Test & Evaluation/Science & Technology Program

NATO S&T Meeting

Autonomous & Artificial Intelligence Test (AAIT)

Technology Area

Executing Agent:
Vernon Panei
vernon.panei@navy.mil

Chief Engineer/Presenter:
Jonathan Elliott
jonathan.elliott@navy.mil

Mission

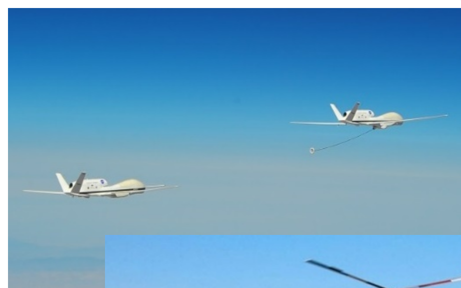
<u>Develop technologies</u> that significantly advance the science of testing autonomous systems

These technologies <u>improve the safety and user trust</u> in autonomous system tests and operations

Autonomous Cargo Transport

Autonomous Troop Transport

Autonomous Aerial Refueling

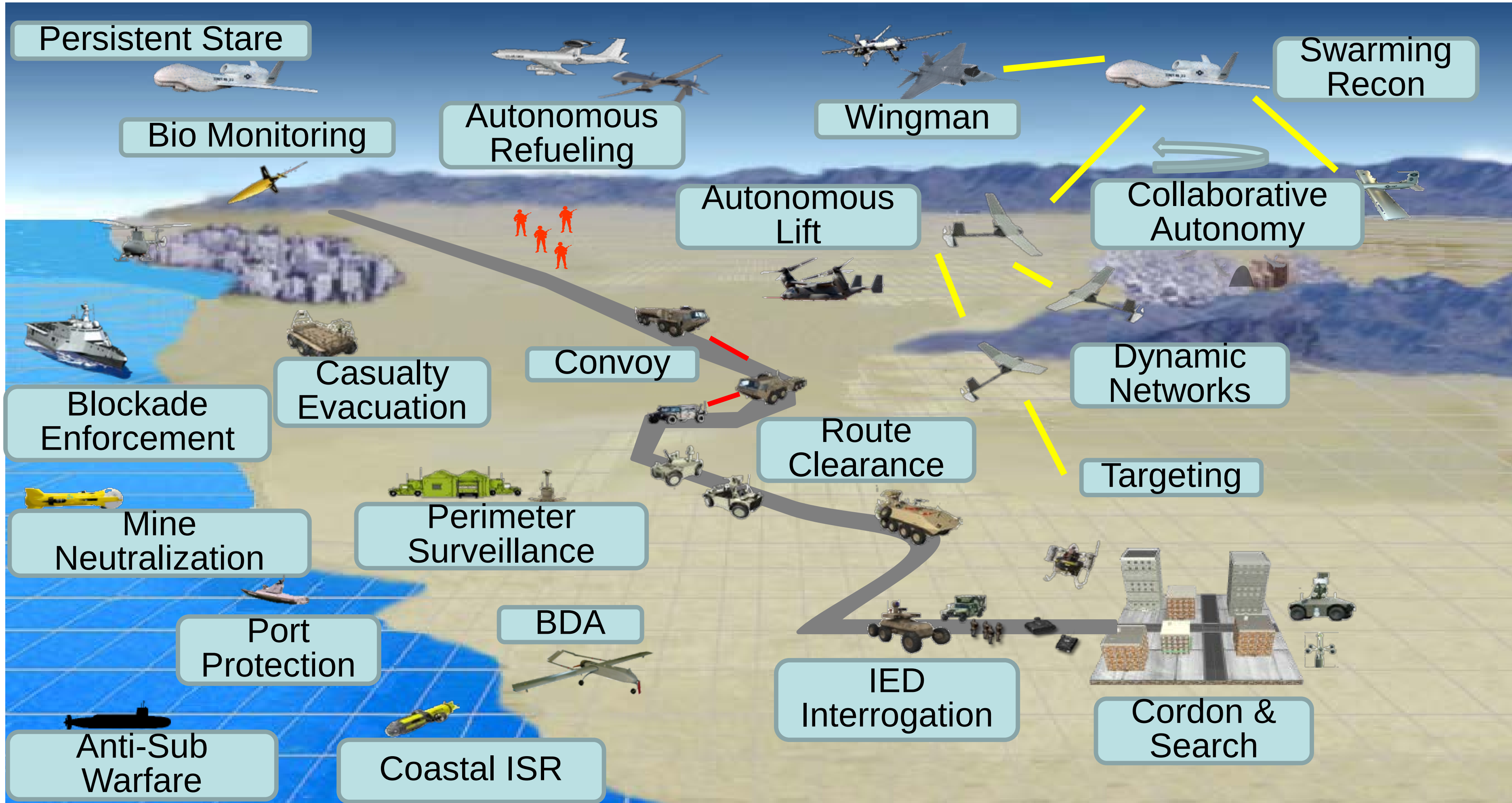


Autonomous Aerial Transport

Autonomous Undersea Survey

Autonomous Port Protection

Unmanned and Autonomous Systems Test Operational Scenarios



Better Testing of Autonomous Systems leads to greater Warfighter Trust in their mission performance

Autonomy Priorities

- Over 150 active programs employing a spectrum of autonomy
- Services have identified 23 programs as high priority

Autonomy Testing Challenges

The testers are asking hard questions, like these:

- How do I measure human-machine interaction effectiveness?
- How do I design tests for manned-unmanned team coordination?
- How do I develop tests for evoking emergent behavior?
- How do I assess the decision process and cognition, especially with a learning system?
- How do I design tests for distributed teams and swarms interaction?
- How do I develop tests that fully exercise rule coverage?
- How do I create sufficiently smart actors for an immersive environment?
- How do I identify the most salient tests based on SUT parameters and mission?
- How do I measure adaptivity and emergence?
- How do I assess maturity of learning systems?
- Can I test it safely?
- Can I test it in budget / on time?

AAIT Wants These Questions Answered

Eras and Testing Challenges

Automated Era…

- **Preprogrammed commands** with explicit tasks
- Deterministic behavior
- Dependence on reliable communications

Testers need to…

- **Verify action**
- Measure physical properties such as position, path, speed, separation distance, completion of event

Autonomous Era…

- Explicit tasks
- Decisions made based on environmental and contextual conditions
- Behaviors are preprogrammed
- **Structured independence,** locally aware

Testers need to…

- **Verify reasoning process**, not just action
- Verify that SUT perceived situation correctly and meant to act the way it did

Intelligent Era…

- **Independent reasoning**
- Experience driven
- Adaptive
- High decision complexity
- UAS-to-UAS cooperation
- Adversary interaction
- Unstructured independence
- Distributed understanding

Testers need to…

- **Verify cognition**
- Recognize that knowledge and decision ability are a function of time and experience
- Need to verify SUT had sufficient knowledge of a situation to form correct intent
- Need to verify combination of multiple mission goals

