Adaptive Stress Testing Applied To Space Domain Awareness Systems

Johnathan Tucker*
University of Colorado Boulder
Jackson Wagner*
University of Colorado Boulder

Zachary Sunberg
University of Colorado Boulder

ABSTRACT

Advanced cyber-physical systems with machine learning components will play an increasingly important role in gathering and processing information for space domain awareness. This work seeks to formulate and implement a flexible method for validating these cyber-physical systems. The method presented here, known as adaptive stress testing (AST), formulates the validation procedure as a Markov decision process (MDP) where the objective is to find the most probable disturbances that lead the system under test (SUT) to a failure event. Reinforcement learning techniques can be used to solve this MDP. This investigation considers two cases: In the first case, the dynamic state of the SUT is observable, allowing the AST agent to learn a state-based feedback policy. The second case considers a complex piece of software where the structure of the dynamic state is not known. In this case, we use Monte Carlo tree search to choose a sequence of random seeds that result in likely disturbances that cause a failure. To demonstrate these approaches, we consider three test systems. The first is an extended Kalman filter applied to a simple pendulum dynamical system that has some representative features of space domain cyber-physical systems. The second is a Kalman filter applied to the Clohessy-Wiltshire-Hill spacecraft relative motion model. Finally, the third system is an integrated machine learning pipeline designed to detect operational changes in geosynchronous satellite operation.

1. INTRODUCTION

Future space domain awareness systems will increasingly rely on autonomous or semi-autonomous components. For example, cyber-physical systems in the form of sensors combined with machine learning are being proposed to improve space situational awareness [1]. As these cyber-physical systems become more complex, it is increasingly important to understand how they might fail due to natural disturbances or the actions of a non-cooperative actor. The problem of finding such disturbances can be framed as a Markov Decision Process (MDP), and reinforcement learning can be used to find likely sequences of disturbances that will cause a cyber-physical system to fail. This approach is known as adaptive stress testing (AST). The contribution of this work is to apply AST to the field of space domain awareness, specifically finding failures in a machine learning algorithm designed to detect anomalous space object behavior.

This work impacts space domain awareness entities that want to validate a wide range of autonomous systems, for example telescope tasking algorithms [2, 3] or automated anomaly detection [4, 5], that they intend to integrate into their space domain awareness systems. Given the increased interest in autonomous space domain awareness systems, flexible ways of finding how they might fail is highly desirable. Furthermore, after failure determination, the autonomous systems can then be improved and rendered more robust and reliable for future use.

Cyber-physical systems in the form of machine learning algorithms and autonomous systems have the potential to relieve operator burden and increase operational efficiency in existing space systems architectures. Despite this, the adaptation of such systems is slow in the space domain as they can fail silently and unexpectedly. Autonomous system failures can have especially drastic consequences in the space domain where the architectures are critical for both safety and infrastructure. The issue of silent and unexpected failures coupled with the wide range of potential applications for autonomous systems in the space domain necessitates a flexible framework that can determine inputs

*These authors contributed equally

that lead to failure in these systems. This paper aims to provide a flexible framework for validating space domain autonomous systems thus making them more robust to failure. In addition, this work addresses a gap in the literature when it comes to autonomous system validation for the space domain.

Validating autonomous systems in the space domain is particularly difficult as testing them in the real world is intractable due to cost or accessibility constraints. For example, validating an autonomous telescope tasking network would drain telescope resources from their critical application of real-time space domain awareness. To avoid draining critical resources, past validation methods have taken advantage of simulations. Adaptive stress testing is a simulation-based validation framework for autonomous systems that has been developed by autonomous vehicle researchers. AST frames the problem of finding failures as a Markov decision process where the actions are disturbance inputs and the state is a combination of the autonomous system state and the environment in which the system operates. AST interacts with the simulation in a loop, as shown in Fig. 1. Here disturbances x are fed into the simulator whose state s is then updated and passed back to the AST agent along with the reward r. The objective of AST is to find sequences of disturbances that cause the autonomous system under test (SUT) to fail by maximizing this reward signal. Although the framework is flexible as originally formulated, it has not been previously extended to the domain of space situational awareness.

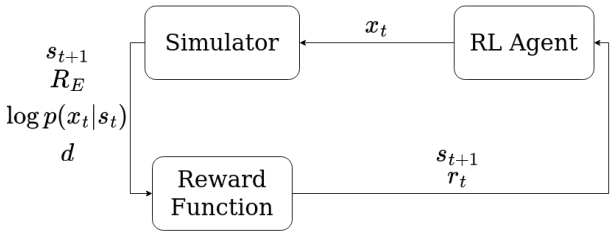


Fig. 1: Adaptive stress testing interaction loop. Symbols are defined in Section 2.2.

This paper extends previous work [6] by formulating a Markov decision process that describes the interaction between a space domain awareness cyber-physical system and an non-cooperative reinforcement learning agent. In this MDP, the actions are disturbances to the system in the form of sensor noise and/or spacecraft maneuvers. The state transition model encompasses the operation of the spacecraft being observed and the cyber-physical space awareness system. Finally, a large positive reward is given to the agent if it causes the system to fail and a smaller reward is given in proportion to the likeliness of the disturbances to guide the agent to find disturbances that seem innocuous. Additional intrinsic rewards may be added to guide the agent in its learning as in [7].

Following the development of the MDP, this work will show how it can be solved using two reinforcement learning agents: a deep reinforcement learning-based soft actor-critic algorithm as well as a Monte Carlo tree search algorithm. The reinforcement learning agent will interact with the autonomous system in a loop by inputting a disturbance and receiving the next state of the autonomous system and a reward at every time step as well as at the end of each episode. This adaptive stress testing process will continue until the reinforcement learning agent finds a sequence of disturbance inputs that lead to a failure in the autonomous system. The adaptive stress testing process has been applied to autonomous aircraft in [8] and to autonomous cars in [9] but has not been implemented in the space domain awareness field. This work will demonstrate the adaptive stress testing framework on three systems. The first system considers a pendulum and an extended Kalman filter with the goal of increasing the Mahalanobis distance between the true and estimated states. The second system considers a Clohessy-Wiltshire-Hill follower satellite and a Kalman filter with the same goal of increasing the Mahalanobis distance between the true and estimated states. The final system considers a one-class support vector machine (OC-SVM) with an optimal control-based estimator (OCBE) developed by Rivera et al. [10] for anomalous space object maneuver detection in both cislunar and geosynchronous orbit regimes. Here AST will attempt to cause the OC-SVM to misclassify a behavior change in the geosynchronous satellite.

