Table of Contents

1 CAV Sensor Testing

1.1 Confidence of Collected Data

UWAFT took measures to ensure the data that was collected is valid. For each drive test that was executed, the start of data collection was signalled with someone standing as a definitive object in the visibility of the front radar. The person moved closer and further in a sine-wave-like manner, so that when viewing the logs, the start of the test can be easily identified by looking for this pattern at the start of the log. This was used to align the data sets from the Mobileye, Radar, and camera. Before conducting the deliverable drive tests, UWAFT executed many preliminary tests to check that the sensors were collecting valid data. During the test drives, a passenger was monitoring the CAN data from the Mobileye and radar to ensure it corresponded to the visible state of the road. Also, these preliminary tests were executed in different driving conditions and times of day to gain a better understanding of the operating conditions of the sensors. From this, we were better able to understand the operating conditions that affect the sensors and mitigate some issues in mounting. It was found the Mobileye reported more accurate data, as judged by the passenger, in daylight which is expected as the sensor is based on video perception. After executing the tests, the saved CAN logs from the Mobileye and radar were synced and compared with the corresponding dashcam videos.

With the dashcam video alongside a birds-eye-view of the sensor data, it was seen that the data is consistent, particularly, using regions of definitive change. For example, the region encompassing a car visibly entering and leaving the dashcam was used to validate that the Mobileye and radar data detected the car for the time it was there. Additionally, the approach test was used as a means of checking if the reported object distance from the radar was valid. In Figure 1 below, the radar's distance to the detected object can be seen to decrease over time, as expected for the approach test. However, the radar sometimes reports detected objects or objects with incorrect locations, as seen in Figure 2. Thus, the sensors need to be characterized in order to better understand their error. Another current issue is with our visualization tool. As objects become invalid a flag is triggered that can be used to clear these invalid objects. The team plans to update the visualization tool to remove these invalid objects in the future to allow for a cleaner visualization of our sensor data. The next steps are to gather statistics about the accuracy of the reported data, such as producing precision and recall for the sensor data using ground truth, as well as gaining an idea about the radar's actual detection field of view.