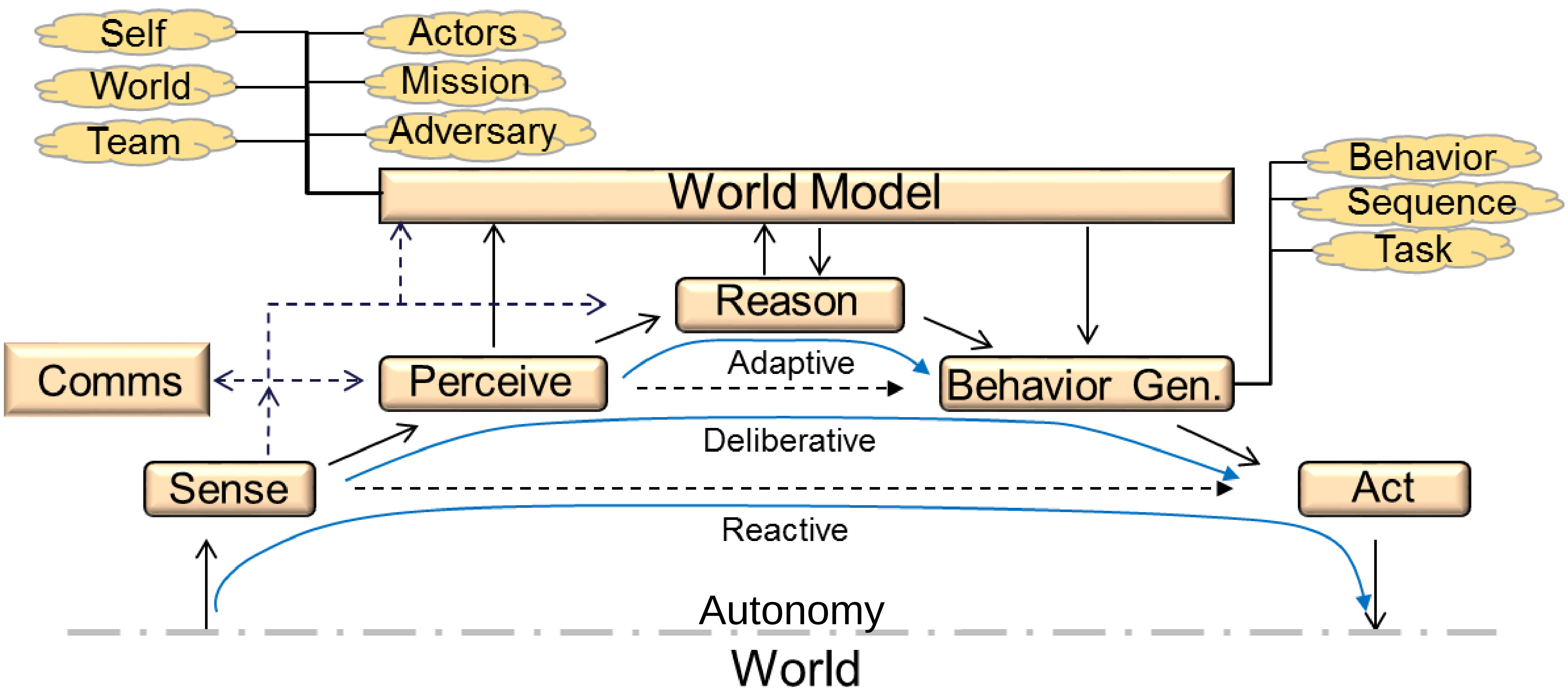
Near — Mid — Far — *Time*

Our Focus is on Testing *Autonomy*

SUT= System Under Test

Autonomous Systems Overview

Test Technologies are needed to measure and assess the **INTERNAL FUNCTIONS** of the autonomy

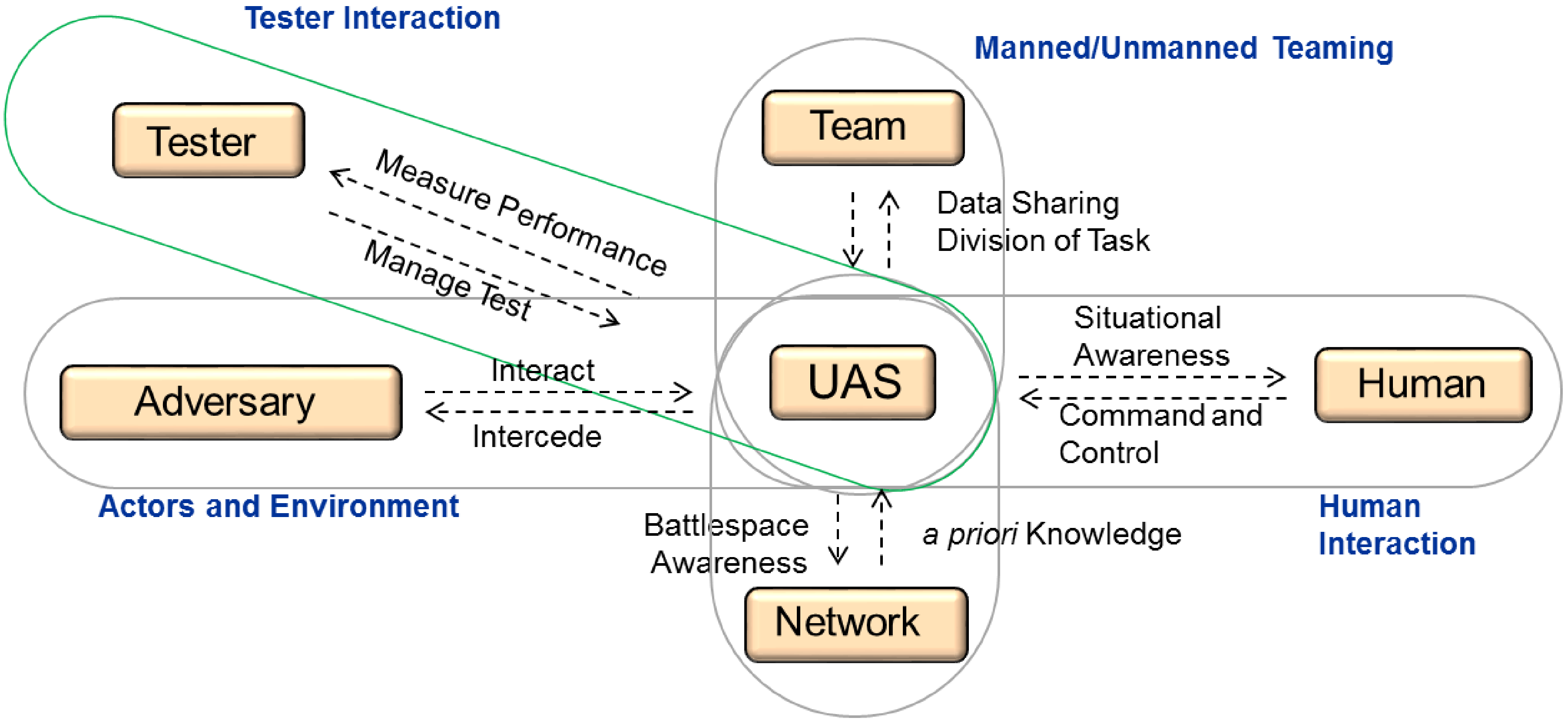


NAVAIR Public Release- 2018-358 'Approved for Public Release; distribution is unlimited'.

Unclassified

Autonomous Systems Overview

Test Technologies are needed to measure and assess the **EXTERNAL INTERACTIONS** of the autonomy



Autonomous Systems Overview

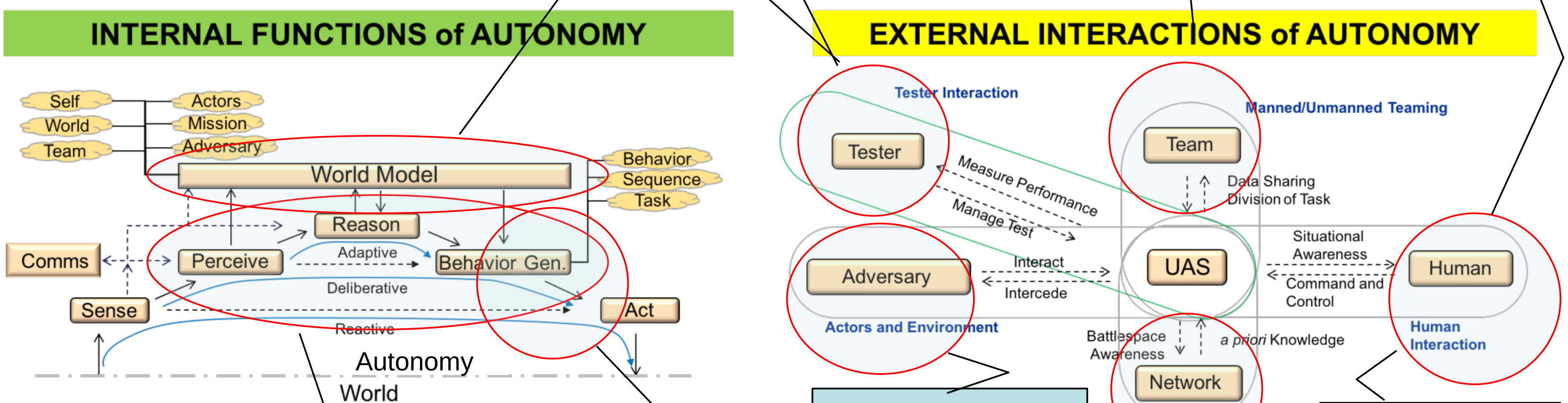
Common architectures, M&S environments & test tools enabling “designed-in” test interfaces, and tester data/knowledge sharing

