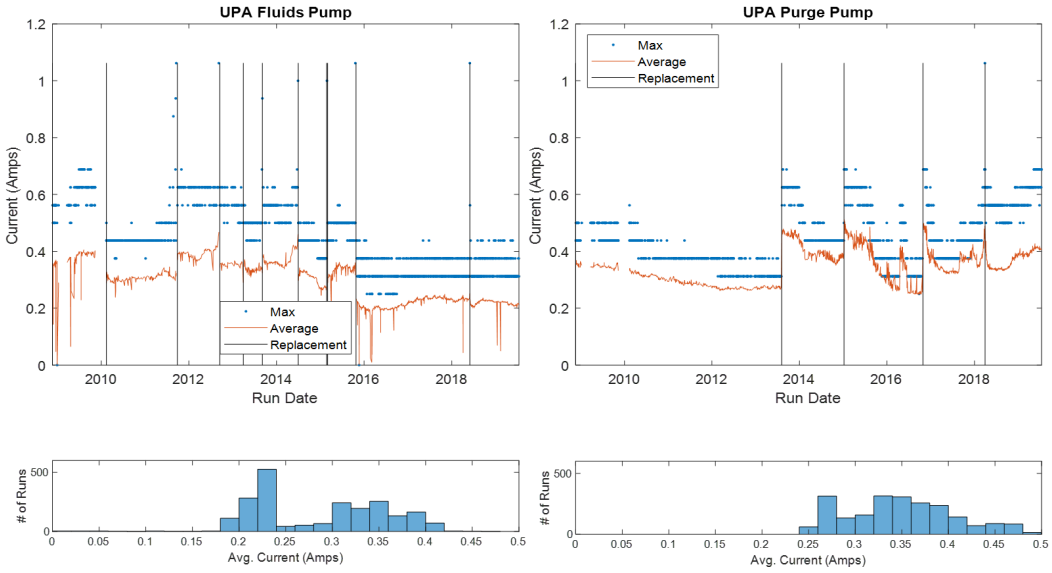

Fig. 6 Provided data from the ISS's UPA.[10]

with the hypothesis that the data came from a Weibull distribution. The test failed to reject the hypothesis with a 5% significance level, so Weibull distributions were used.

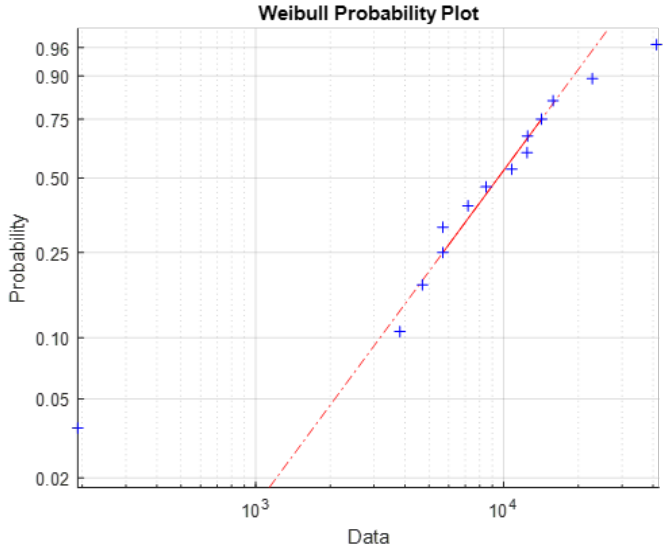

Fig. 7 Comparing the distribution of the data provided to a Weibull distribution.

The fitted Weibull distributions for time between failures and failure duration can be seen in Figures 8 and 9. With the fitted distributions, a realistic failure model for HabNet's UPA subsystem could now be implemented. Furthermore, an additional module was developed to randomly generate an entire failure scenario to be given as an input to HabNet. Of course, with additional data corresponding to the other subsystems, failure distributions could be tailored to each of the subsystems, so that a unique failure model could feed into each subsystem.

D. Defining the Scope of the Optimization Block

Given the limited computational resources and time for this project, the implementation of the methodology had to be narrowed to controlling one subsystem's operation and in light of just one type of subsystem failure. These

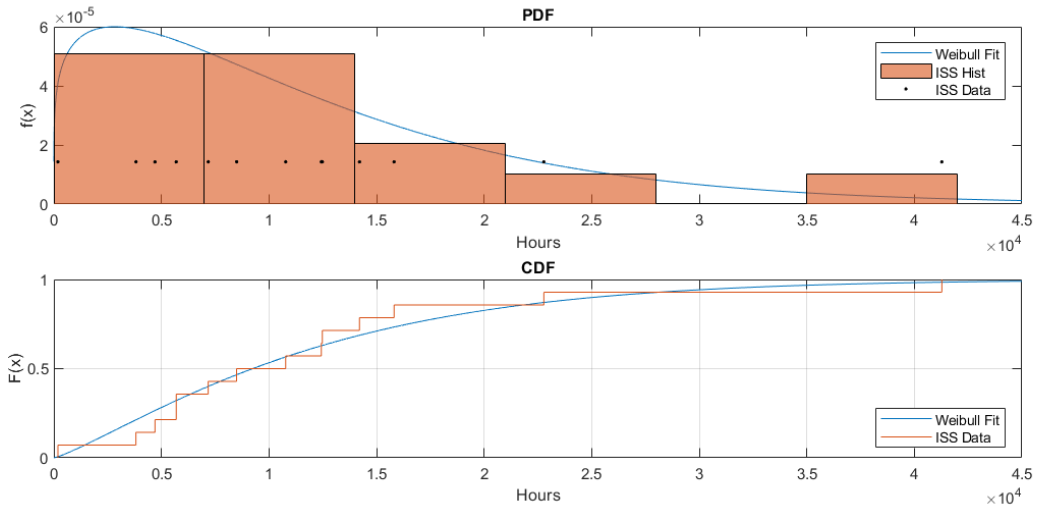

Fig. 8 A PDF (top) and CDF (bottom) for a Weibull distribution describing time between failures.