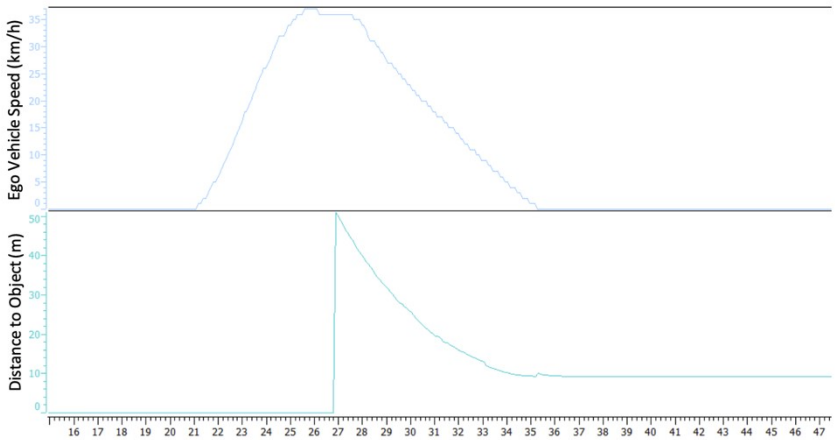

Figure 1 Front Radar Data for Approach Test

Figure 2 Birds-eye-view Of Detected Objects for Highway Drive Test

1.2 Mule Vehicle Installation

For the mule vehicle, UWAFT used a 2013 Lexus RX450 Hybrid that was obtained from the University of Waterloo's Centre for Automotive Research (WatCAR). On this vehicle the Bosch MRR14 forward-facing radar and Intel Mobileye Series 6 camera were mounted. Additional hardware that was used for the test was a Windows laptop with the CANoe software installed, two Vector CANcases, and a wire harness that contains three CAN lines (one each for Mobileye, radar, and vehicle CAN) as well as 12V power, fused at 2A, providing power through the vehicle's accessory power jack. The CANcases and the laptop were used to allow the incoming CAN messages from the sensors to be parsed and logged in CANoe. An additional webcam was also installed to capture the front view of the vehicle and the Windows Camera application was used to record the footage. The radar was mounted on a piece of polystyrene foam that was cut and shaped to allow for the team to place the radar mounting plate flush on the foam and attach the block to the front grill. Foam was chosen as it would dampen some vibration as well as protect the mule vehicle from damage. This mounting procedure ensured that the vertical misalignment of the radar was less than 0.5 degrees and horizontal misalignment was less than 1.5 degrees. The mounting position is skewed to the passenger side of the vehicle to more closely approximate where the sensor will be located on the Blazer. The team now has a standard operating procedure for how to setup and execute a mule vehicle test. Below in Figure 3 and Figure 4 the radar in its final position on the grill can be seen.

Figure 3 Front Radar Mounting Position Side View 1

Figure 4 Front Radar Mounting Position Side View 2

A wire harness was created for the radar that contained lines for power, ground, CANH and CANL. This harness was routed from the radar through the grill into the engine bay and out a gap between the hood of the vehicle and the windshield. It was then run along the side of the body through the window into the cabin where it could be connected to the CANcase. All wires that were exposed outside of the vehicle were also insulated. Below in Figure 5, Figure 6, and Figure 7 the routing that was done for the radar harness can be seen.

Figure 5 Front Radar Harness Routing View 1

Figure 6 Front Radar Harness Routing View 2

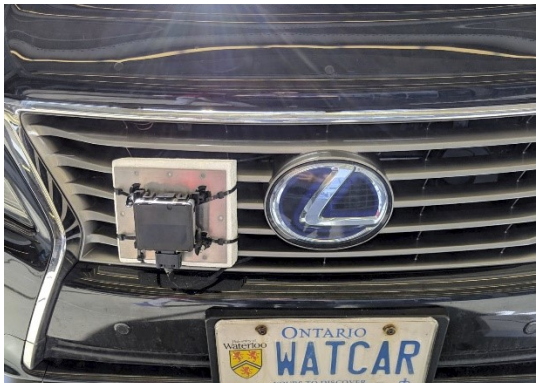

Figure 7 Front Radar Harness Routing View 3

On the inside of the vehicle the Mobileye was mounted to the windshield of the vehicle using double sided foam tape. The secondary webcam was mounted to the underside of the Mobileye, also using double sided foam tape, so that a video feed could be obtained from the Mobileye perspective. The camera wires were run through headliner and down the A-pillar trim to the passenger side. From here the Mobileye was connected to the vehicle's OBDII port and the CAN output was connected to the same

CANcase as the radar. Additionally, the vehicle CAN was connected to our second CANcase, using a splitter cable, to allow for vehicle information to be routed to the radar using CAPL scripts. The CANcases and the webcam were connected through USB to our laptop. Power for all the sensors was obtained from the vehicle accessory power jack. The camera mounting locations and wiring configuration can be found below in Figure 8 and Figure 9 respectively.

Figure 8 Camera Mounting Locations

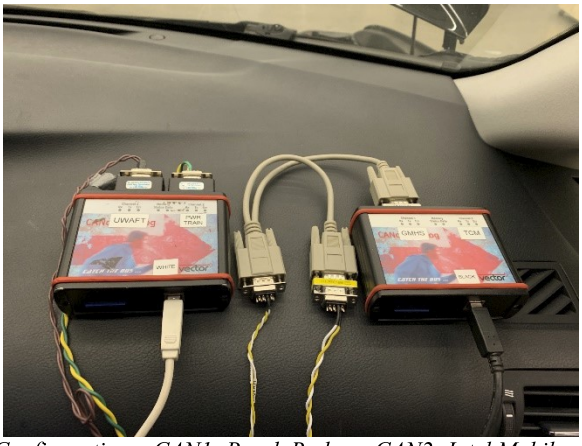

Figure 9 CANcase Connection Configuration – CAN1: Bosch Radar – CAN2: Intel Mobileye Camera – CAN3: Vehicle CAN

A schematic of the sensor wiring on the Lexus RX450H can be seen below in Figure 10.