Evaluate Swarm Performance & Behavior

Intelligent Test Planning to Bound Performance & Test Space

Agile & Adaptive Test Ranges

Optimize Human Machine Relationship



Evaluate Internal Autonomy Functions in LVC Environment

Intelligent Threats

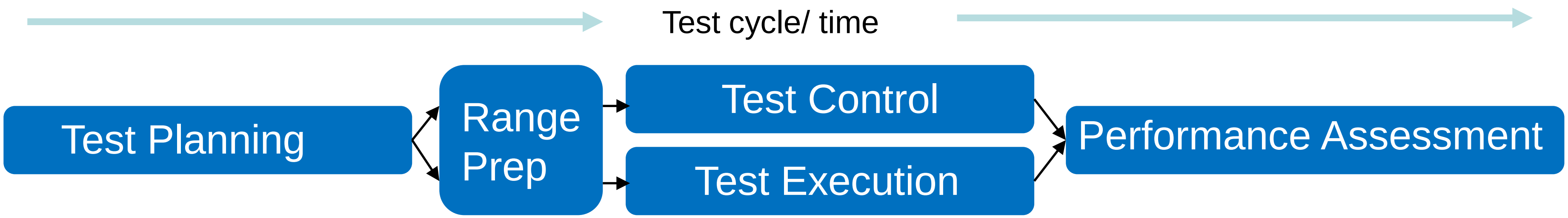
Evaluate Trust

Quantify Learning Ability

Emulate & Connect to Cloud

Common test cases, policy, metrics and methods enabling consistency/continuity/reciprocity across DoD as well as other federal and state regulatory bodies in licensing/certification/VV&A

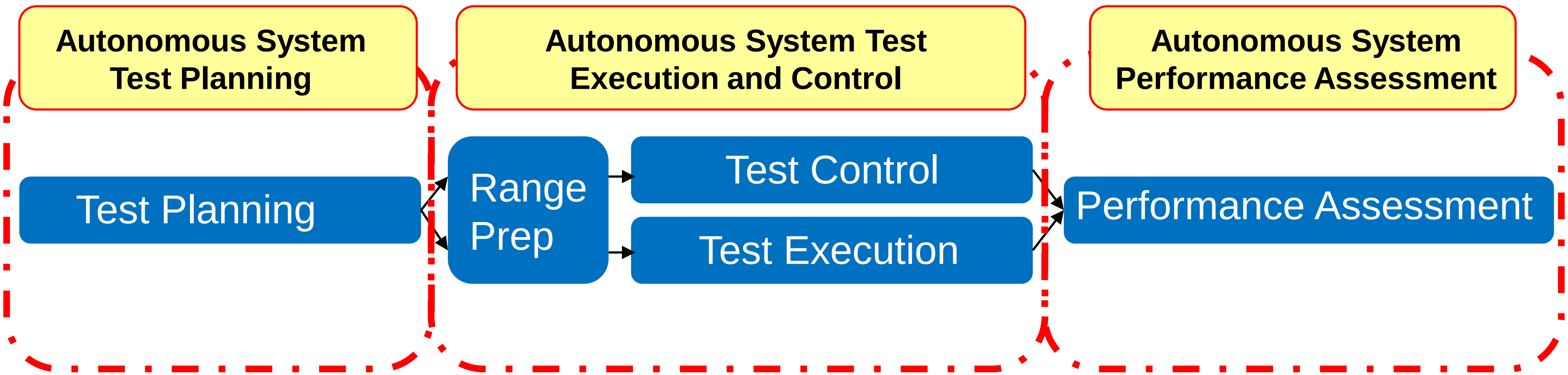
AAIT Tester Timeline



- **Test Planning** – long lead time, SUT design and/or program not necessarily set
- **Range Prep** – SUT and/or SUT models identified, short lead time
- **Test Control** – SUT present, focus on safety of range/personnel/SUT
- **Test Execution** – SUT present, focus on efficient test, proper stimulus, data collection
- **Performance Assessment** – SUT and/or test environment no longer available, focus on data, feedback to next test cycles

Different perspectives/technologies needed across the test cycle

UAST Domain Partitions



- **Tester Timeline perspective drives UAST domain partitions**
  - Autonomous System Test Planning
  - Autonomous System Test Execution and Control
  - Autonomous System Performance Assessment

Test Innovations Needed to Identify Limitations, Compress the Timeline and Expedite Soldier Acquisition

UAST Roadmap

Automated Era	Autonomous Era	Intelligent Era

Use Cases

Land Route Protection
Logistics Soldier Offload
Battle space awareness ISR
Logistics Transport
Pallet Loader
Tactical Urban Support
Urgent Logistics/ Casualty Evac
Urgent Logistics
Kinetic Attack
Transit and Refuel
Transport and Drop
Electronic Attack
UUV Large Area Sweep
USV Large Area Sweep
UUV Payload/Sensor Deploy
USV ASW
UUV Coastal ISR
USV Port/Maritime Force Protection

- **Roadmap driven by use-case assessment from tri-service working group**

UAST Roadmap

Automated Era	Autonomous Era	Intelligent Era

Use Cases

- Land Route Protection
- Logistics Soldier Offload
- Battle space awareness ISR
- Logistics Transport
- Pallet Loader
- Tactical Urban Support
- Urgent Logistics/ Casualty Evac
- Urgent Logistics
- Kinetic Attack
- Transit and Refuel
- Transport and Drop
- Electronic Attack
- UUV Large Area Sweep
- USV Large Area Sweep
- UUV Payload/Sensor Deploy
- USV ASW
- UUV Coastal ISR
- USV Port/Maritime Force Protection

Test Planning Needs

- Exploit white box methods for decision assessment
- Design tests for distributed teams and swarms interaction
- Identify failure boundaries for test selection
- Identify most salient tests based on SUT parameters
- Develop tests to exercise rule coverage
- Design tests for exercising and software resiliency
- Exploit white box methods for fault analysis
- Design tests for MUM team coordination
- Develop test for evoking emergence

Test Execution/Oversight Needs

- Develop immersion for close maneuvering and formation
- Correlate separate but networked awareness and decision
- Develop dynamic virtual geo-fences
- Provide embedded prediction for collision safety
- Tools to preload experience
- Synchronize distributed adversarial response
- Develop immersive environment for complex terrain
- Integrate dynamic LVC stimulus in open air ranges
- Integrate smart actors in immersive environment
- Adapt test environment to multiple smart agents

Performance Assessment Needs

- Measure distributed situational awareness and decision making
- Develop measures for negative requirements
- Detect and trace faults and fault propagation
- Measure human machine interaction effectiveness
- Measure adaptivety and emergence
- Assess maturity of learning systems
- Assess rule adherence and mission coverage
- Ability to identify and predict aberrant behavior from observation

- **Produce needs and gaps partitioned into UAST domains**

UAST Roadmap

Automated Era	Autonomous Era	Intelligent Era

Use Cases

- Land Route Protection
- Logistics Soldier Offload
- Battle space awareness ISR
- Logistics Transport
- Pallet Loader
- Tactical Urban Support
- Urgent Logistics/ Casualty Evac
- Urgent Logistics
- Kinetic Attack
- Transit and Refuel
- Transport and Drop
- Electronic Attack
- UUV Large Area Sweep
- USV Large Area Sweep
- UUV Payload/Sensor Deploy
- USV ASW
- UUV Coastal ISR
- USV Port/Maritime Force Protection

Test Planning Needs

- Exploit white box methods for decision assessment
- Design tests for distributed teams and swarms interaction
- Identify failure boundaries for test selection
- Identify most salient tests based on SUT parameters
- Develop tests to exercise rule coverage
- Design tests for exercising and software resiliency
- Exploit white box methods for fault analysis
- Design tests for MUM team coordination
- Develop test for evoking emergence

Test Execution/Oversight Needs

- Develop immersion for close maneuvering and formation
- Correlate separate but networked awareness and decision
- Develop dynamic virtual geo-fences
- Provide embedded prediction for collision safety
- Tools to preload experience
- Synchronize distributed adversarial response
- Develop immersive environment for complex terrain
- Integrate dynamic LVC stimulus in open air ranges
- Integrate smart actors in immersive environment
- Adapt test environment to multiple smart agents

Performance Assessment Needs

- Measure distributed situational awareness and decision making
- Develop measures for negative requirements
- Detect and trace faults and fault propagation
- Measure human machine interaction effectiveness
- Measure adaptivety and emergence
- Assess maturity of learning systems
- Assess rule adherence and mission coverage
- Ability to identify and predict aberrant behavior from observation