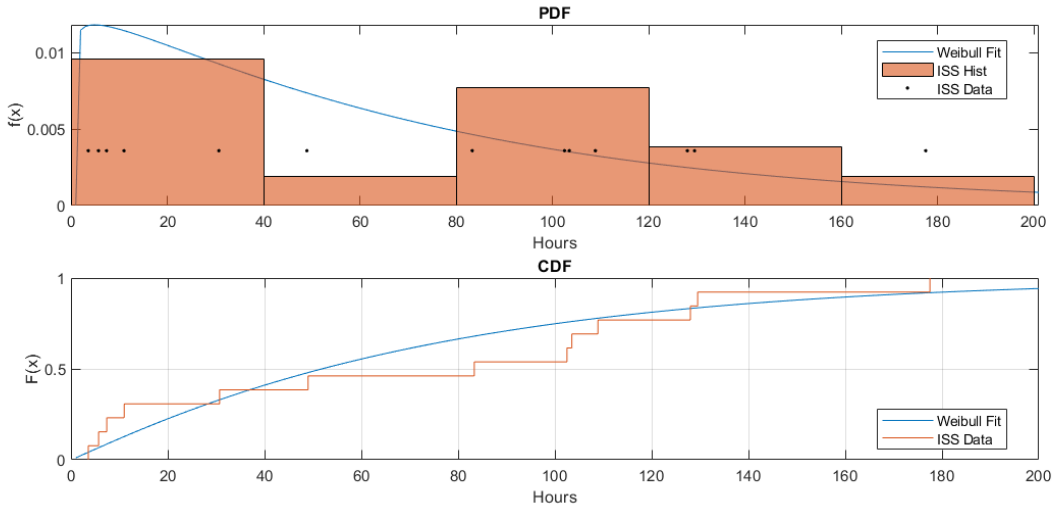

Fig. 9 A PDF (top) and CDF (bottom) for a Weibull distribution describing duration of failures.

subsystems are chosen to be the UPA for generating failure scenarios and the OGA for the optimizer to control.

Figure 10 shows the process of down-selecting subsystem failures to establish the make up of the failure scenarios used in the optimization process. All HabNet subsystems that have failures enabled are initially listed. It was decided to focus on subsystem failures that would directly affect mission failures related to hypoxia or dehydration. The reason for selecting these two mission failures from the possibilities shown in Table 5 was due to their tight coupling. Since the OGA requires water to create oxygen these two resources are easily exchanged. The second column in Figure 10 shows the five subsystems that were left after taking this criteria into account. Furthermore, since a reduced water supply naturally leads to dehydration or possibly hypoxia via inoperability of the OGA, it was decided to stick with systems that directly affected the amount of water on the habitat. From the three subsystems left in the third column, the UPA was ultimately selected because the failure model created was based on data relevant to the UPA and failure scenarios using just the UPA would therefore be more realistic.

With the selection of the UPA as the subsystem to fail, a set of failure scenarios needed to be created that would vary the start time, duration, and impact of the failures. The failure model was used to generate 100 failure scenarios limited to just one failure per simulation. Figure 11 shows statistics on the failure scenarios that were generated. The start times

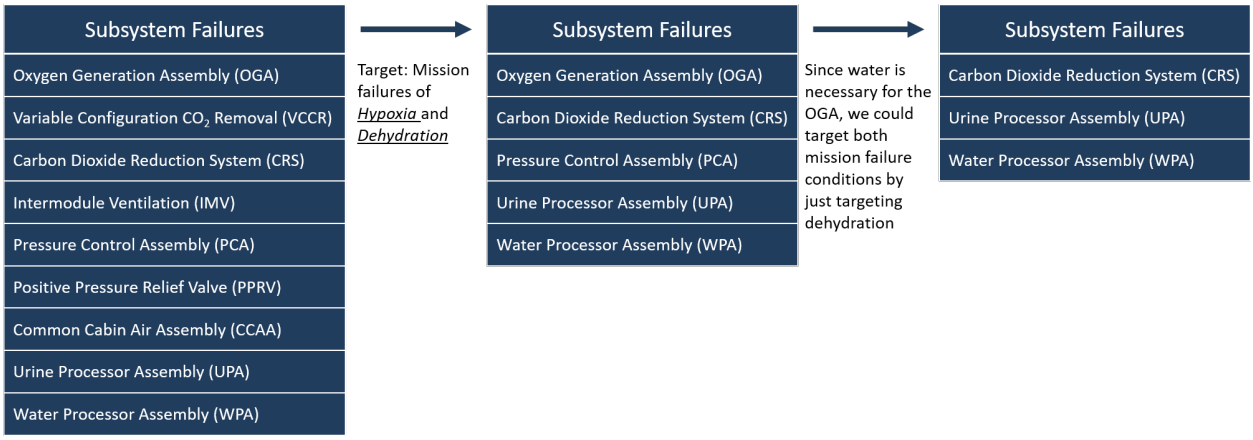

Fig. 10 Process of down selecting subsystems to include in failure scenarios.

and durations are based on Weibull distributions fitted to provided UPA data, whereas the degradation level is simply based on a uniform distribution. Figure 12 shows the same failure scenarios more visually. Each horizontal red line is a single failure showing when it occurs during the mission as well as the impact associated with it. Overall, this goes to show the sampling of UPA failures provided in the 100 scenarios.