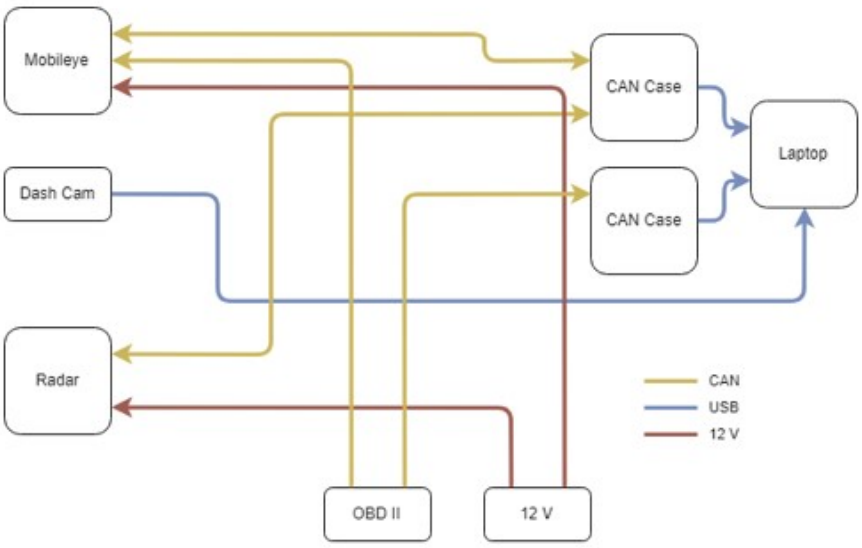

Figure 10 Sensor Wiring Schematic

2 Developing Requirements, Test Planning, and Software Architectures

2.1 Diagnostic Check Description and Implementation

The radar diagnostic check was implemented within a MATLAB class. The class was used as a means of organizing data and to accommodate multiple layers of functions required by the diagnostic check. Included in the functionality of the class is the ability to pass a log file upon calling the constructor and running the diagnostic check on a real set of data. This is in addition to a separate unit test function.

The design of the diagnostic check is intended to be modular in the sense that it is flexible with how many messages and how many attributes per message it can handle. This was accomplished by creating two levels of functions (see Figure 11), the first to check whether one variable indicates a fault, the second to return whether one message indicates a fault. The latter need only be passed an arbitrary number of messages of arbitrary size to run. With this design, the diagnostic check itself can be used for both the radar and Mobileye with very few adjustments. The diagnostic check will analyze a message by first calling the diagnose message function and loop through any given number of attributes depending on the size of the message. Each iteration, the diagnose message function will call the diagnose variable function which will return whether the given variable indicates a fault. The result need only be stored until the message is completely checked, at which point the diagnose message function will publish the results.

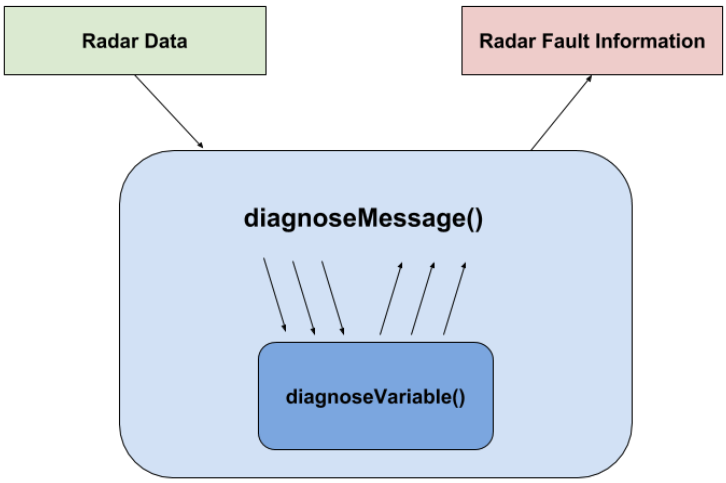

Figure 11 Diagnostic Check Function Calls

The radar diagnostic check was designed for practical implementation in software. That said, included in the output of the check is useful information with respect to the state of the radar at the time of the fault. The message data that contains the fault information along with when the fault occurred is propagated up from the innermost function and returned to the system (or simply printed for unit testing purposes). This is more useful for the user or engineer implementing the radar and associated software, as it is for the functionality of the CAVs software system.