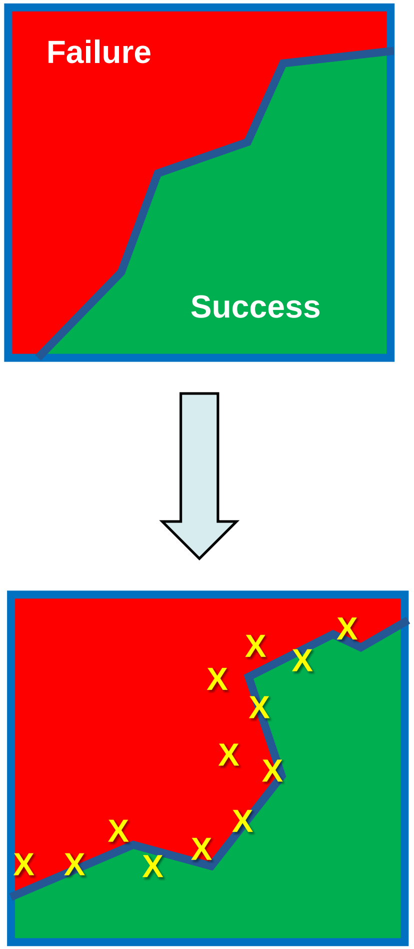
UAST Investment

- RAPT
- STAA
- DIPT
- DDT
- RIOT
- Future Project
- ATAS
- TACE
- Future Project

- **Investment**

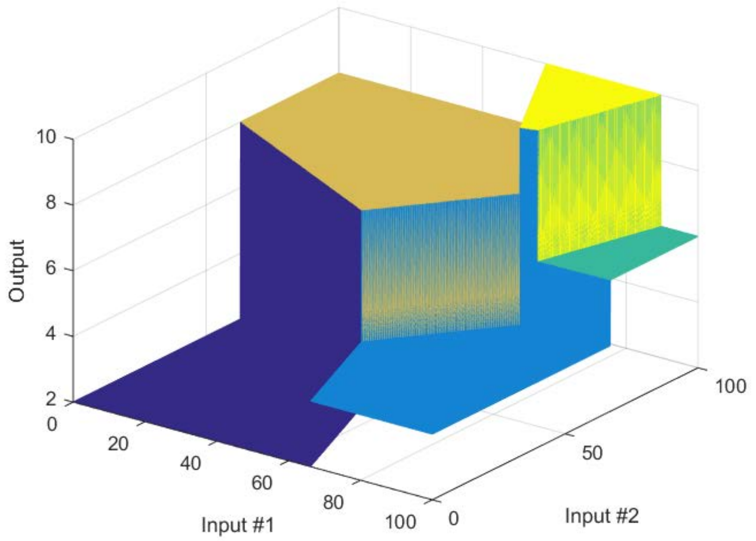
Range Adversarial Planning Tool (RAPT)

- **Autonomy Test Question:**
  - How does a tester identify the most relevant tests for OAR?
  - How does a tester ensure Autonomous System has been fully “exercised” and emergent behavior identified?
- **Proposal:**
  - Develop software to generate mission simulations using adaptive sampling techniques to:
    - ➢ Identify critically-ranked, performance-stressing scenarios
    - ➢ Identify pass/fail boundaries



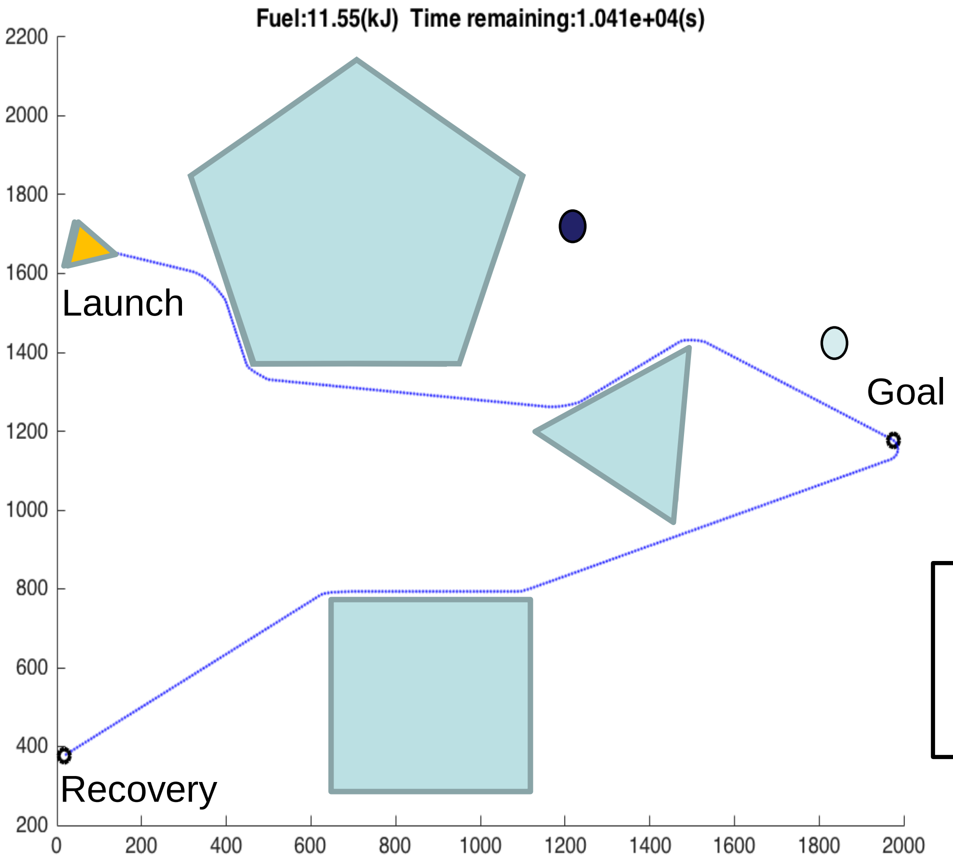
Range Adversarial Planning Tool (RAPT)

- **Autonomy Test Question:**
  - How does a tester identify the most relevant tests for OAR?
  - How does a tester ensure Autonomous System has been fully “exercised” and emergent behavior identified?
- **Proposal:**
  - Develop software to generate mission simulations using adaptive sampling techniques to:
    - Identify critically-ranked, performance-stressing scenarios
    - Identify pass/fail boundaries



Critical transitions between performance modes are inherently discontinuous

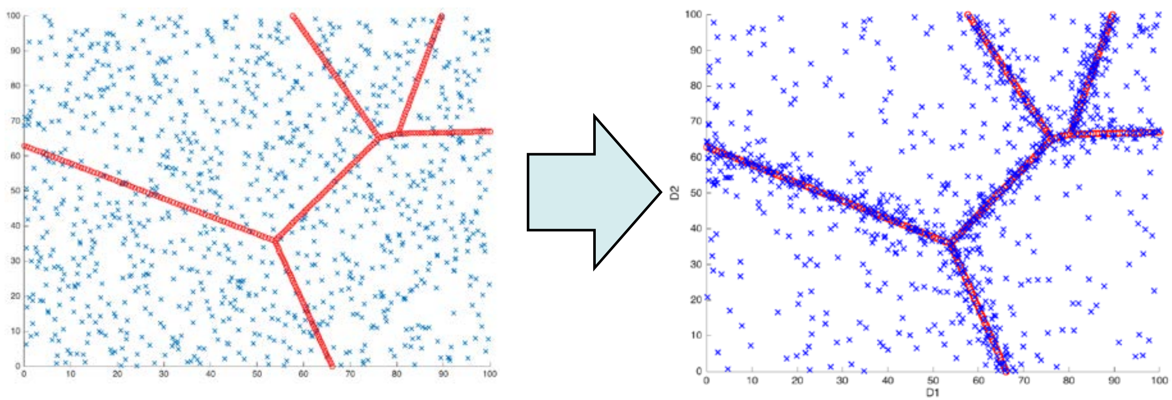
RAPT Scenario Comparison



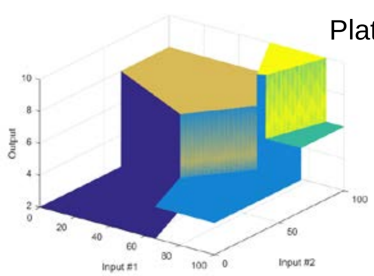
Determine where small changes in the environment can cause drastic changes in behavior