2. THEORY

2.1 Markov Decision Processes

The solution methods presented in this paper depend on the agent interacting with an environment in an observe-act loop. Here, the agent will sequentially receive an observation of its state, make a decision on what action it should perform, and then act. This is broadly known as sequential decision making or reinforcement learning [11, 12]. A flexible mathematical formulation for these optimization problems is the Markov Decision Process (MDP) [13]. An MDP is defined using the 5-tuple $\langle S, A, T, R, \gamma \rangle$. At each time step, the MDP system has a state $s \in S$, and a decision-making agent takes actions $a \in A$ according to a policy $\pi(s)$, and then receives reward r based on the reward function $R(s,a)$. The next state $s' \in S$ that the agent transitions stochastically to is determined by the state transition distribution, $T(s'|s,a)$. The fact that the next state s' depends only on the current state s and the action a is known as the Markov property. The final component of the 5-tuple, $\gamma \in (0,1)$, is the discount factor that determines the relative value of immediate and future rewards as shown in Eq. (1) below.

In the context of stress testing, the MDP system that the agent interacts with is known as a simulator. The simulator is the combination of the system under test and the environment that the SUT needs to interact with. For example, if the SUT is a filter then the environment is the corresponding dynamical system and the simulator is the entire filter-dynamical system pipeline. This work makes the assumption that the simulations are finite, or that they terminate after a certain period of time t_0 to t_{end}. The terminal time, t_{end}, occurs either when a failure event is reached or after a maximum number of time steps t_{max}. Following the MDP formulation, the agent will interact with the simulator via inputs and outputs (actions and states). In the context of stress testing, the actions will be disturbances that agent imparts on the simulator and the outputs will be the simulator's state.

The agent will be a model free reinforcement learner whose goal is to optimize their policy $\pi(s)$ through sequential interactions with the simulator. The utility of a policy is estimated using the state-action value function $Q(s,a)$ which is defined as

$$Q(s,a) = \mathop{\mathbb{E}}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t | s_t = s, a_t = a]. \tag{1}$$

Given the agent wants to take the action that maximizes the expected future reward, the optimal policy is:

$$\pi^*(s) = \operatorname*{argmax}_a Q(s,a) \tag{2}$$

As the agent interacts with the environment it will improve the state-action value function and thus the policy. This work focuses on two reinforcement learning approaches: Monte Carlo Tree Search (MCTS) and soft actor-critic (SAC) deep reinforcement learning. Prior to discussion of the reinforcement learning agents the next section describes the objective and reward functions they will be optimizing.

2.2 Adaptive Stress Testing

The adaptive stress testing Markov decision process formulation begins with a simulator that contains the system under test as well as the environment that it interacts with. The state of the AST MDP is the state of the simulator, s, and the actions are the disturbance inputs, x, from the reinforcement learner. The MDP transitions are dictated by the transitions of the environment and the system under test. Given this is a black box simulator we do not need to know the transition function and instead treat the simulator as a generative function. Following [6], to construct the reward function we must first begin with the optimization objective below.

$$\underset{x_0,\ldots,x_{t_{end}}}{\text{maximize}} \quad \prod_{t=0}^{t_{end}-1} p(x_t|s_t) \tag{3}$$

$$\text{subject to} \quad s_{t_{end}} \in E \tag{4}$$

In this formulation, the goal of AST is to find the most likely sequence of disturbance inputs that lead to a failure event. Reinforcement learning requires a reward function that can be evaluated at each step of the Markov decision

process. The following reward function incentivises the agent to optimize the objective above:

$$R(s,x) = \begin{cases} R_E & \text{if } s \text{ is terminal and } s \in E \\ -d & \text{if } s \text{ is terminal and } s \notin E \\ \log(p(x|s)) & \text{otherwise} \end{cases}. \tag{5}$$

The first term in the reward function is a strong reward bonus that the agent earns when a failure is reached. When the simulator does not terminate in a failure state the agent receives a miss-distance penalty. This penalty is an environment dependent heuristic that encourages the agent to reach the failure event. Together these first terms direct the agent to meet the constraint in the objective above. The final term in the reward function is the log-likelihood of each disturbance that the agent selects. Since the sum of the log-likelihoods of the disturbances is the log of the product of the likelihoods, this term will lead the agent to maximize the likelihoods of the disturbances.

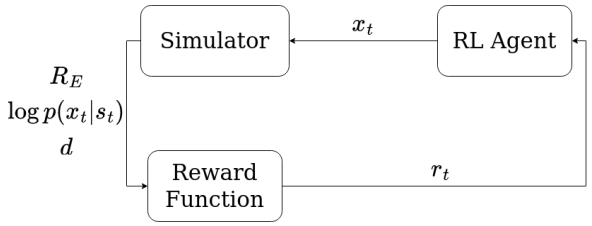


Fig. 2: Seed action adaptive stress testing interaction loop.

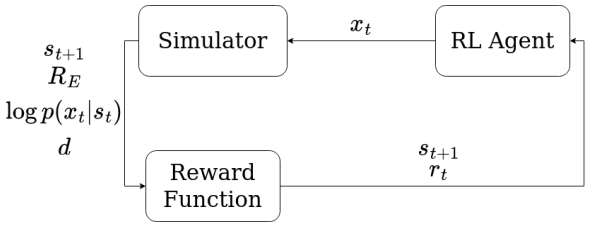


Fig. 3: Deep reinforcement learning adaptive stress testing interaction loop.