An important advantage of creating a modular radar diagnostic check is its flexibility in future implementations. As the team comes to better understand the functionality of the radars and Mobileye, the attributes included in fault diagnostic may change. For this simple implementation and unit testing, four message components are analyzed for fault detection. Those are the radar timestamp, the hardware failure flag, the internal trouble code (ITC), and the functionality failure flag (SGU failure). These four components can indicate a fault of the radar. It was decided not to include object data, as these components are only capable of indicating a less valid detection as oppose to a faulted radar. Aspects such as less accurate data will be handled in sensor fusion where the validity of detections and probabilities will all be considered. That said, the modularity of the diagnostic check is crucial in allowing for flexibility in what messages are considered in the diagnostic check, the sensor fusion algorithm, a watchdog/heartbeat system or elsewhere in the perception system software.

2.2 Diagnostic Check Requirements

The diagnostic check is defined by three main requirements. Namely, messages must be sent at a given frequency, no hardware fault is present, and no SGU failure is present. Each requirement has corresponding message attributes that must be analyzed in the diagnostic check to determine more information regarding the state of the sensor. The three main requirements are what define the diagnostic check, but the diagnostic check will publish relevant data if a fault is detected. A detailed overview of the diagnostic check requirements, their descriptions, and correlated message attributes can be found in the Appendix in Table 1.

2.3 Unit Test Descriptions

The unit tests were designed to test every component of the diagnostic check and reach as many edge cases as possible, assuming the data being sent from the radar is valid. The inputs and expected outputs

are clearly defined in the Appendix in Table 2. The unit tests proved useful in developing a robust diagnostic check and will be maintained for further testing of the diagnostic check system if changes are made. After developing the unit tests for the diagnostic check, the tests caught a specific error in our tool where it would not trigger on a hardware fault. This allowed the team to go back and fix the issue. The tests will also allow the team to catch any regression issues that may arise when the diagnostic test is expanded to contain more errors and multiple sensors. Brief written descriptions of each test are also provided below. Note that when testing ITC values, many different possibilities were tested, but for the sake of the report a placeholder value of 999 was used in every case.

Unit Test 1 Definition

Designed to pass every message. The radar timestamps are set to their limit for 6 messages in a row at which point a fault would be generated if they were above the given threshold of 0.15. As the timestamps were at the limit and not over, the fault did not occur.

Unit Test 2 Definition

Designed to fail for radar timeout on messages 7, 8, 9, then recover, and timeout again at message 25. The diagnostic check is also passed 5 consecutive messages with timestamps greater than the threshold which should not trigger a fault, as a fault is only generated after 6 consecutive timestamp errors.

Unit Test 3 Definition

Designed to fail at messages 4 and 5 for an SGU failure with an ITC of “999”, then recover. Then fail again at messages 9 and 10 for an SGU failure with an ITC of “999” and recover once more.

Unit Test 4 Definition

Designed to fail at messages 3, 4, and 5 for hardware failure. Then at 6, 7, and 13 for both hardware failure and SGU failure with an ITC of “999”.

2.4 Diagnostic Results Software Propagation

When the diagnostic check outputs a non-zero value, indicating the sensor in question is faulted, the fault should propagate through the system and disable or limit the capabilities of related CAVs controllers. It is crucial that the signal propagation be simple and robust as the generated radar faults are safety critical. Depending on the fault, it may also be possible for the sensor to recover in which case the fault message will no longer be generated, and impacted controllers can be re-enabled.