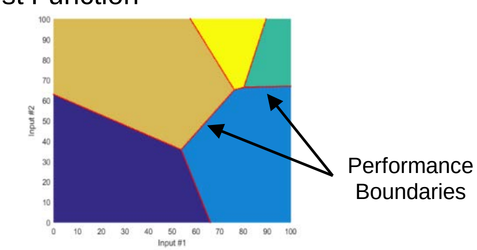
RAPT: Adaptive Sampling

Goal: With a limited number of available simulation runs, create a set that provides the maximum amount of information about the boundaries

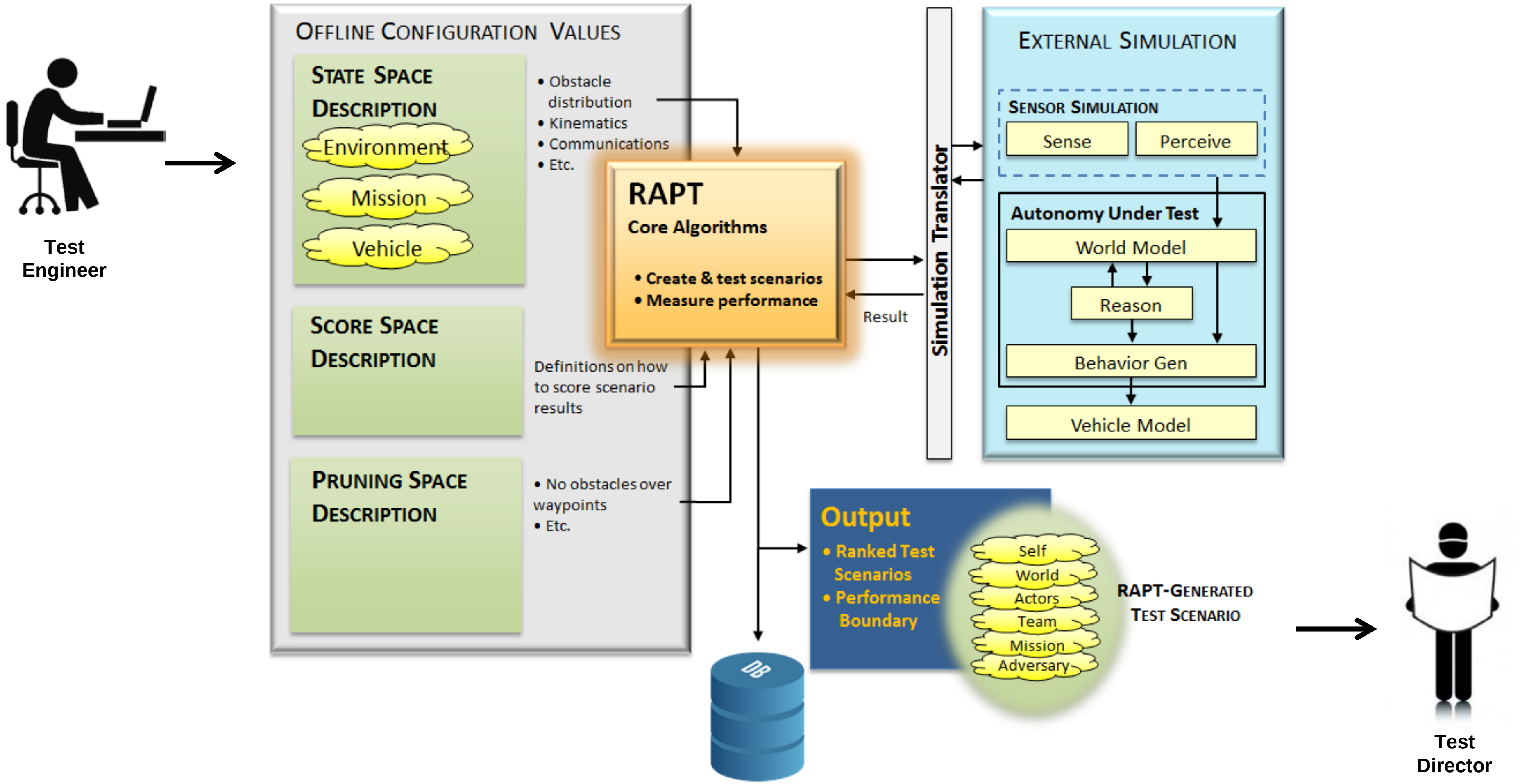


Plates2D Test Function



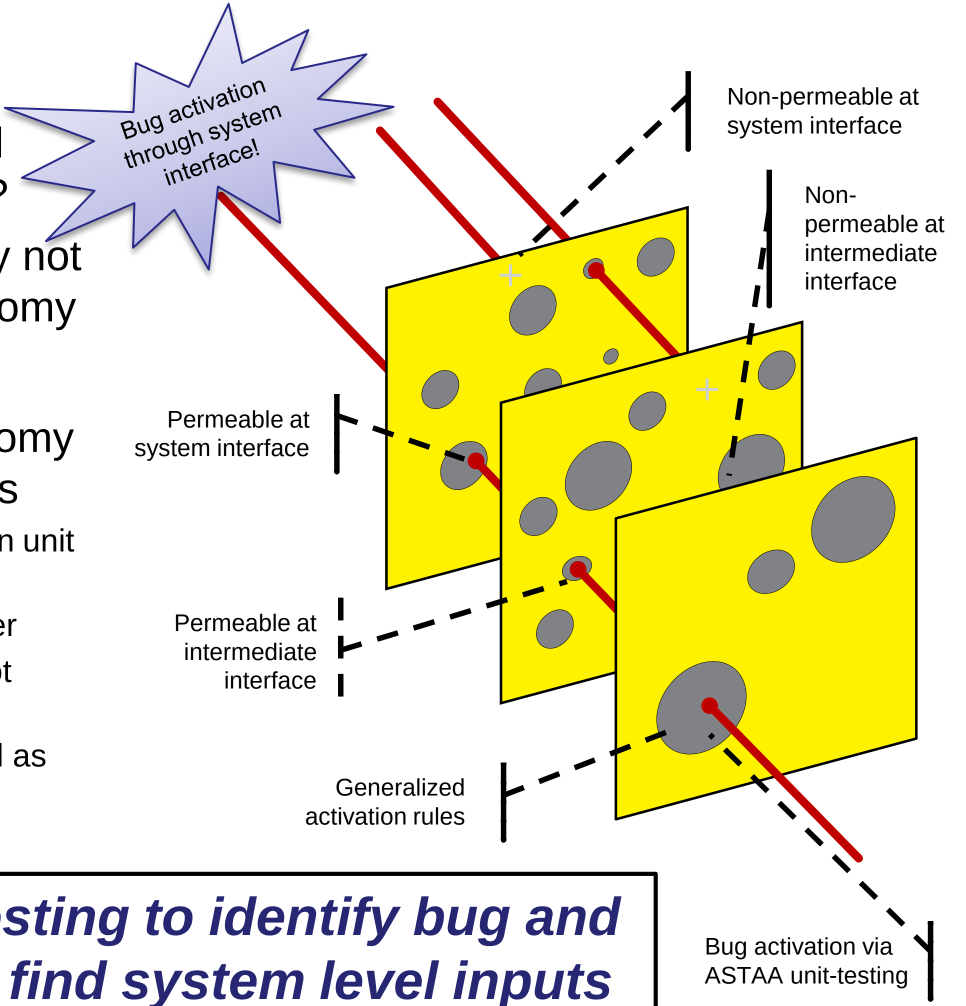


RAPT Architecture



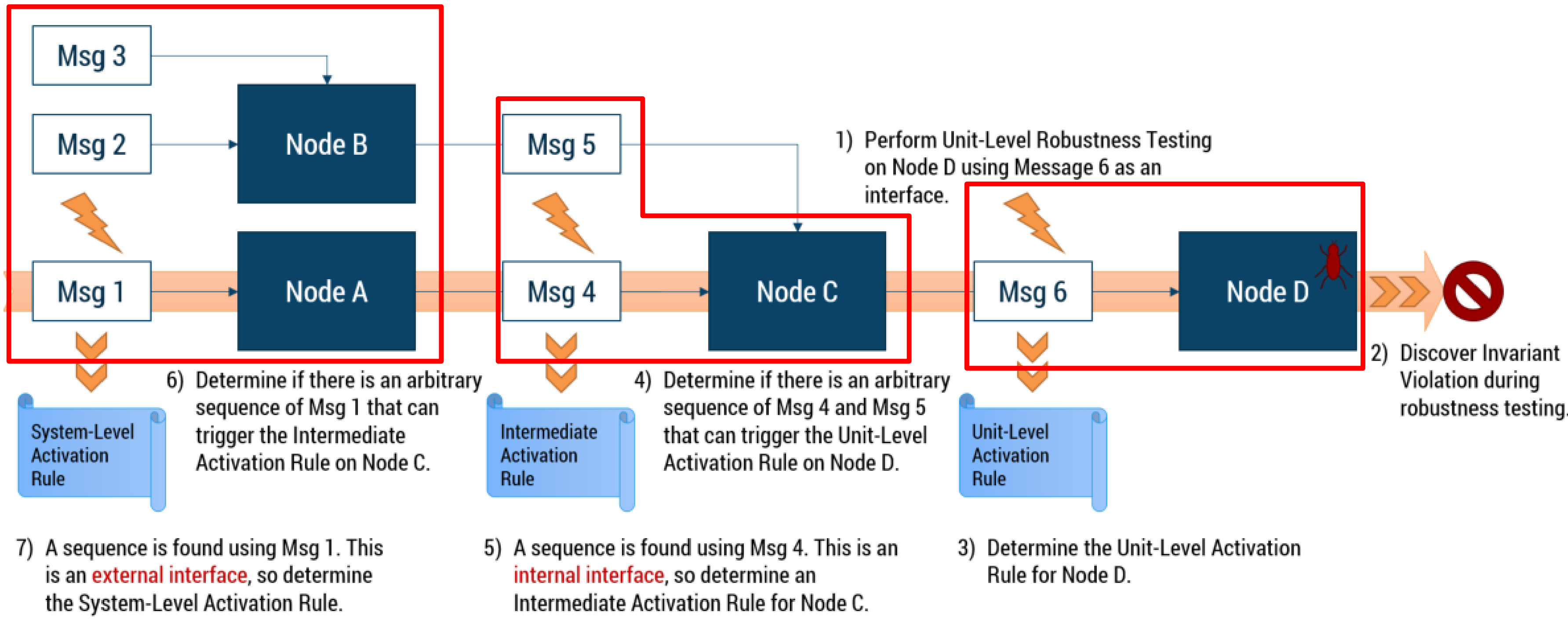
Robustness Inside-Out Testing (RIOT)

- **Autonomy Test Question:**
  - How does a tester identify unit level "bugs" that trigger unsafe behavior?
  - Full system level "Fuzz" testing may not identify Unit Level bugs in an autonomy
- **Proposal:**
  - Develop tool to find "bugs" in autonomy software that cause safety violations
    - ➢ Conduct Unit Level testing and back-chain unit bugs to system level inputs
    - ➢ Find system level bugs faster and cheaper
    - ➢ Find bugs that system level testing cannot
    - ➢ Identify bugs that cause typical software failures (Ex. Segmentation Faults) as well as safety failures (Ex. Max Speed Violation)



Conduct Unit Level testing to identify bug and back-chain results to find system level inputs

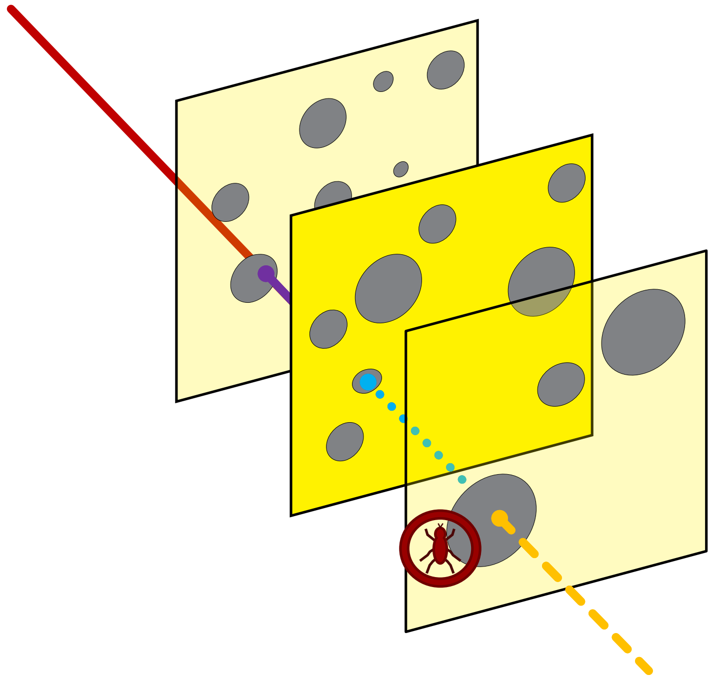
RIOT Process



8) The process delivers a specific test case that activates an internal bug from an external interface.

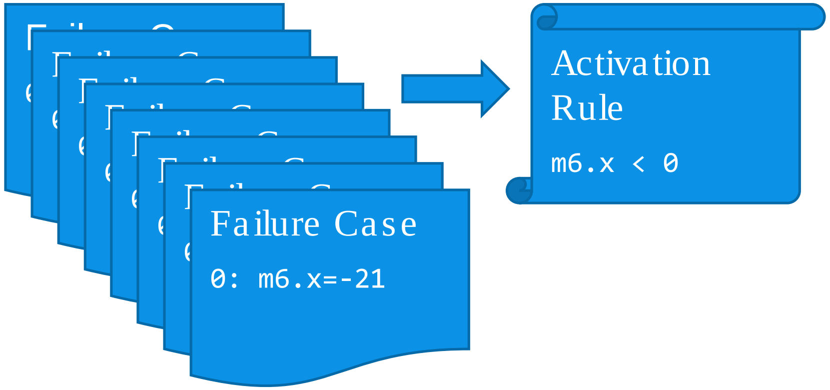
RIOT Challenges

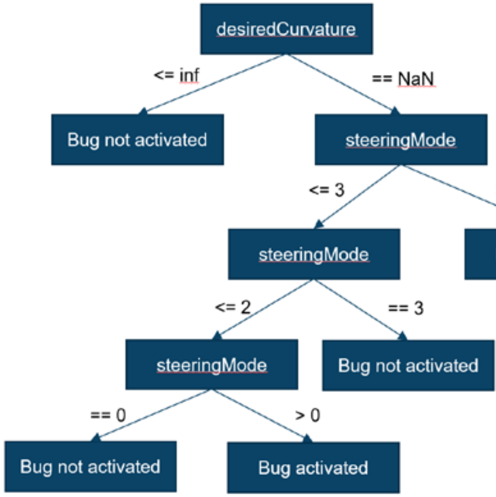
- **Typical Swiss cheese model fails because values do not simply pass through holes (i.e. interfaces)**
  - Values are transformed as they are processed by intermediate layers
    - Example: Motion planner receives goal.x = 5 and publishes cmd_vel.rpm = 3.068
  - Transfer functions are unknown with many-to-many mapping
  - Transformations are temporal and non-deterministic, even with identical experiments and inputs
- RIOT utilizes techniques for noisy costly black-boxes and implements them at the unit level
    - Black box testing is noisy and costly when testing a complex autonomous system
    - Identify bugs that cause typical software failures (Ex. Segmentation Faults) as well as safety failures (Ex. Max Speed Violation)



RIOT Generalization

- **Expand the target area**
  - Take a set of specific test values and infer the circumstances (e.g. range of values) under which the bug would be activated
- **Strategies for Generalization:**
  - Delta Debugging
    - Reduce the message log to improve efficiency of generalization
  - Decision Trees
    - Given large test field, find the “best” fields to split the reults
  - Omni-Trees
    - Evolution of decision trees to improve results
    - Ability to split on one or more field
  - Hierarchical Product Set Learning (HPSL)
    - Active learning strategy to infer values that caused an error based on initial error
  - Relationship Object Approximator for Domains
    - Augments HPSL by capturing and representing correlations between fields that cause an error