The reward function acts to guide the reinforcement learning agent as it interacts with the environment, iteratively selecting disturbances and receiving the subsequent reward in a loop. The two variations of the AST interaction loop can be seen above in Fig. 2 and 3. The key difference between these figures is that the reinforcement learning agent in Fig. 2 does not have access to the state s_t or next state s_{t+1} of the simulator. This variant of AST, seed action AST, is particularly useful when the simulator state is practically inaccessible due to the size of the software or package interactions within the software. In this case, the disturbances are pseudorandom seeds that seed the sampling of internal disturbances within the environment. This means that the state of the simulator is updated in place and the agent does not need access to it, but that it can be recreated deterministically using the sequence of pseudorandom seeds that the agent selects. Despite these differences, both AST loops follow the following general procedure:

- *Initialize:* Reset the environment and system under test to their respective deterministic initial states.
- *Step:* Increment the environment and system under test states by inputting a disturbance into the simulator. The simulator will then output the log-likelihood of the disturbance given the internal simulator state.
- *IsTerminal:* If the simulator has transitioned to a terminal state, the simulator will output either the fail event bonus or the miss distance penalty depending on whether or not the terminal state is a failure state.

2.3 Monte Carlo Tree Search

Monte Carlo Tree Search is an online sampling based method for optimizing MDPs that has proven to be capable of solving high dimensional problems [14, 15]. The tree is formed through a combination of directed sampling based on an estimate of the state-action value function and forward simulation called rollouts. To balance the exploration-exploitation trade off, new nodes of the tree are explored based on a pre-defined exploration policy whose effect decays proportionally to the number of times the node is visited.

This work uses a specific variant of MCTS known as progressive widening (PW) [16, 17], which allows the algorithm to handle large or continuous actions spaces. The MCTS-PW process can be condensed to four stages seen above in Fig. 4 and is described below:

1. *Selection*: The algorithm begins at the root of the search tree, iteratively selecting a node to follow. At every node, the progressive widening criteria is evaluated to determine whether a new action should be created or if

an existing one should be selected instead. More specifically, new actions are sampled based on $A(s) < kN(s)^{\alpha}$ where k and α are user selected hyper-parameters, $A(s)$ is the number of actions at state s, and $N(s)$ is the number of visits to state s. After progressive widening, an action will be selected based on Upper Confidence Bound (UCB) exploration: $Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$. Here c is a user selected hyper parameter that controls the amount of exploration and $N(s,a)$ is the amount of visits to action a at state s. This process continues until a leaf node is reached.

2. *Expansion*: When a leaf node is finally reached, a new node will be added that represents a seed action and the amount of visits is initialized to zero.

3. *Rollout*: From the new seed action node, rollouts are performed according to a pre-determined rollout policy until the episode terminates. In MCTS seed action, the rollout policy is to sample seeds from a uniform distribution. The rollout process allows for the algorithm to estimate the value of the newly added seed action node.

4. *Backpropagation*: Once the value of the new node is estimated the value of the parent nodes can then be updated via backpropagation.

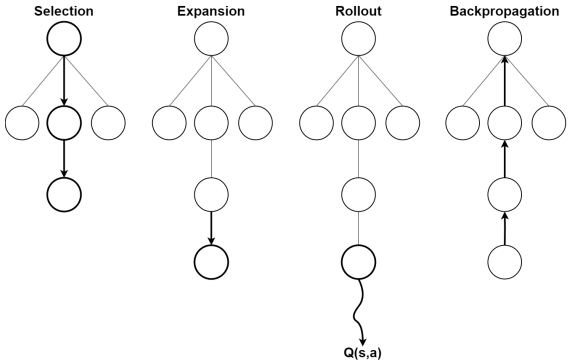


Fig. 4: The four stages of the MCTS-PW process.

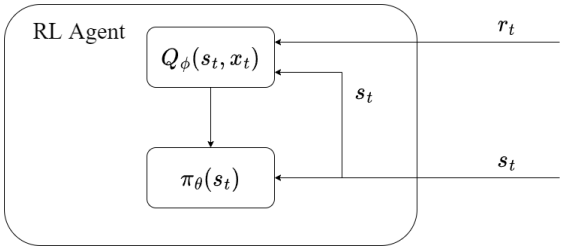


Fig. 5: Zoom in on the reinforcement learning agent in the DRL implementation.

2.4 Deep Reinforcement Learning

Deep Reinforcement Learning provides an alternative to MCTS where the policy is approximated by a dense neural network $\pi_\theta(s)$, where the parameters of the neural network are θ. The advantage of function approximation in reinforcement learning in particular is twofold. First, the state space is generally continuous and information is lost during the discretization process. Second, a neural network is sometimes able to converge on parameters that allow the agent to exploit the reward function well and generalize to similar problems. This work specifically implements the soft actor-critic algorithm [18], which also approximates the state-action value function with a neural network as $Q_\phi(s,a)$.

Soft actor-critic is an entropy regularized reinforcement learning method that trains a policy network as well as one or more Q-function networks (Q_{ϕ_i}). Having multiple Q-function networks allows for the algorithm to avoid maximization bias during training. Fig. 5 provides a detailed view of how the actor and critic take in the state and reward. In the reinforcement learning setting, entropy regularization results in a reward bonus at every time step as so:

$$Q^{\pi}(s,a) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t) + \alpha \sum_{t=0}^{\infty} \gamma^t H(\pi(s_t)) \mid s_0 = s, a_0 = a\right] \tag{6}$$

Here the parameter α is the entropy regularization coefficient that is varied over the course of training to manage the exploration-exploitation trade off and $H(\pi(s))$ is the Shannon entropy defined by $H(X) = \mathrm{E}[-\log p(X)]$. Rewriting the above equation in a recursive Bellman form leads naturally to the loss function for the Q-function networks derived here:

$$Q^{\pi}(s,a) = \mathrm{E}[R(s,a) + \gamma(Q^{\pi}(s',a') - \alpha \log \pi(s')] \tag{7}$$

$$Q^{\pi}(s,a) \approx r + \gamma(Q^{\pi}(s',a') - \alpha \log \pi(s'), \tag{8}$$

leading to the loss function

$$L(\phi_i) = \mathrm{E}[(Q_{\phi_i}(s,a) - (r + \gamma(1-d)(\min_{j=1,2} Q_{\phi_j}(s',a') - \alpha \log \pi_\theta(s')))^2]. \tag{9}$$

The target Q-function network in the above loss function, Q_{ϕ_j} is determined based on the minimization between the two Q-function networks which aids in training stability as well as preventing maximization bias. The policy loss function follows directly from the Q-function network loss and is:

$$L(\theta) = \mathrm{E}[\min_{j=1,2} Q_{\phi_j}(s, \pi_\theta(s)) - \alpha \log \pi_\theta(s)] \tag{10}$$

The parameters of the policy and state-action value function approximators are iteratively improved using the gradient of the above loss functions. The full process for the soft actor critic is shown in Algorithm 1.