Although detailed software architecture has not yet been designed, the sensor diagnostic tool will consist of its own node and publish generated faults to an associated topic. Any relevant nodes on the Tank would be able to access the fault data from this topic, namely, the supervisor and sensor fusion nodes. As the most crucial system response of any given fault is the deactivation of related controllers (ACC, Lane Change, Lane Keep, AEB), the fault information must also propagate to the Hybrid Supervisory Controller (HSC) where the AEB controller will be executed. However, the fault message itself will not be sent to the HSC, but handled in the Tank, which would generate an appropriate signal to disable the AEB algorithm (or any other controller) residing on the HSC. The flow of data can be seen in Figure 12, but for the sake of simplicity and clarity, the HSC itself is not shown; it is understood that any messages being sent from the supervisor to a control algorithm on the HSC would simply be over CAN as opposed to the publisher-subscriber system of ROS. A similar explanation can be given for the propagation of the fault signal to the Jetson TX2 (this time shown in Figure 12). The Jetson TX2, the computing system

that will run the HMI, will need to receive any sensor fault information in order to notify the driver. But as with the controllers, the fault messages will be handled on the Tank, and any relevant messages will be formatted appropriately and relayed to the Jetson for notification of the driver. Both the HSC and Jetson TX2 will be connected to the Tank directly though CAN so any communication between the two will be of minimal cost.

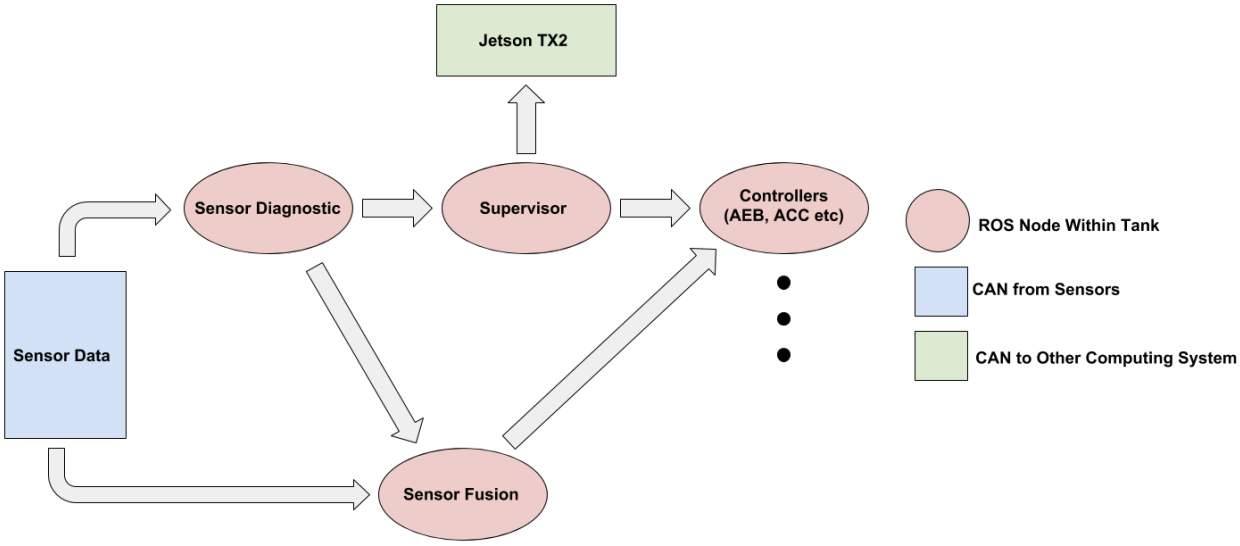

Figure 12 High Level Software Architecture

The signal propagation path will differ based on sensors as different sensors have varying levels of importance for different controllers. The driver notification however will remain the same regardless of which sensor caused the fault.

If the front radar fails, the fault signal will be sent to the Tank, which in turn sends a corresponding message to the HSC. The failure of the front radar is the most serious as it would severely impact the forward-facing field of view of the perception system. Adaptive Cruise Control, Lane Change and Automatic Emergency Braking will all be deactivated as the front radar is critical in the operation of all three. The deactivation of AEB requires that the signal, in some form, be propagated from the Tank to the HSC where the AEB controller is planned to reside. The HSC will then receive the deactivation command and disable the AEB controller. In this scenario, three out of the four controllers will be deactivated.