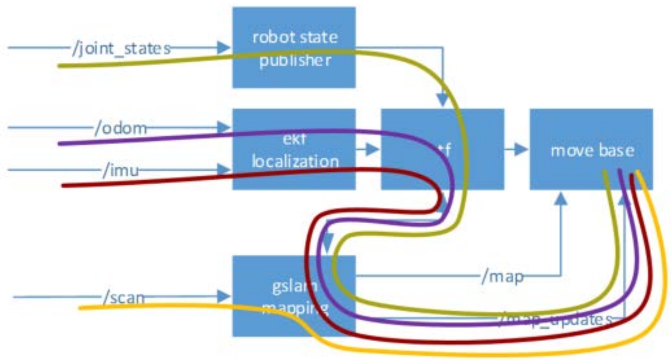


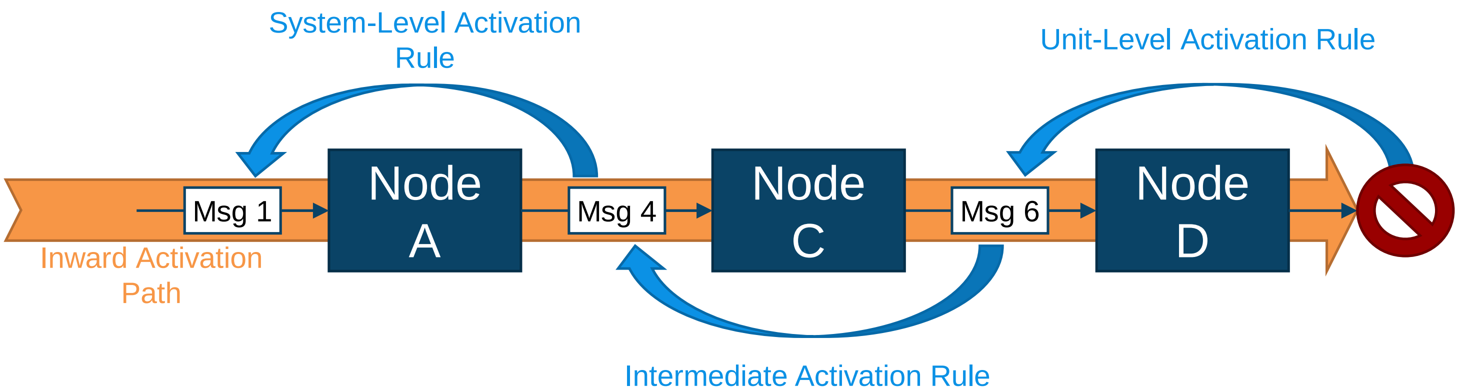
0	Manual
1	Drive
2	Reverse
3	Neutral

Decision Tree Example

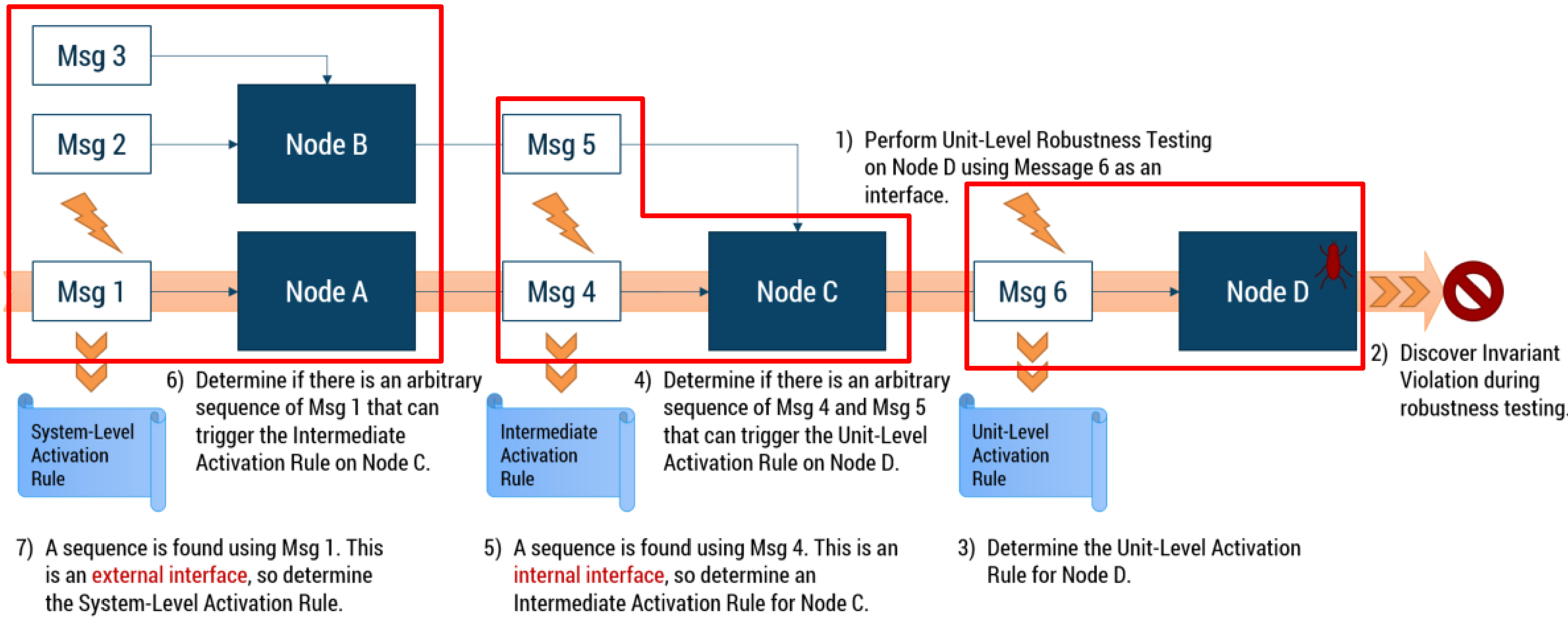
RIOT Back-Chaining

- **Determine messages that cause activation of the bug**
- **Strategies for Back-Chaining:**
  - Automate testing & exploration of message fields because search space is too large for a tester
  - Utilize multiple classification techniques to detect if an input message effects and output message
    - Time-series classification using a distance based classifier
    - Time-series classification using a feature based classifier
  - Do not need to determine the exact “transfer function”. We just need to determine if a previous message causes changes to the faulty message





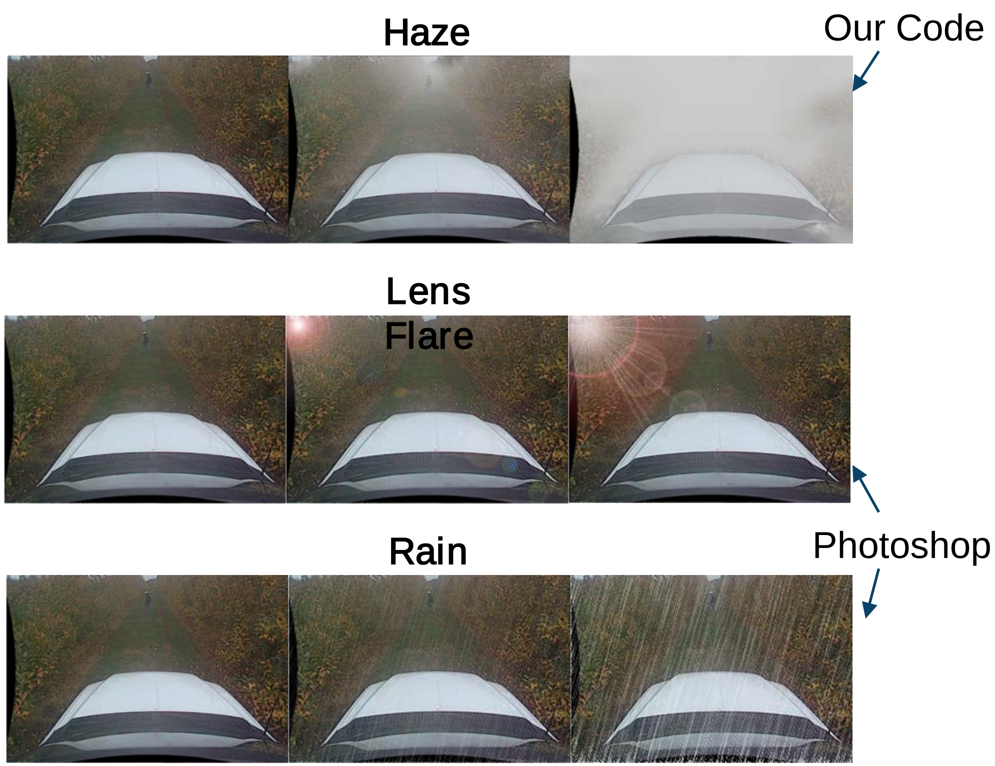
RIOT Process



8) The process delivers a specific test case that activates an internal bug from an external interface.