Algorithm 1 Soft Actor-Critic

Initialize replay buffer and network parameters
for Number of steps **do**

if Buffer length is less than desired amount **then**
Take a random action in the environment
else
Get action from the policy
end if

Step the environment
Store the trajectory in the replay buffer
Check if done and reset the environment if necessary

if Update step is reached **then**
Sample replay buffer for batch B
for $(s,a,r,s',d) \in B$ **do**
Update the Q-function networks with gradient descent using:
$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s,a) - (r + \gamma(1-d)(\min_{j=1,2} Q_{\phi_j}(s',a') - \alpha \log \pi_\theta(s')))^2$

Update the policy network with gradient descent using:
$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (\min_{j=1,2} Q_{\phi_j}(s, \pi_\theta(s)) - \alpha \log \pi_\theta(s))$
end for
end if

if Test step is reached **then**
Test the agent and record performance statistics
end if
end for

3. DEMONSTRATION SYSTEMS

3.1 Simplified Example: Pendulum

The first problem is defined as a pendulum model and an Extended Kalman filter (EKF) observing the model. This simplified problem allows us to study the behavior of a system that is easy to understand, but is still nonlinear and

periodic. The nonlinear pendulum model used to propagate the dynamics is

$$\begin{aligned} \xi_{t+1} &= \xi_t + T_s\dot{\xi}_t + \frac{T_s^2}{2}\left(\frac{-g}{l}\sin(\xi_t) + \frac{1}{ml^2}w_t\right) \\ \dot{\xi}_{t+2} &= \dot{\xi}_t + T_s\left(\frac{-g}{l}\sin(\xi_t) + \frac{1}{ml^2}w_t\right) \\ y_t &= \xi_t + v_t \end{aligned} \tag{11}$$

using the parameters $T_s = 0.05$ s, $g = 9.81$ m/s^2, $l = 1$ m, and $m = 1$ kg. The initial state is sampled from $\xi_0 \sim U[-\pi/4, \pi/4]$ rad and $\dot{\xi}_0 \sim U[-0.5, 0.5]$ rad/s. The linearized system used in the EKF is given by

$$\begin{aligned} \bar{\boldsymbol{\xi}}_{t+1} &= A_t\bar{\boldsymbol{\xi}}_t + B_t w_t \\ \bar{y}_t &= C_t\bar{\boldsymbol{\xi}}_t + v_t \end{aligned} \tag{12}$$

where $\bar{\boldsymbol{\xi}} = [\bar{\xi}\ \dot{\bar{\xi}}]^T$, with

$$A_t = \begin{bmatrix} 1 + \frac{T_s^2}{2}\alpha_t & T_s \\ T_s\alpha_t & 1 \end{bmatrix}, \qquad B_t = \begin{bmatrix} \frac{T_s^2}{2ml^2} & \frac{T_s}{ml^2} \end{bmatrix}^T, \qquad C_t = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

where $\alpha_t = -(g/l)\cos(\xi_t)$. The process disturbance and measurement noise covariances used in the EKF are

$$W = BB^T\sigma_w^2, \qquad V = \sigma_v^2$$

with $\sigma_w = 0.5$ rad/s^2 and $\sigma_v = \frac{5\pi}{180}$ rad. The initial estimated state covariance P_0 is set equal to the process disturbance covariance. The initial estimated state is set equal to the initial state of the pendulum.

Fig. 6 illustrates the reinforcement learning loop for this problem. The MDP state s of this problem is comprised of the true state ξ, the estimated state $\bar{\xi}$, the upper triangular entries of the estimated state covariance matrix P, and the simulation time t. The action space A is defined as the combinations of process and measurement noise such that

$$A = \{x = (w, v)\ :\ |w| \leq \sigma_w,\ |v| \leq \sigma_v\}.$$

The reward is defined as:

$$R(s,x) = \begin{cases} R_E = 0 & \text{if } s \text{ is terminal and } s \in E \\ -d = -100(2 - 0.2 \times d_M(\boldsymbol{\xi}, \tilde{\boldsymbol{\xi}})) & \text{if } s \text{ is terminal and } s \notin E \\ \log(p(x|s)) & \text{otherwise} \end{cases} \tag{13}$$

where $d_M(\boldsymbol{\xi}, \tilde{\boldsymbol{\xi}})$ is the Mahalanobis distance mahalanobis1936generalized between the true and estimated state

$$d_M(\boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) = \sqrt{(\boldsymbol{\xi} - \tilde{\boldsymbol{\xi}})^T P^{-1}(\boldsymbol{\xi} - \tilde{\boldsymbol{\xi}})},$$

and the failure event E is $d_M(\boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) \geq 5$. The state s is terminal if either $s \in E$ or the simulation's maximum time has been reached, in this case $t_{end} = 5$ s.

The neural networks trained were comprised of two hidden layers of 256 densely connected neurons followed by an output layer, using ReLU activation functions. The networks were trained using the soft actor-critic framework using the parameters shown in Table 1. The entropy regulation coefficient is varied [18] to minimize the loss

$$L(\alpha) = \mathrm{E}[-\alpha \log \pi_\theta(s) - \alpha H] \tag{14}$$

with a learning rate of 3×10^{-4}.

Table 1: Pendulum training hyperparameters.

Hyperparameter	Value
Steps	1×10^6
Replay Buffer Length	5×10^5
Initial Steps	1×10^4
Minibatch Size	256
Batches per step	1
Q Networks	2
Optimizer	ADAM
Learning Rate	3×10^{-4}
Discount (γ)	1.0

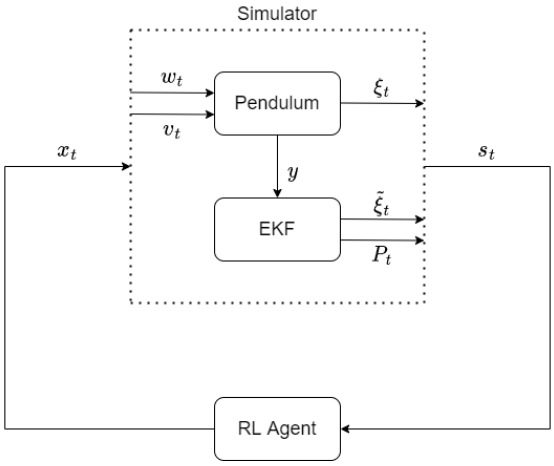


Fig. 6: AST loop for the pendulum problem.