A faulted front-corner or rear radar has less effect on the CAVs controllers. The fault signal is only propagated throughout software within the tank to the Lane Change controller. If the left front-corner radar fails, only lane change to the left would be disabled due to the lack of information of objects on that side. The same goes to the right front-corner radar. However, if it is the rear radar that malfunctions, lane change to either side would be disabled because the vehicle would no longer receive object data in its rear field of view.

As previously mentioned, regardless of the failure, the tank would also send a message to Jetson TX2 indicating which sensor failed and which CAVs controller(s) would be shut down. After receiving this information, the Jetson would immediately prompt the driver to regain control by visually displaying a

message on the instrument panel, vibrating the driver's seat and an audio signal(beeping). Also, any relevant information will be displayed to the driver.

2.5 Future Testing with Hardware Signals

The diagnostic check will be further evaluated using hardware once it is implemented into the CAV system. This could be done at multiple levels, the most comprehensive of which is to have a radar mounted to the outside of a mule vehicle and take the vehicle for a drive to provide real sensor data in real time. The CAN data (sensor outputs) can be logged alongside the results of the diagnostic check for a very comprehensive test of a very large set of data. The data can be thoroughly reviewed, and the performance of the diagnostic check can be accurately assessed.

An easier and simpler test of the diagnostic check is to run tests on the bench. Once the Tank is running the diagnostic check algorithm, CAN messages can be forced to the Tank to simulate radar messages. In this scenario, any CAN message can be produced, including the most obscure and rare faults. Although less data would be run through the diagnostic check, this would allow for very specific CAN messages to be run through the check allowing us to cover any extreme edge cases that may arise.

The bench tests will be executed both in a fully software environment and with the dSPACE hardware-in-the-loop (HIL) simulator. In HIL testing, the Tank will be connected to the HIL with all the appropriate CAN lines to simulate the sensors. Errors will be introduced in the CAN data to test specific cases of the diagnostic checks. The HIL testing will be able to provide the team with additional confidence that the system can perform correctly in real-time situations.

The team also plans to automatically execute unit and integration tests on the source code through a continuous integration (CI) system. This system will execute the suite of tests to ensure these diagnostic safety checks are functionally correct and does not contain any regression of bugs. The team also plans to expand the automated test system to include automated HIL bench testing where faults can be injected, and the system can be tested in real-time. UWAFT has created an additional Developer Operations sub team to handle the creation of this test platform for both CAVs and CSMS and significant progress has been made to establish this infrastructure already.

3 Appendix

3.1 Diagnostic Check Requirements

Table 1 Diagnostic Check Requirements

Requirement	Description	Correlated Message Attributes	Fault Publishing
Radar is sending message at an acceptable frequency	The radar timestamp is of reasonable value. The radar timestamp is not allowed to continuously exceed a certain threshold.	• Hardware failure flag will be examined in case of correlation. • SGU failure flag will be examined in case of correlation. • If SGU failure flag is set, ITC must be examined to determine cause. • The time of the fault in seconds must be checked and published.	• If no other failure is found, do not immediately publish a fault. If for 6 consecutive messages the timestamp remains above a certain threshold, publish a fault as sensor data is no longer valid. • If another fault is found, publish fault according to its standards (this will happen automatically).
Radar hardware failure flag is not set	The hardware failure flag must not be set true.	• Hardware must be verified. No software implementation. • The time of the fault must be checked and published.	• Immediately publish returned fault message, radar is no longer sending objects.
No radar SGU failure	The SGU failure flag must not be set true	• ITC attribute must be examined to determine cause of SGU failure. • The time of the fault must be checked and published.	• Immediately publish returned fault message, radar is no longer sending objects. Wait for possible recovery or for fault to be handled.