Robustness Testing for Perception Systems

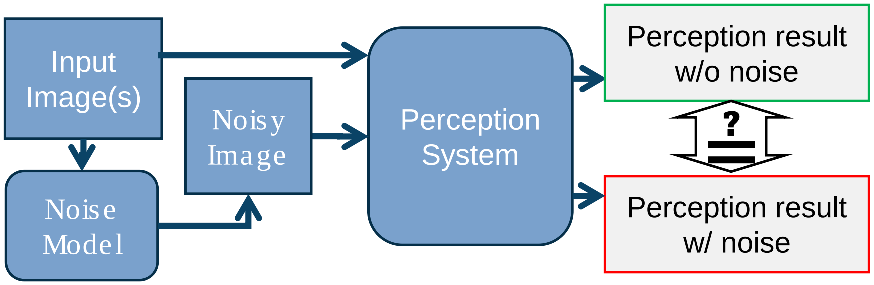
- **Autonomy Test Question:**
  - How does a tester determine the reliability of a perception system?
- **Proposal:**
  - Develop software to determine the "Robustness" of a Neural Network Perception System
    - Difficult to define "correct: for arbitrary images in perception systems
    - Instead, measure if the output is stable with the addition of noise
  - There are many stressful conditions that lead to noise
    - Environmental conditions (e.g. haze or fog)
    - Hardware effects (e.g. motion blur, focus)
    - Difficult scenes (eg. Occlusion of objects)



Behavior of an "ideally" robust system should be invariant to the addition of noise

Robustness Testing for Perception Systems

- **Testing the robustness of a perception system means checking that it's behavior is invariant under noise**
  - An input image is evaluated both with and without noise, and the results are compared
  - The results should be roughly equivalent for a perception system to be considered "robust"
    - NREC Agricultural Detection Benchmark was used
    - Used classifiers trained on data set as SUT
- **Example to the right, perception system fails to detect pedestrian with addition of blurring noise**
  - The ground-truth labeling is blue
  - Result of perception without noise is orange
  - Result of perception with noise is (not) shown in red

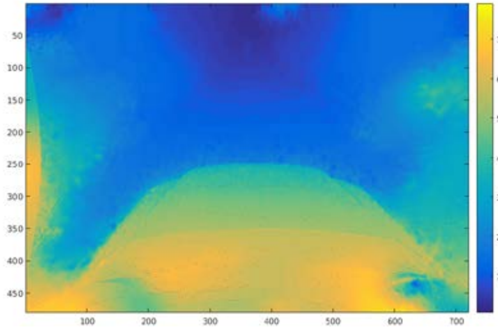


In this case, Gaussian blur noise made the pedestrian disappear (i.e., the red bounding box is missing).

Haze Mutator

- **We follow vision dehazing literature by using a simple alpha-blending approach w/ few parameters:**
  - Color of haze, c
  - Density of haze, b
  - Equation for each pixel
    - Visibility depends on depth at pixel, $\mathrm{z}(\mathrm{x})$
      $\mathrm{a} = e^{-\mathrm{bz}(x)}$
    - Hazed image is alpha blend w/ haze color
      $\mathrm{H}(x) = \mathrm{I}(x)\mathrm{a} + \mathrm{c}(1 - \mathrm{a})$
- **Our dataset has stereo images, so we can compute scene flow and filter to get smooth, dense estimates of depth (and motion) throughout scene**

Input Image



Estimated Disparities

Different levels of haze with simulated visibility distance, where $visibility = \frac{3.912}{b}$

1 km visibility

100 m visibility

30 m visibility

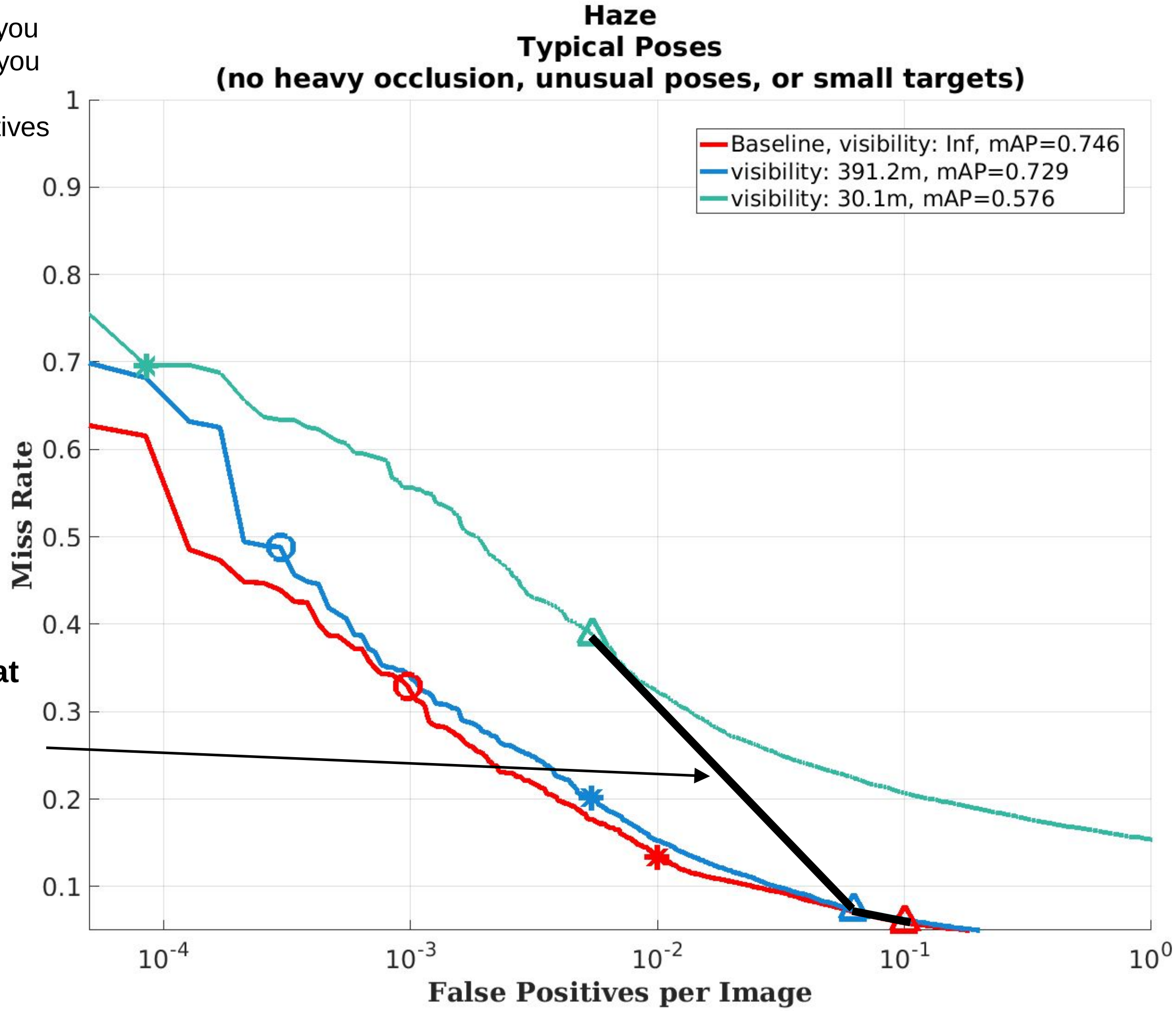
NAVAIR Public Release- 2018-715 'Approved for Public Release; distribution is unlimited'.

Perception System Performance with Haze

Shapes show how performance changes if you keep sensitivity fixed as you add haze. Sensitivity threshold with false positives per image in baseline conditions of…

- △ 1 per 10 images
- ✸ 1 per 100 images
- O 1 per 1000 images

Triangles show baseline system with sensitivity chosen that produces one false detection every 10 images and how it performs under different levels of haze.



Baseline assumes infinite visibility.

Note that both miss rate and false positive rate change as haze is introduced.

As a result, miss rate increases more than you might expect from just looking at the ROC curves.

Haze Visibility of ~400 m

Some strong detections become extremely weak with barely perceptible image changes.

Ground Truth Label | Detection | Detection Strength | False Positive Rate

Questions

???