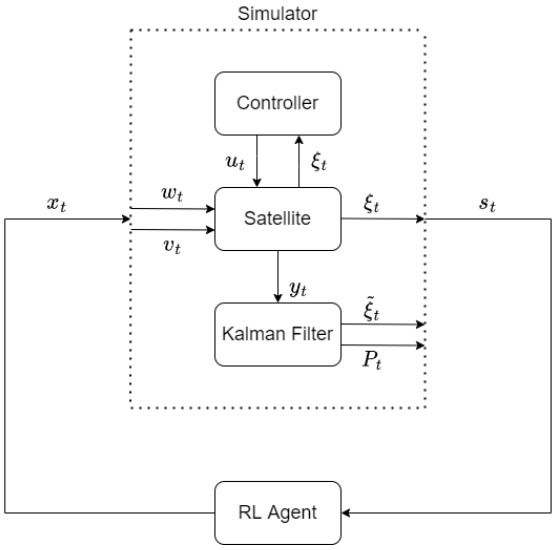


Fig. 7: AST loop for the CWH problem.

3.2 Clohessy-Wiltshire-Hill System

The next test system is slightly more complex and relevant to the space domain: a Clohessy-Wiltshire-Hill (CWH) model of a satellite's dynamics. The CWH equations are a system of coupled second order linear differential equations modeling orbital motion of relative to a reference orbit:

$$\begin{aligned} \ddot{x} - 3n^2 x - 2n\dot{y} &= a_x \\ \ddot{y} + 2n\dot{x} &= a_y \\ \ddot{z} + n^2 z &= a_z \end{aligned} \tag{15}$$

with coordinates x radially outward from the target, y along the orbit track of the target, and z along the orbital angular momentum vector. Because the z dynamics are decoupled, we look at the simplified $x-y$ system with forces $a = [a_x \ a_y]^T = u + w$. In addition, this simplified problem only looked at process noise in the x-direction, and so $w_y = 0$.The system is discretized using zero-order hold with timestep $T_s = 1$ second. To control the system, an infinite-horizon linear quadratic regulator is used with

$$\begin{aligned} Q &= I \cdot 10^{-9} \\ R &= I \end{aligned}$$

resulting in a gain matrix K. Taking the output as y-coordinate of (15) results in a system of the form

$$\begin{aligned} \boldsymbol{\xi}_{t+1} &= (A - BK)\boldsymbol{\xi}_t + Bw_t \\ y_t &= C\boldsymbol{\xi}_t + v_t. \end{aligned}$$

The initial state of this system is sampled from $U[-1,\ 1]$ km for each of the positions, and $U[-10,10]$ m/s for each of the velocities. A linear Kalman filter is constructed with process covariance $W = \sigma_w^2 I_2$ where $\sigma_w = 10^{-3}$ m/s^2, and measurement covariance $V = \sigma_v^2 = 1$ m. The MDP state, reward, and action space followed those used in the pendulum problem, with

$$-d = -500(2 - 0.2 \times d_M(\boldsymbol{\xi}, \tilde{\boldsymbol{\xi}})).$$

The failure event E is similarly defined as $d_M(\boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) \geq 5$, and the maximum time of the simulation is $t_{end} = 500$ s. The soft actor-critic training parameters used were also identical to those used in the pendulum problem shown in Table 1, using a fixed entropy regularization coefficient $\alpha = 1$.

3.3 Geosynchronous Pattern of Life Anomaly Detection System

The final system under test is a pattern of life (POL) anomaly detection system consisting of an OCBE and a one-class support vector machine (OC-SVM). While the previous systems were simplified to introduce the AST approach and concepts, this system is much more complex and realistic.

Following the formulation presented in [10], the ballistic form of the OCBE is capable of reconstructing dynamical mismodeling in the form of accelerations. These accelerations, when combined with the state of the satellite, form the feature vectors that the one class support vector machine learns to form decision boundaries around. Thus, the OC-SVM acts to detect maneuvers and behavior changes in the geosynchronous satellite. The goal of AST is then to determine a sequence of measurement noise disturbances that will cause the OC-SVM to misclassify anomalous satellite behavior as nominal behavior. In this simulator the OC-SVM and OCBE make up the system under test while the satellite is the environment. Fig. 8 shows how the AST agent will interact with the simulator in a loop.

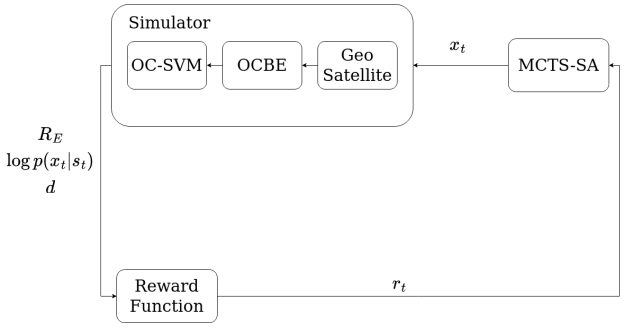


Fig. 8: Diagram of the AST loop for the geosynchronous POL anomaly detection system.

As previously stated, when the agent is not able to access every component of the state, as with large software packages such as this, we will attempt to use seed action MCTS to find a path to the failure event. Therefore, the state of the MDP is the sequence of pseudorandom seeds and the actions are the space of all pseudorandom seeds. The reward function that will guide the search is seen below:

$$R(s,x) = \begin{cases} R_E = 100 & \text{if } s \text{ is terminal and } s \in E \\ -d = 0 & \text{if } s \text{ is terminal and } s \notin E \\ \log(p(x|s)) & \text{otherwise} \end{cases} \tag{16}$$

The software package that contains the geosynchronous system divides fifty orbits of data into forty orbits of training data to construct the decision boundaries in the OC-SVM and 10 orbits of test data to evaluate the performance of the OC-SVM. The satellite's controls are zeroed at 45 orbits which ensures the OC-SVM forms its decision boundary using entirely nominal satellite trajectory data while also ensuring the OC-SVM is evaluated using both nominal and anomalous data. Ultimately, this makes the test window accuracy a more effective representation of the OC-SVM's effectiveness at forming a decision boundary between nominal and anomalous data. It is important to note that the fail event criteria for this problem will be a test window accuracy less than or equal to 45% or a false nominal rate of 30%.