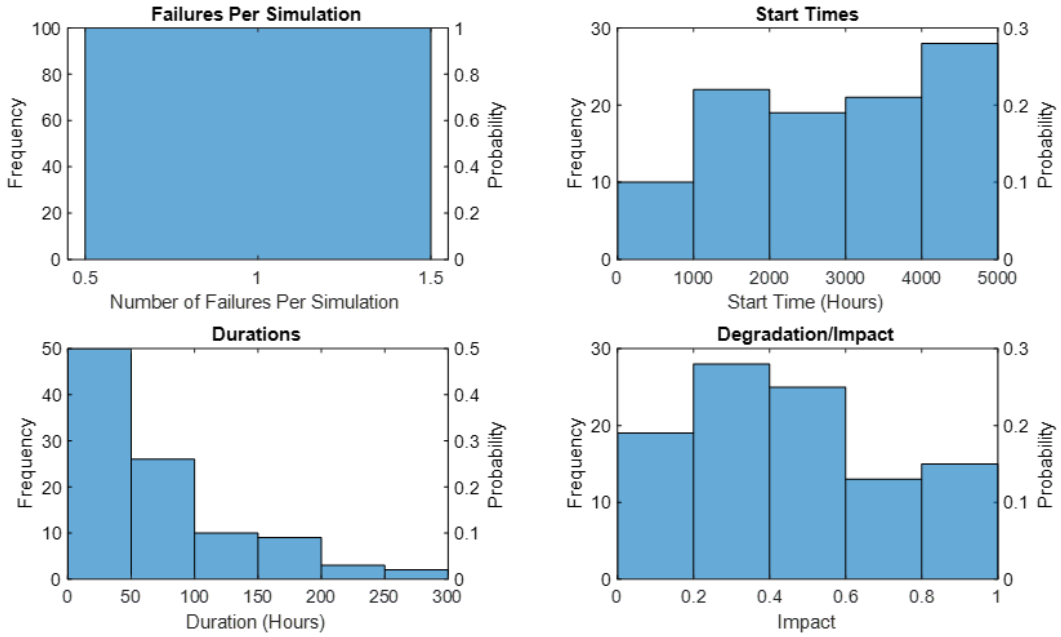

Fig. 11 Statistics of 100 UPA failure scenarios.

With the failure scenarios created, simulations can be run to model their effect and establish a baseline that the performance with optimized controls in the demonstration can be compared against. Figure 13 shows the change in resilience from the nominal mission value to the UPA failure scenario value. The average decrease was 9.47×10^{-6} and the median was 6.48×10^{-8}. The reason the decreases in resilience are so small is because the severity of the degradations had little effect on the overall functioning of the habitat and health of the crew.

The failure scenarios with the greatest decreases in resilience are highlighted in Figure 14. It can be seen that these four scenarios were all of relatively long duration, high impact levels, and not very near the end of the mission.

The next component to decide for implementation is the subsystem that should be controlled in order to maximize

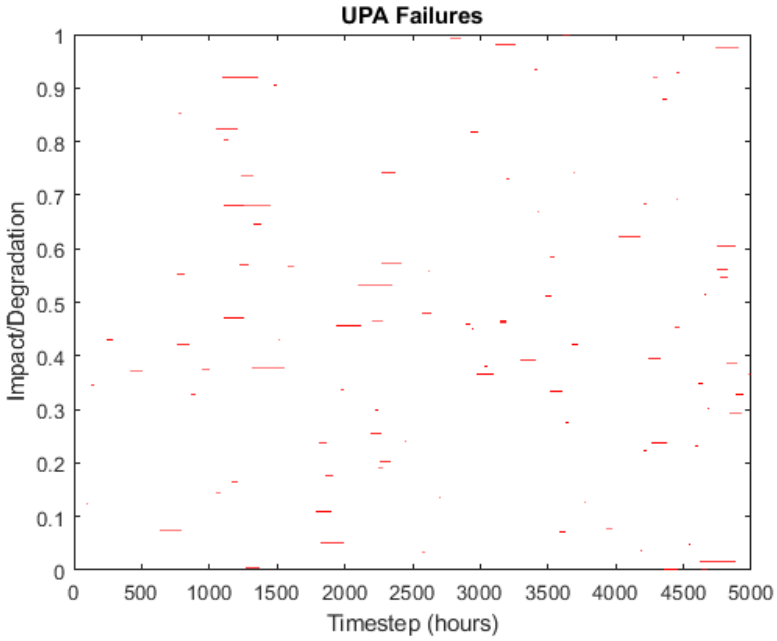

Fig. 12 Visualization of 100 UPA failure scenarios.

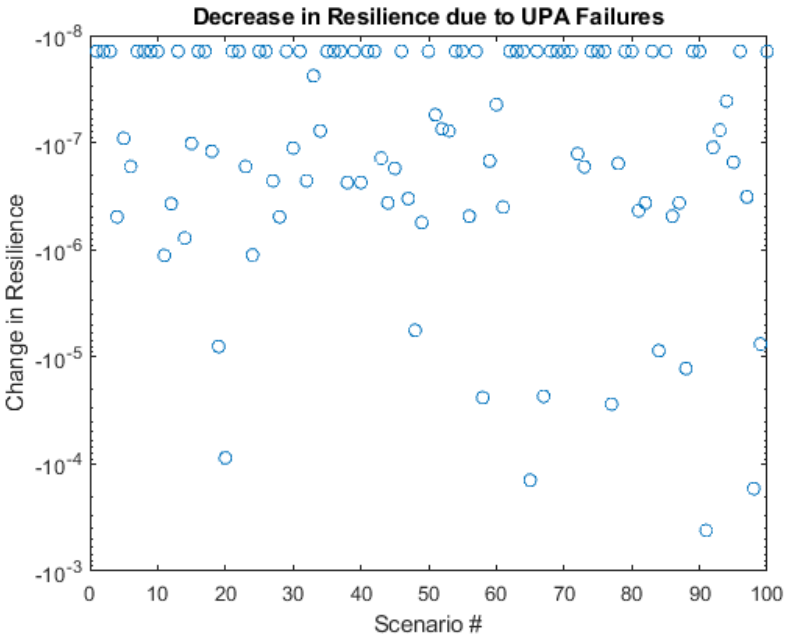

Fig. 13 Decrease in the habitat aggregate resilience due to UPA failure scenarios.

resilience of the habitat. Although controlling many or all of the subsystems would provide the best results, one subsystem needed to be selected because of the computationally expensive nature of optimization. Additionally, to speed up the optimization process, the controls would only be optimized during subsystem failures. Although this is the most critical time for advanced subsystem controls, it is likely that more benefit to the space habitat would be seen if the resource allocation algorithm is applied during all times.