3.2 Unit Test Definitions

Table 2 Unit Test Definitions

Test #	Input Description	Input	Expected Output	Actual Output	Pass / Fail
1	- Radar timestamps are set at their limit for messages 7 to 12.	0.1434 0 0 0; 0.0601 0 0 0; 0.1212 0 0 0; 0.0579 0 0 0; 0.0633 0 0 0; 0.0633 0 0 0; 0.1500 0 0 0; 0.1500 0 0 0; 0.1500 0 0 0; 0.1500 0 0 0; 0.1500 0 0 0; 0.1500 0 0 0; 0.0522 0 0 0; 0.0826 0 0 0; 0.0148 0 0 0;	No output	No output	Pass

2	- Radar timestamps are set above limit for messages 2 to 9, 13 to 17, and 20 to 25.	0.0901 0 0 0;
		2.7345 0 0 0;
		2.295 0 0 0;
		1.5082 0 0 0;
		2.6527 0 0 0;
		2.2484 0 0 0;
		2.8739 0 0 0;
		2.7436 0 0 0;
		2.5975 0 0 0;
		0.0472 0 0 0;
		0.0324 0 0 0;
			- Radar timeout error for messages 7 to 9 and 25	- Radar timeout error for messages 7 to 9 and 25.
				`Time: 7 error: radar_timeout 2.87e+00`
				`Time: 8 error: radar_timeout 2.74e+00`
				`Time: 9 error: radar_timeout 2.60e+00`
				`Time: 25 error: radar_timeout 2.50e+00`
					Pass

		0.0743 0 0 0; 2.5985 0 0 0; 2.6583 0 0 0; 2.6853 0 0 0; 2.2457 0 0 0; 2.2465 0 0 0; 0.0532 0 0 0; 0.0324 0 0 0; 3.0012 0 0 0; 2.5482 0 0 0; 2.8254 0 0 0; 2.8535 0 0 0; 2.3568 0 0 0; 2.4969 0 0 0; 0.0467 0 0 0; 0.0362 0 0 0;			

3	- Messages 4 and 5 are passed an SGU failure with ITC of "999".
	- Messages 9 and 10 are passed an SGU failure with ITC of "999".
		0.0325 0 0 0;
		0.0409 0 0 0;
		0.0846 0 0 0;
		0.0816 0 999 1;
		0.0433 0 999 1;
		0.0383 0 0 0;
		0.0239 0 0 0;
		0.0294 0 0 0;
		0.0702 0 999 1;
		0.0379 0 999 1;
		0.1001 0 0 0;
		0.0374 0 0 0;
			- Radar SGU failure with ITC "999" for messages 4 and 5.
			Radar SGU failure with ITC "999" for messages 9 and 10.
				- Radar SGU failure with ITC "999" for messages 4 and 5.
				Radar SGU failure with ITC "999" for messages 9 and 10.
				`Time: 4 error: internal_trouble_code 999`
				`Time: 4 error: functionality_failure 01`
				`Time: 5 error: internal_trouble_code 999`
				`Time: 5 error: functionality_failure 01`
				`Time: 9 error: internal_trouble_code 999`
				`Time: 9 error: functionality_failure 01`
				`Time: 10 error: internal_trouble_code 999`
				`Time: 10 error: functionality_failure 01`
					Pass

4	- Messages 3 to 7 passed a hardware fault.	0.0429 0 0 0; 0.0418 0 0 0; 0.0921 1 0 0; 0.0436 1 0 0;	- Hardware failure from messages 3 to 5. - Hardware and SGU failure with ITC of "999"	- Hardware failure from messages 3 to 5. - Hardware and SGU failure with ITC of "999" from messages 6, 7, and 13.	Pass

	- Message 6, 7, and 13 passed SGU failure with ITC of “999”.	0.0653 1 0 0; 0.0206 1 999 1; 0.0120 1 999 1; 0.0136 0 0 0; 0.0323 0 0 0; 0.0521 0 0 0; 0.0221 0 0 0; 0.0360 0 0 0; 0.0412 1 999 1;	from messages 6, 7, and 13.	Time: 3 error: hardware_failure 01 Time: 4 error: hardware_failure 01 Time: 5 error: hardware_failure 01 Time: 6 error: hardware_failure 01 Time: 6 error: internal_trouble_code 999 Time: 6 error: functionality_failure 01 Time: 7 error: hardware_failure 01 Time: 7 error: internal_trouble_code 999 Time: 7 error: functionality_failure 01 Time: 13 error: hardware_failure 01 Time: 13 error: internal_trouble_code 999 Time: 13 error: functionality_failure 01	