To obtain a baseline for comparison, a random search solver will be implemented on the geosynchronous system. The random search solver will sample from a uniform distribution of pseudorandom seeds to create a trajectory of measurement noise disturbance samples. This will occur in a loop 100 times in an attempt to capture any variation in

the performance of the solver. Similarly, Monte Carlo tree search will have 100 search iterations to find a trajectory of random seeds that cause a failure event. The inclusion of the random search solver will effectively show if there is an advantage to using AST seed action as opposed to sampling seeds at random. Furthermore, if there are seeds that do not yield a failure event during a random search they can be used as a baseline that shows nominal OC-SVM performance.

4. RESULTS AND DISCUSSION

4.1 Pendulum Simplified Example Results

Fig. 9 shows an example training curve for the pendulum problem, with the episodic reward decomposed into its components. For the first 2×10^4 steps the likelihood is negated by the constant fail-to-find penalty, while the distance penalty aids the AST agent in finding the failure event. For the next 2×10^5 steps the agent learns to maximize the reward while consistently achieving the failure event with minimal distance penalty. For the second half of the training the agent almost always achieves the failure event, and can focus on maximizing the likelihood.

Fig. 10 shows two sample trajectories from a common initial condition. The plots are of the true and estimated angular positions and velocities of the pendulum, the noise actions commanded by the AST agent, and the distance to failure. The heuristic policy used for comparison in Fig. 10a is to maximally accelerate the pendulum in the direction of its velocity while maximizing the measurement error:

$$w_t = \sigma_w \operatorname{sign}(\dot{\xi}_t)$$
$$v_t = \sigma_v \operatorname{sign}(\tilde{\xi}_t - \xi_t).$$

The trajectory given by one of the learned policies in Fig. 10b spends much less time at its action bounds while lengthening the time to failure, which corresponds to noise coming from a normal distribution with mean zero. The sign of the process noise is the negative of our heuristic, while the sign of the measurement noise matches our heuristic. AST validates our intuition as to what combination of measurement and process noise, given the constraints, maximizes the estimated state error relative to the estimated covariance.

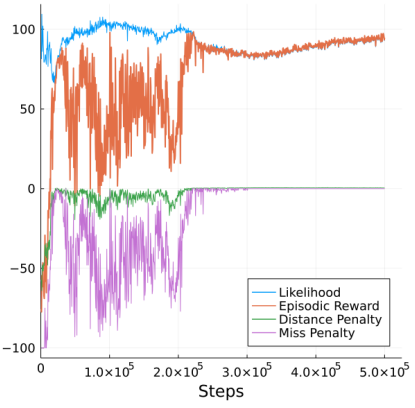
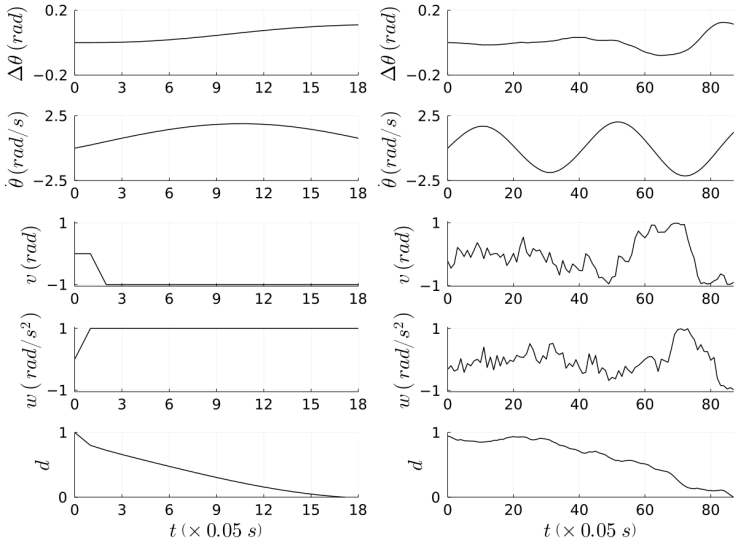


Fig. 9: Sample learning curves for the pendulum example. The episodic reward is plotted along with its components.



(a) Heuristic (b) AST

Fig. 10: Sample trajectories for the Pendulum Example. The process and measurement noise values are normalized to fall in [-1,1].

4.2 Clohessy-Wiltshire-Hill Example Results

Fig. 11 shows that unlike the pendulum example, policies resulting from untrained networks are able to reach the failure event. Here the AST agent learns to increase the likelihood of the noise while still reaching the failure event. The agent is able to achieve this higher likelihood failure through a policy of zero mean noise followed by maximizing the magnitude of the noise samples after $t = 350$ s as shown in Fig. 12. This policy corresponds to the Kalman filter being unable to sufficiently compensate for constant biases in the process and measurement dynamics. Unlike the pendulum problem, using a fixed entropy regularization coefficient resulted in significantly improved performance during training.

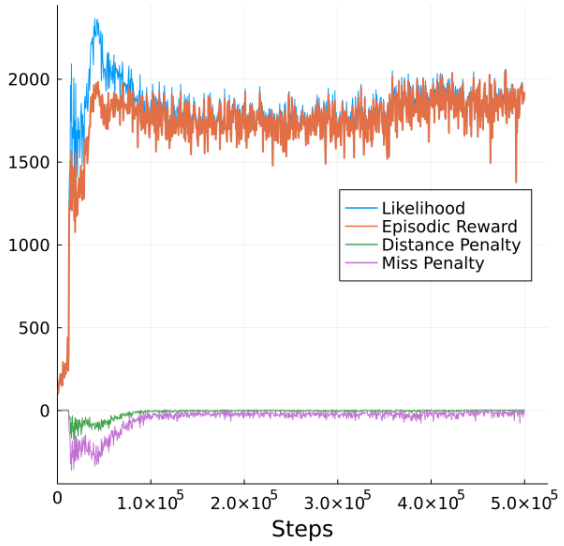


Fig. 11: Sample learning curve for the CWH example. The episodic reward is the sum of the Likelihood, Distance Penalty, and Miss Penalty components.