The subsystem to control needed to have some ability to operate in a more beneficial way than the default operating

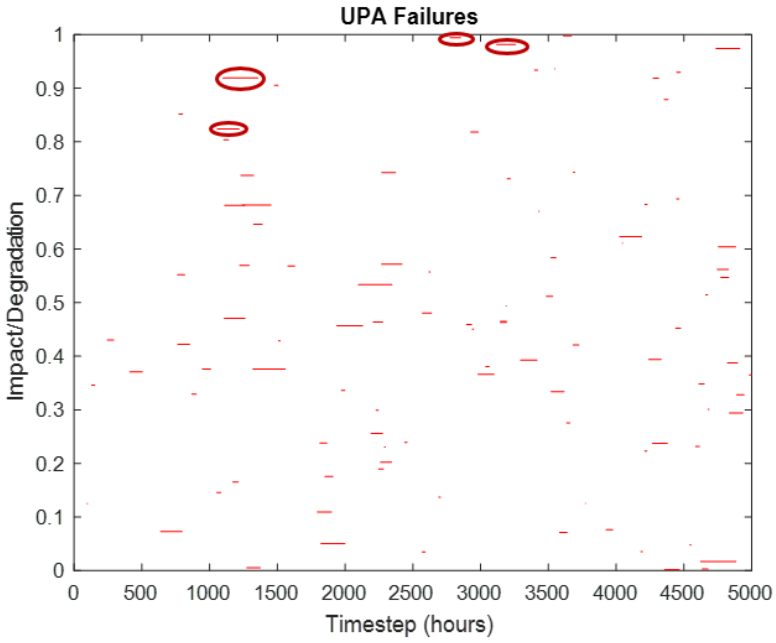

Fig. 14 UPA failures that resulted in the greatest decreases to habitat resilience.

procedure in light of the UPA failures. The OGA was selected because it affected the water supply and the environmental oxygen supply. Controlling the UPA would not be optimal as this was the subsystem that would be failing, and the WPA operates in tandem with the UPA so that also would not be highly beneficial. Controlling whether the crew conducted EVAs or remained inside the habitat to conserve oxygen would have likely been very beneficial, but since this was a binary decision, it was decided to go with the OGA, which had a range of options that could be optimized.

The optimization algorithm also needed to be selected. Since HabNet is in Matlab, it was decided to use a built-in Matlab optimizer. Furthermore, because the OGA does not operate in a continuous scale, it was necessary to stick with a zero-order method. Matlab's patternsearch method was chosen to do the optimization of resource allocation [11].

The last component to be decided before performing the optimization is which performance metrics should be plugged into the aggregate resilience equation (Equation 3). Since the failure scenarios are affecting the water supply and resource allocation is optimizing the oxygen production, it makes sense to use all the performance metrics directly related to these two aspects. This leads to seven metrics: two for the water and oxygen store levels and five for the oxygen levels in the habitat compartments. The form of the aggregate resilience equation used is shown in Equation 4. Since, the interaction effects on resilience capacities are not currently known, this form is just illustrative and enables an objective function for now. As is standard for optimization algorithms, the aggregate resilience equation is negated when converted into the cost function to be minimized for the pattern search algorithm.

$$R_{agg} = \frac{1}{7}\sum(R_{O_2Store} + R_{H_2OStore} + R_{Lab,O_2} + R_{Loft,O_2} + R_{PLM,O_2} + R_{PCM,O_2} + R_{Suitlock,O_2}) \tag{4}$$

After the optimization, machine learning (ML) algorithms need to be developed to automatically control the environmental systems to maximize the resilience of the space habitat. The ML controller emulates the system optimizer discussed above. There are two distinct benefits of using the ML controller:

- The ML controller computes the most resilient control settings very quickly (orders of magnitude faster than the optimizer).
- The ML controller can interpolate within the parameter space previously simulated and optimized, and thus map a continuum of points. The ML controller can also extrapolate outside of the optimized parameter space, but these estimates are expected to be far less accurate.

Regression ML algorithms were considered to emulate the optimizer, and the Python language was used as it arguably has the best supported data science packages. A Gaussian Process Regressor, Random Forest Regressor, and XG Boost are tested and compared using the Scikit-Learn (sklearn) package. Sklearn has a broad catalog of ML models

(but no deep-learning models) [12].

V. Results

A. Optimizing the OGA Performance

With all of the steps of the optimization part of the resource allocation algorithm completed, the cases can be run and the results investigated. Figure 15 shows the improvement in resilience of the optimized cases over the resilience of the degraded cases operating in default mode. 96 of the 100 scenarios showed at least some improvement. On average, the resilience increased by 3.35×10^{-4}, which is 35 times the average decrease seen. The median increase was 1.48×10^{-6}, which is 14 times the median decrease seen.

Fig. 15 Resilience improvement for failure scenarios due to optimizing OGA operation.

To further investigate how failure scenarios affect the resilience of the habitat, the resilience improvement was mapped against the failure impact, duration, and time remaining to look for any trends. This can be seen in Figure 16. There does not seem to be a strong correlation between the failure impact and resilience improvement. The mapping against failure duration seems to indicate that failures with low duration also had small improvements in resilience. The plot with time remaining looks like scenarios with a lot of time remaining, failures that happen near the beginning of the mission, correspond to large resilience improvements.

These trends were further investigated in Figures 17 & 18, which support the conclusion that missions with UPA failures that occur early in the mission and for longer durations are most readily improved by optimizing the OGA controls.

Figure 19 looks at the sub-categories of the aggregate resilience metric and how they improved due to optimization. The resilience values associated with the amount of oxygen in the habitat compartments look to make up the majority of the improvement seen in the aggregate resilience value. The resilience value associated with the O_2 store does not change from the simulations without optimizing the OGA to the optimized simulations. The H_2O store shows very small improvement that can not be seen at the scale of the other components. This indicates that the main driver in resilience improvement for this case study came from the optimized OGA controls being able to target optimal O_2 levels in the habitat more effectively than the default control mechanism. To better understand how the optimized simulations differed from the non-optimized cases, actual performance data can be compared.