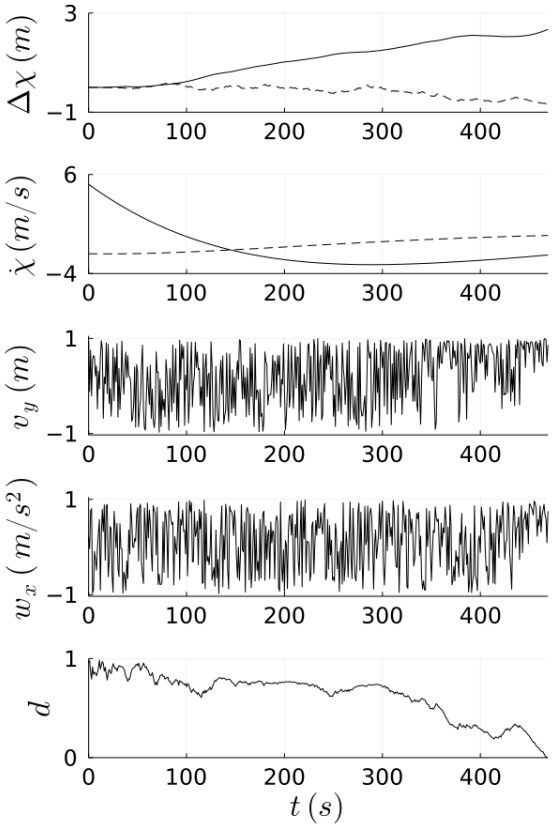


Fig. 12: Sample trajectory for the CWH Example. In the first two plots $\chi = [x,\ y]$, the solid and dashed lines correspond to the x and y directions respectively. The process noise is normalized to fall in [-1,1].

4.3 Geosynchronous Satellite Example Results

Recall that a random search solver, where seeds are sampled from a uniform distribution, is the baseline of comparison to seed action AST. Fig. 13 depicts the relative spacecraft trajectory over fifty orbits that is used in the geosynchronous system for both the baseline and seed action AST approaches. None of the random search solver seed action trajectories resulted in a failure event, while the MCTS agent did produce a trajectory that led to a failure event.

A baseline trajectory was randomly sampled from the set of random search solver trajectories. The baseline trajectory train window accuracy is 84.37% and the test window accuracy is 70%. The MCTS seed action agent's train window accuracy is 77.5% and test window accuracy is 45%. The decrease in test accuracy is a result of the classifier labeling every test window data as anomalous.

Fig. 14 shows a comparison between the filtered acceleration estimates from the OCBE for both the random search solver (left) and the seed action MCTS agent (right). First, the accelerations in the x and y directions show the AST

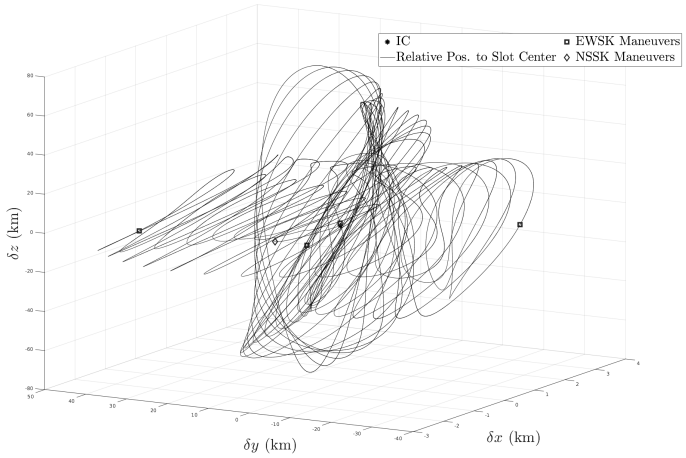


Fig. 13: Spacecraft trajectory with stationkeeping relative to the slot.

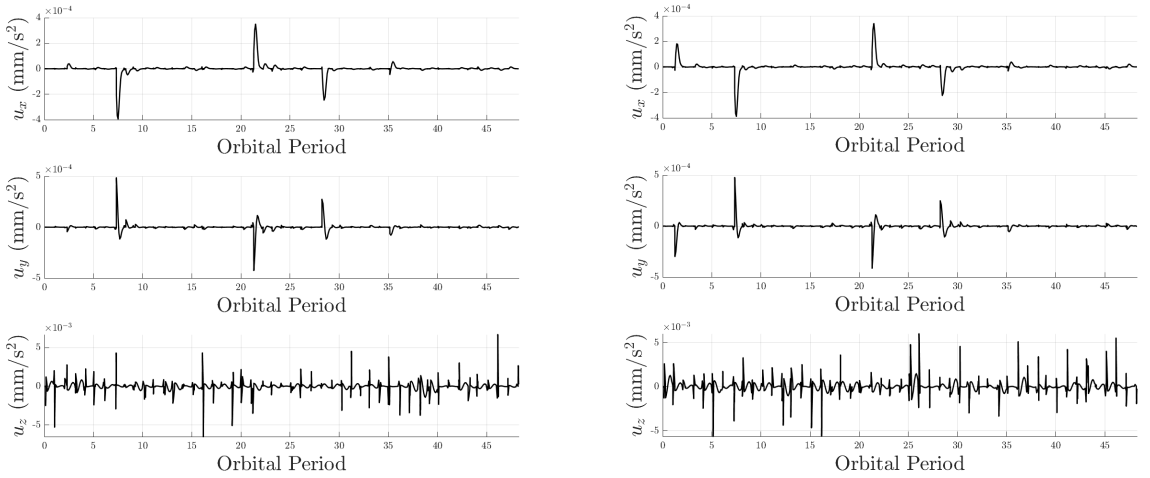
agent has injected noise such that the OCBE estimates an additional acceleration within the first few orbits. Given the training data for the OC-SVM is drawn from the first 40 orbits, this is evidence that the AST agent is "corrupting" the nominal satellite trajectories in the training data. This would naturally lead the OC-SVM to misclassify nominal trajectories as anomalous during test time. The design goal of the failure event definition is to encourage the AST agent to find a trajectory of random seeds that cause false nominal predictions from the classifier. This result shows the agent is able to exploit the test time accuracy component of the fail event definition.