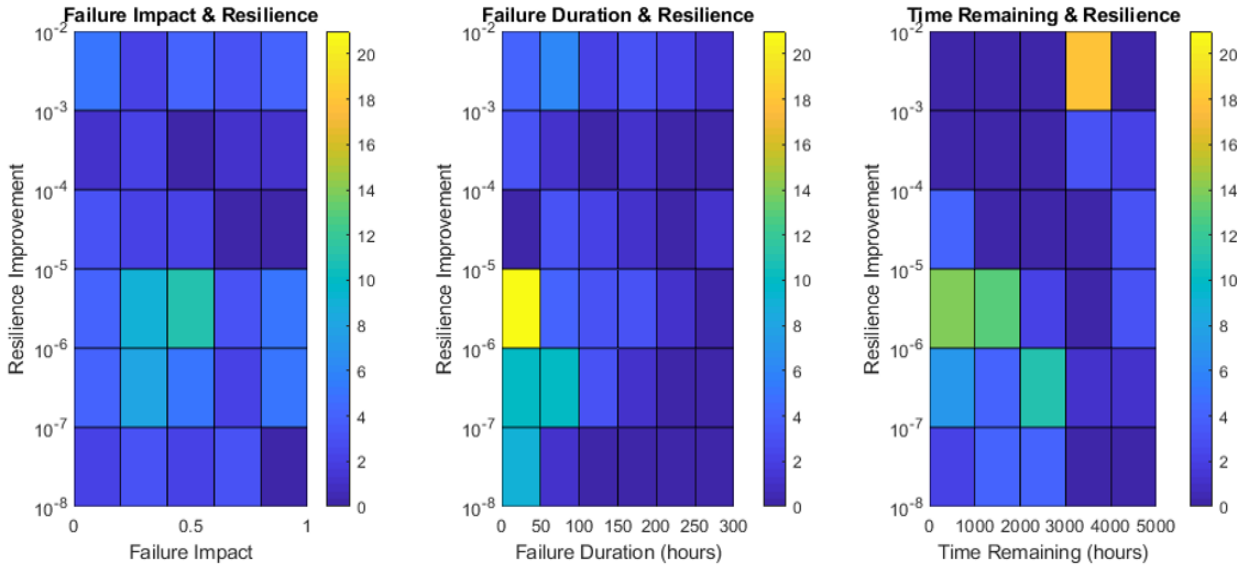

Fig. 16 Breakdown of resilience improvement and failure scenario characteristics. The colorbar corresponds to the number of scenarios that fall within a given bin.

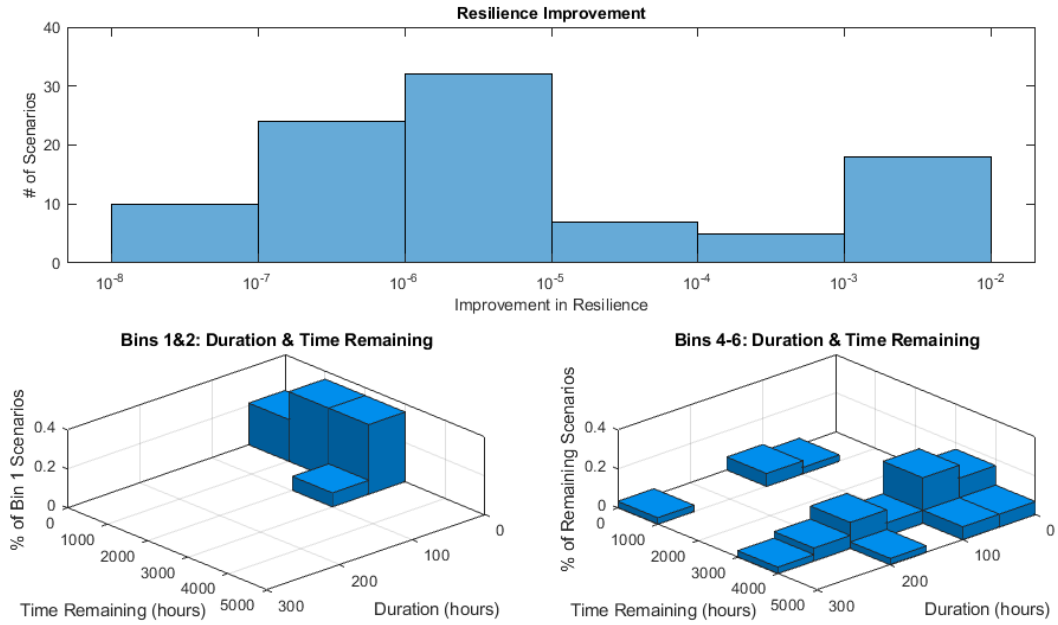

Fig. 17 Breakdown of resilience improvement based on failure duration and time remaining in the mission after the failure.

Figure 20 shows how optimization changed the amount of potable water and oxygen remaining at the end of mission as well as how the total amount of water processed by the UPA and oxygen produced by the OGA changed. On average, the optimized simulations actually have a little less water at the end of the mission, however most scenarios show little change. The amount of water processed by the UPA during the entirety of the mission was not affected by the optimized

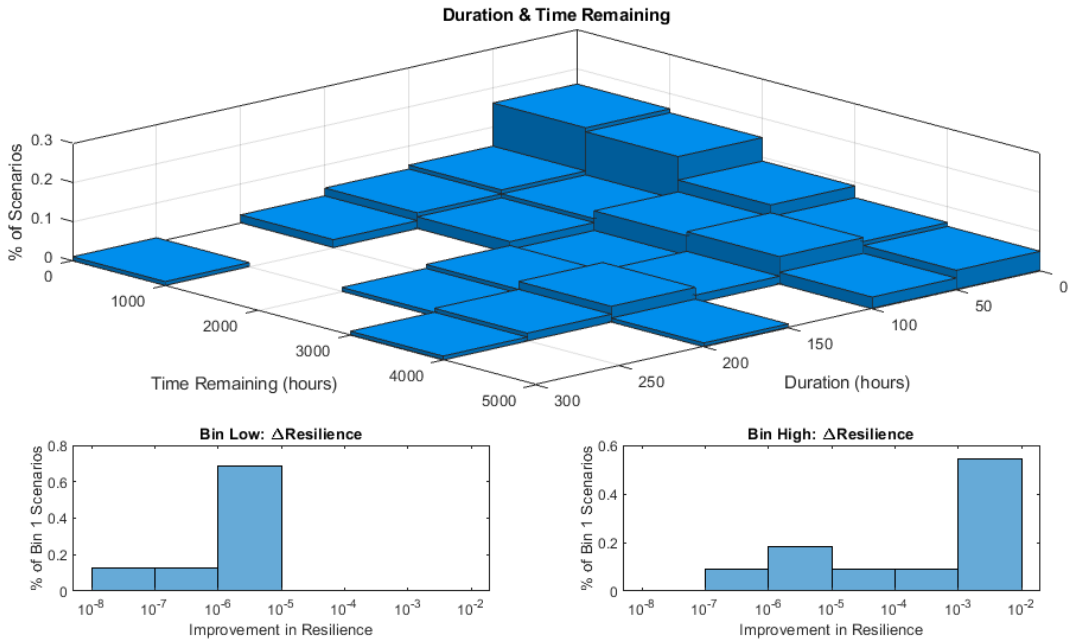

Fig. 18 Breakdown of failure duration and time remaining in the mission after the failure based on resilience improvement.

controls. Since the UPA typically does not constantly run at maximum capacity during a nominal mission, it most likely was able to process additional water after the failure in it ended. The oxygen store levels at the end of the missions are unaffected by optimizing the OGA controls, which explains why the resilience value associated with the O_2 store was also unchanged. On average, half a mole of additional oxygen is produced by the OGA over the course of optimized missions. Despite having a failure in the water recycling system, the optimized OGA tended to run more, consuming more water in order to create more oxygen. This seems like the opposite of how the OGA should have performed in the case of water scarcity, however in reality, the water store was never scarce.

As was seen with the incredibly small magnitude decreases in resilience from the nominal mission to the failure scenarios, the UPA failures based on the ISS UPA data are very small and inconsequential to the functioning of the habitat given the large initial water capacity at the start of the mission. Given that the decrease in the water supply was small and non-threatening to the preservation of the crew and that the aggregate resilience equation was formulated such that five of the seven equally weighted resilience components pertained to the amount of oxygen in the habitat, it makes sense that the OGA controls were altered in a way that primarily focuses on that. This highlights the importance of testing the algorithm on a more critical failure and on developing a better way to aggregate resilience across various performance metrics based on their impact to the mission and health of the crew.

With the completion of the optimization, a machine learning algorithm would next be trained using this data, which could then be plugged into a habitat and act as a decision making component as outlined in Figure 1.

B. System Control Automation with Machine Learning

The training dataset included 493,097 observations where there was not system fault and 6,903 observations with system faults and optimized controls. Features used to train the models were O_2 store level, H_2O store level, lab O_2 level, loft O_2 level, PCM O_2 level, PLM O_2 level, and suitlock O_2 level; and the target variable was the control setting for the next timestep, i.e., the next set point. Thus the training features and target sets were arrays of shape 500,000x8 and 500,000x1 respectively. This dataset was split into a train and test datasets of size 400,000 and 100,000 respectively.

Initially, sklearn's Gaussian Process Regressor (GPR) was investigated because a Gaussian regression estimates the model's level of uncertainly in addition to estimating the target variable. The GPR model was tuned with train and

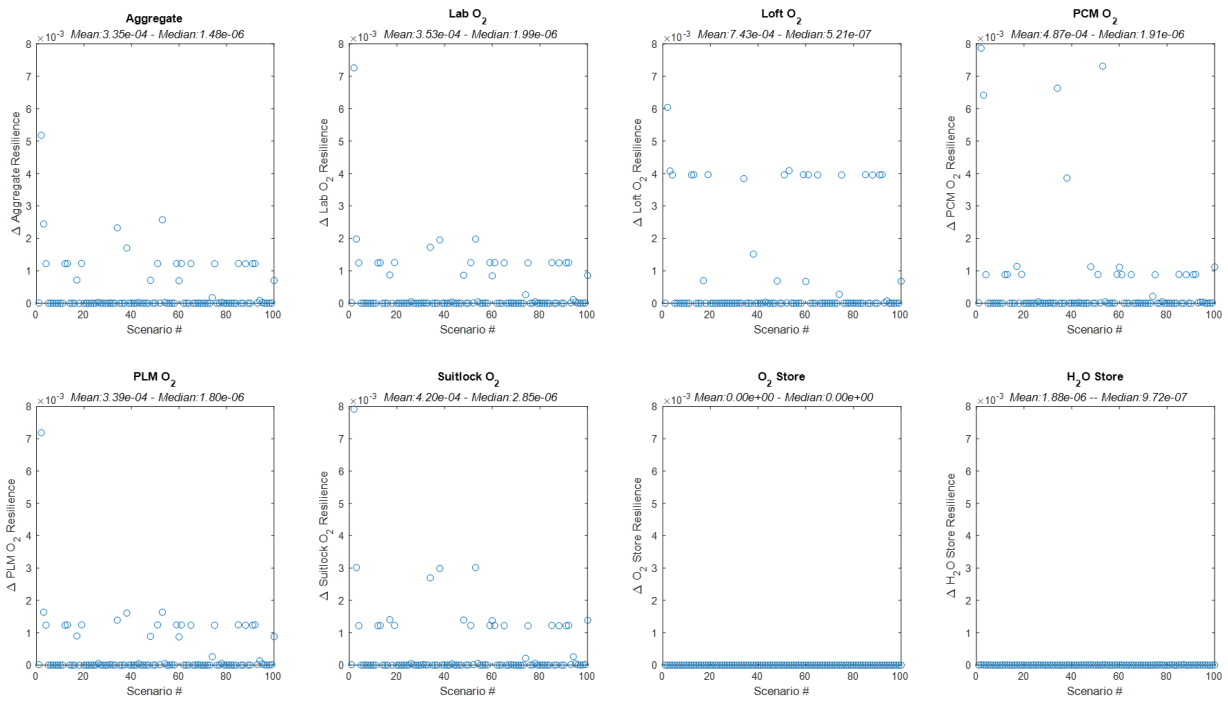

Fig. 19 Breakdown of resilience improvement into resilience sub-categories.

test datasets. During the tuning process various kernels were investigated, and it was determined that the Radial Basis Function (RBF) yielded the best results. Once the model was tuned, it was evaluated on the subset of test data that contained faults, with a subsample size of 1381. The coefficient of determination, R^2 parameter, was used to determine the model's accuracy. The tuned GPR model had an R^2 of 0.51 and 0.12 for the test set and test subset (faults only) respectively.