5. CONCLUSION

Cyber-physical systems have the potential to relieve operator burden and increase operational efficiency in space systems. However, adoption of machine learning and autonomous systems into space system architectures is slowed by lack of trust and unknown failure modes that these cyber-physical systems could exhibit. This work describes the adaptive stress testing framework and how it can be used to search for failure scenarios in space-related cyber-physical systems. Adaptive stress testing frames this search as a Markov Decision Process where the actions are disturbances that a reinforcement learning agent chooses and the state is the combined state of the system under test and the environment. This work considers three cases and three corresponding solution methods and applies these to space domain awareness problems. In the first two cases, the SUT and environment states are observable, and likely failures are found using deep reinforcement learning. In the third case, the state is not observable, and a sequence of pseudorandom seeds or disturbances that lead to failures is found using Monte Carlo Tree Search

To demonstrate AST with a deep reinforcement learning agent effectively, two fully observable simulations based on dynamic systems being observed by Kalman filters were formulated. The dynamics of the first simplified example is a swinging pendulum, while the second system is based on a satellite performing station-keeping maneuvers. In both simulations the state of the MDP was the combined state of the physical system, estimated state of the system, and estimated covariance while the actions were comprised of process disturbances and measurement noise. Training the AST agents led to policies highlighting weaknesses of the tracking filters: disturbances based on the true and estimated states of the pendulum, and constant disturbances corresponding to sensor bias and modeling error of the satellite.

The ultimate goal of this work is to contribute an effective AST framework that could be applied to a more complex SDA system where a one class support vector machine detects behavior change in a geosynchronous satellite using ballistic OCBE control estimates. As there are extensive dependencies on different software packages in this scenario, the full state of the simulator is not readily available at every time step. Therefore, the seed action AST framework is used to determine pseudorandom seeds that generate the measurement noise samples in the ballistic OCBE with the goal of causing the OC-SVM to misclassify a behavior change. The results showed the AST agents ability to exploit



(a) Filtered acceleration for the baseline trajectory.

(b) Filtered acceleration for the MCTS noise trajectory.

Fig. 14: Comparison of the filtered acceleration estimate output from the OCBE.

access to the training data by injecting noise that would cause the nominal data to look anomalous thus making the decision boundary less precise and led to the test accuracy of 45%.

There are several possible directions for future improvement. First, this study only considered individual systems in isolation. Future work should take advantage of the differential adaptive stress testing framework which is used to compare two cyber-physical systems meant for the same application. This framework could be leveraged to adaptively stress test two different methods of anomaly detection which could provide insights into the strengths and weaknesses of each. To promote the ability for the learned policies to transfer across cyber-physical systems in a more useful manner, future work should investigate the re-framing of the deep reinforcement learner using successor features. The successor feature framework takes advantage of a value function representation that decouples the reward and dynamics of the environment and has been shown to improve a reinforcement learning agents ability to transfer across tasks [19].

6. ACKNOWLEDGEMENTS

The authors would like to thank L3 Harris Technologies for providing funding for this research and Derek Kingrey, Dustin Platter, and Galen Nickey for technical advice. This article contains the opinions and conclusions of its authors and not L3 Harris Technologies or its employees. In addition, the authors would like to thank the Vision, Autonomy, and Decision Research Lab for providing their behavior change detection simulation for stress testing.

7. REFERENCES

[1] SM Lederer, EG Stansbery, HM Cowardin, P Hickson, LF Pace, KJ Abercromby, PW Kervin, and RJ Alliss. The nasa meter class autonomous telescope: Ascension island. Technical report, National Aeronautics and Space administration Houston TX Lyndon B Johnson . . . , 2013.

[2] Andris D Jaunzemis, Marcus J Holzinger, and K Kim Luu. Sensor tasking for spacecraft custody maintenance and anomaly detection using evidential reasoning. *Journal of Aerospace Information Systems*, 15(3):131–156, 2018.

[3] Zachary Sunberg, Suman Chakravorty, and Richard Scott Erwin. Information space receding horizon control for multisensor tasking problems. *IEEE transactions on cybernetics*, 46(6):1325–1336, 2015.

[4] Yu Gao, Tianshe Yang, Minqiang Xu, and Nan Xing. An unsupervised anomaly detection approach for spacecraft based on normal behavior clustering. In *2012 Fifth International Conference on Intelligent Computation*

Technology and Automation, pages 478–481. IEEE, 2012.

[5] Sylvain Fuertes, Gilles Picart, Jean-Yves Tourneret, Lotfi Chaari, André Ferrari, and Cédric Richard. Improving spacecraft health monitoring with automatic anomaly detection techniques. In *14th international conference on space operations*, page 2430, 2016.

[6] Ritchie Lee, Ole J Mengshoel, Anshu Saksena, Ryan W Gardner, Daniel Genin, Joshua Silbermann, Michael Owen, and Mykel J Kochenderfer. Adaptive stress testing: Finding likely failure events with reinforcement learning. *Journal of Artificial Intelligence Research*, 69:1165–1201, 2020.

[7] Anthony Corso, Peter Du, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Adaptive stress testing with reward augmentation for autonomous vehicle validatio. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 163–168. IEEE, 2019.

[8] Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. Adaptive stress testing of airborne collision avoidance systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 6C2–1. IEEE, 2015.

[9] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2018.

[10] Karina Rivera, Anthony Zara, Daniel Aguilar-Marsillach, Marcus Holzinger, Ian Elliott, and Natasha Bosanac. Patterns of life and maneuver detection for cislunar trajectory maintenance. 2021.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. *Algorithms for decision making*. MIT press, 2022.

[13] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[14] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[16] Guillaume M JB Chaslot, Mark HM Winands, H Jaap van den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[17] Robert J Moss, Ritchie Lee, Nicholas Visser, Joachim Hochwarth, James G Lopez, and Mykel J Kochenderfer. Adaptive stress testing of trajectory predictions in flight management systems. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2020.

[18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[19] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.