Two other models were evaluated in an effort to achieve a better predictive performance: sklearn's Random Forest Regressor and XG Boost, a gradient boosting algorithm. Both of these algorithms are derivatives of a decision tree ensemble. The Random Forest model creates many parallel decision trees, or estimators, with weights given to each estimator. XG Boost creates decision trees evaluated in series. Each sequential estimator is trained on the residual data. The random forest regression was tuned to use 1000 estimators, which produced an R^2 accuracy of 0.97 and 0.77 for the test set and test subset (faults only) respectively; a marked improvement over the GPR. The XG Boost was also tuned with 1000 estimators and had a respective R^2 accuracy of 0.98 and 0.84; an improvement over the random forest regression, and the best performing model trained.

C. Extreme failure case study

In an attempt to see a more drastic change in resilience both before and after optimization, an extreme failure scenario in the UPA was created. The failure is a complete degradation 2000 hours in duration, happening at the beginning of the mission, and is mapped against the previous 100 failure scenarios in Figure 22. This failure scenario results in an aggregate resilience decrease from the nominal mission of 6.1×10^{-3}, about 1000 times larger than the average decrease in the 100 previous scenarios. After optimizing, the resilience improved by 5.2×10^{-3}, or about 85% of the decrease. The water store at the end of the optimized mission had 0.86 liters less than the non-optimized mission. The optimized OGA controls had one altered timestep from the default controls, operating at a 50% higher level as shown in Figure 23.

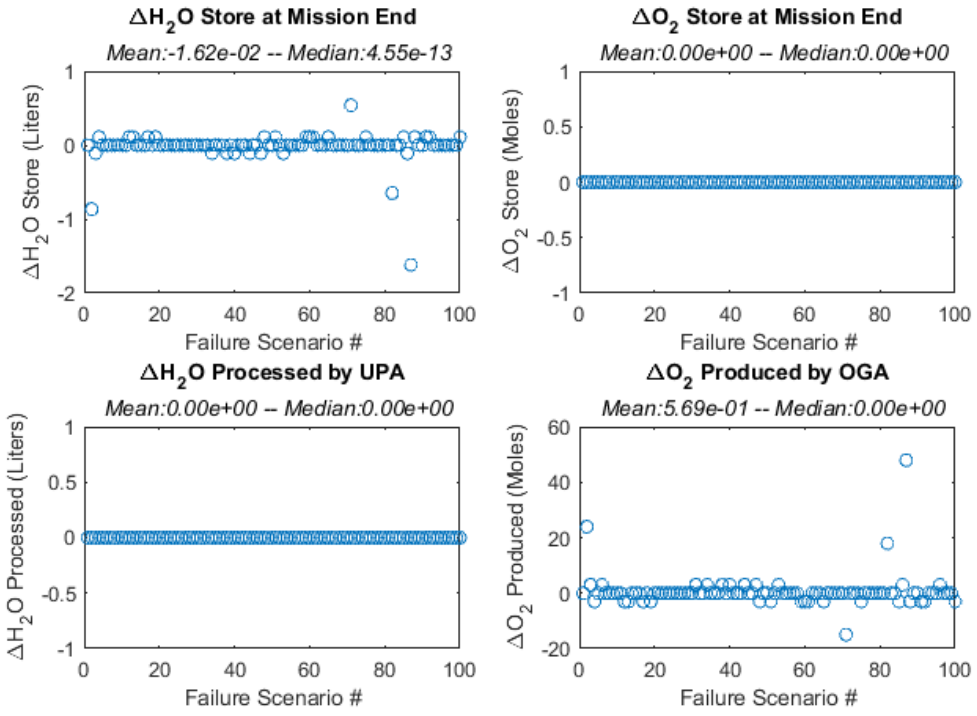

Fig. 20 Change in habitat resource stores and operations from optimized versus default operation of OGA.

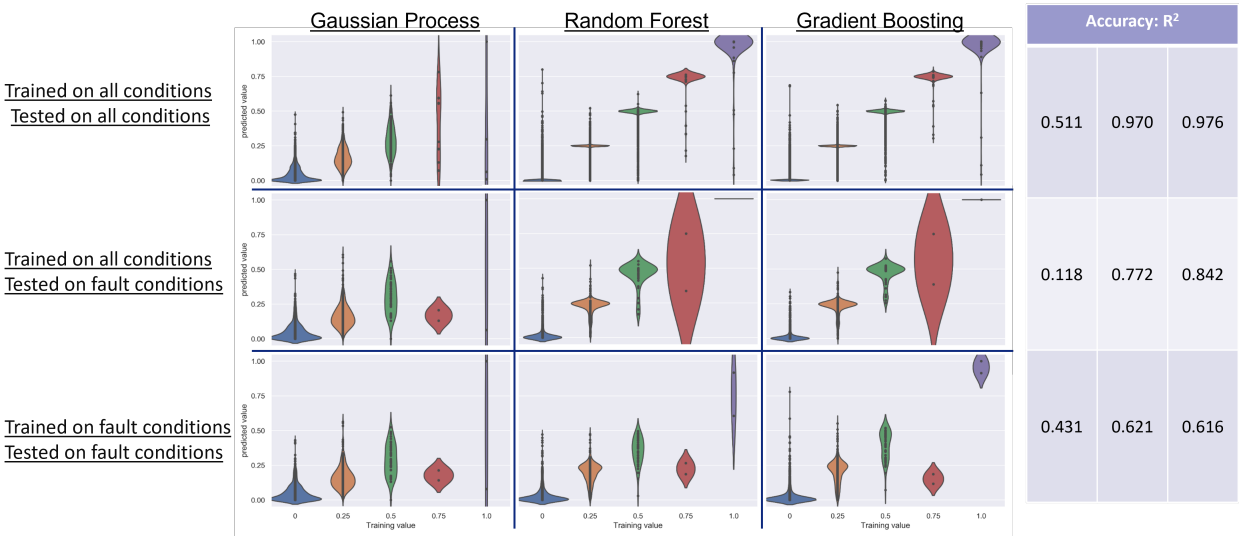

Fig. 21 Predicted versus actual values for the OGA controls for the three different algorithms with three training and testing scenarios. The corresponding R^2 values are shown.

VI. Conclusions

This paper outlines a methodology to improve the resilience of a space habitat ECLSS through subsystem operations. The methodology begins with the setup of a simulation environment and set of failure scenarios. Then, a supervised machine learning algorithm is trained to allocate resources resiliently. The algorithm could then be implemented on a real system and continue to improve as additional data is collected during real operation.

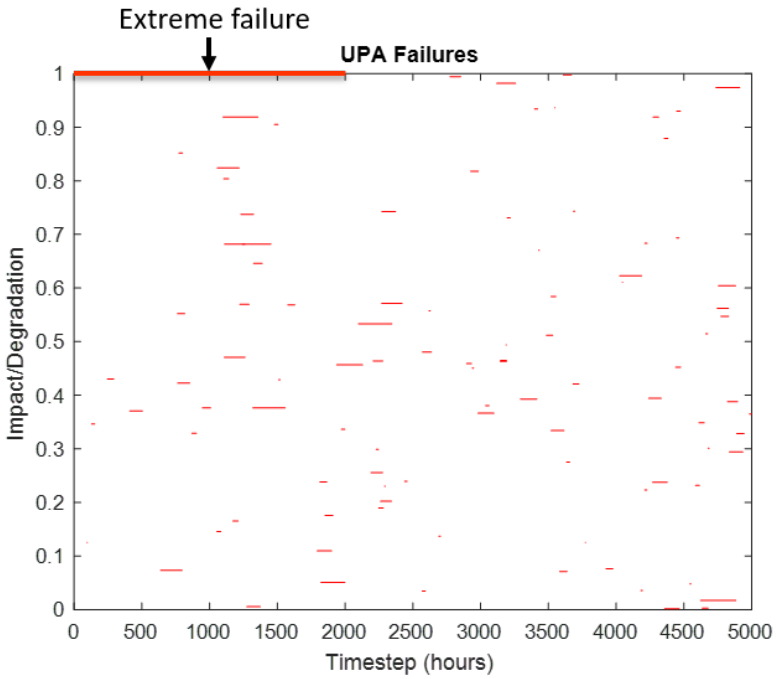

Fig. 22 The extreme UPA failure scenario mapped against previous failure scenarios.

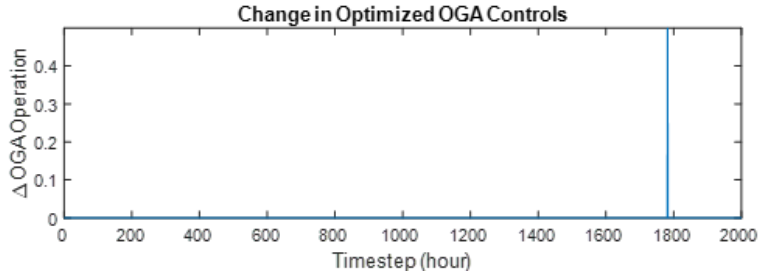

Fig. 23 The difference between default OGA controls and optimized controls for the extreme UPA failure scenario.

The changes required to HabNet in order to demonstrate the methodology are shown as well as how real data from an existing ECLSS could be used in order to develop a failure model to generate the failure scenarios.

The methodology, up to deployment to a real system, is demonstrated in a limited use-case for various failure scenarios in the UPA while the OGA operation is controlled to improve resilience. A zero-order optimization algorithm is found to be able to increase the resilience of the habitat over the existing logic-based controller that is part of HabNet. The optimized controls are then used to train Gaussian Process, Random Forest, and XG Boost algorithms. The XG Boost algorithm is best able to learn the optimized OGA controls.

Future work should test a more encompassing use-case where failures can exist in and the algorithm learns to control all main subsystems. The resilience metric used to determine the optimal controls could also be improved to better reflect the relative importance of the different performance attributes instead of simply averaging them into an aggregate. Reinforcement learning algorithms should also be explored as a way to combine the optimization and development of a resource allocation algorithm.

VII. Acknowledgments

This work was funded by NASA Ames through a Phase I SBIR award, performed under contract 80NSSC19C0376.

VIII. References

[1] "Final Report of the International Space Station Independent Safety Task Force," , 2007.

[2] "NASA - JSC Engineering - Life Support Systems & Environmental Control," , 2013. URL `https://www.nasa.gov/centers/johnson/engineering/life_support_systems/index.html`.

[3] Kortenkamp, D., and Bell, S., "Simulating advanced life support systems for integrated controls research," Tech. rep., SAE Technical Paper, 2003.

[4] Tran, H. T., Balchanos, M., Domerçant, J. C., and Mavris, D. N., "A framework for the quantitative assessment of performance-based system resilience," Vol. 158, 2017, pp. 73–84.

[5] Hollnagel, E., *Resilience engineering in practice: A guidebook*, 2011.

[6] Do, S., "Towards Earth independence-tradespace exploration of long-duration crewed Mars surface system architectures," phdthesis, 2016.

[7] Howe, S. A., Kennedy, K. J., Gill, T. R., Smith, R. W., and George, P., "NASA habitat demonstration unit (HDU) deep space habitat analog," *AIAA Space 2013 conference and Exposition*, 2013, p. 5436.

[8] Howe, A. S., Spexarth, G., Toups, L., Howard, R., Rudisill, M., and Dorsey, J., "Constellation Architecture Team: Lunar Outpost" Scenario 12.1" Habitation Concept," *Earth and Space 2010: Engineering, Science, Construction, and Operations in Challenging Environments*, 2010, pp. 966–988.

[9] Do, S., Owens, A., and Weck, O. d., "HabNet–An Integrated Habitation and Supportability Architecting and Analysis Environment," 45th International Conference on Environmental Systems, 2015.

[10] NASA, "ISS UPA Failure Data," , 2019. Unpublished data.

[11] "MATLAB," , 2018a. The Math Works, Inc.

[12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, Vol. 12, 2011, pp. 2825–2830.