Formal verification of the Autosub Autonomous Underwater Vehicle: A Case Study

Levente Molnar and Jonathan Ezekiel, Sandor M Veres, Alessio R Lomuscio, Miles Pebody

I. INTRODUCTION

The present document is a case study describing the formal verification of the Autosub Autonomous Underwater Vehicle (AUV). Section II provides a description of the Autosub AUV multi-agent system model. Section III details the specifications that were formulated to verify various requirements and properties of the Autosub model. Appendix I presents the Statemate diagrams containing the labelled transition system formulation of the multi-agent model. Appendix II provides a listing of the Autosub multi-agent model formulated in the interpreted system programming language (ISPL) input language of the MCMAS model checker for multi-agent systems. Appendix III details the output listing of the verification property results as provided by the MCMAS model checker.

II. DESCRIPTION OF THE MULTI-AGENT SYSTEM ISPL MODEL

The Human ISPL agent illustrates the functionality of the HTSC human tracking and supervisory control. After vehicle launch and prior to safe recovery, the human operator from the support ship will closely observe the AUV during its operation. When the AUV is within range of the ship, short range telemetry is used to monitor the vehicle systems and transmit surfacing commands. During the mission the vehicle can exceed the short range or acoustic telemetry range such that no information of the vehicle systems or location is available. If the vehicle does not return to the support ship or to a predetermined rendezvous location for recovery as that would be determined by the mission control program and mission configuration parameters during normal/correct operation, an acoustic beacon is lowered from the support ship to trigger the vehicle’s onboard homing system. The Human agent implements this model behaviour, such that while the human observes the correct AUV operation it does not take any action, but when the AUV appears faulty the Human agent triggers the homing system with lowering the homing beacon.

The MissionScriptExecution ISPL agent models the state machine functionality of the mission control process that is being executed in the Mission Control node [1], [2], [3]. The mission control process consists of two major components, that is the event processing and the mission control state machine. The mission control state machine executes mission element instructions that are triggered by mission events. When a mission event occurs, mode and demand instructions enclosed in a mission element are sent to the various control, actuator and sensor payload systems. After the mode and demand instructions from the current mission element are issued, the execution of the mission state machine iterates to the next mission elements and waits until the next specified mission event occurs. Two alternative mission ending scenarios are provided by the “mission termination exceptions”. The triggering events for the execution of the “mission termination exceptions” are defined in the mission control configuration. The “homing underway” state is another mission element modelled as a local state for the MissionScriptExecution ISPL agent. The triggering event for the execution of the “homing underway” mission element is the receiving of the homing signal by the vehicle’s onboard homing system. During the “homing underway” mission element the vehicle is controlled to move towards the support ship’s position at a predetermined depth and speed as specified in the mission control configuration. The mission script included mission element local states indicate the correct behaviour of the agent, while the “mission termination exceptions” and “homing underway” mission elements are the local states representing the non-correct behaviour of the agent.

The role of the agents PositionControlInstructionParser, DepthControlInstructionParser, MotorControlInstructionParser, MissionTimerInstructionParser and LineTimerInstructionParser is to parse the ISPL actions of joint mode and demand instructions and timer setup, performed at mission elements, into corresponding individual actions for the each actuator control node and the line timer.

The MissionControlInternalEvents ISPL agent is responsible for producing the missions script execution triggering events that relate to the current depth status of the vehicle. Depth status information such as “vehicle is on surface”, “vehicle has dived”, “vehicle reached depth demand”, or “vehicle is too deep”, will serve as the basis of the triggering events that will determine the evolution of the MissionScriptExecution agent, amongst the mission elements illustrating correct mission execution or non-correct recovery orientated behaviours. The MissionControlInternalEvents agent compares the current depth information of the vehicle, against depth threshold
The evolution of the MissionScriptExecution actions that can be taken in this state consists of a 180 degree turn and retreat along the same path the vehicle had come along, to a preset safe retreat distance. When the retreat distance is covered, the “trying new track” state is entered, during which the vehicle will try a new track, parallel to the original. If the vehicle encounters an obstacle, it will switch back to “retreating” state. If the vehicle has covered a clearance distance on the alternative path, the obstacle avoidance process is finished, the “trying new track” state is exited and the “waiting” state is entered. The avoidance process for control of the horizontal control dimension is built on top of the position control operation that uses waypoint guidance principles to head towards a target position or to carry out track following.

The SafePositionMeter agent illustrates the operation of a self-resetting distance meter. During the vehicle’s travel the SafePositionMeter adds the travelled distance segment to the safe position meter’s accumulated distance [5]. When the accumulated distance reaches a preconfigured value for track update, the SafePositionMeter agent triggers the RetreatPathArray to store the vehicle’s current position into a circular waypoint array. Following this, the safe position meter resets itself and will continue accumulating the travelled distance until the preconfigured value is reached again.

The RetreatPathArray agent models the operations performed with the circular waypoint array. During the “waiting” and “trying new track” states, when triggered by the SafePositionMeter agent, the retreat path array is used to store at regular distance intervals the current position of the vehicle [5]. During the “retreating” state of the HorizontalAvoidanceStrategy the waypoints stored in the retreat path array are sent as demands to the position control, in order to navigate the AUV back from the detected obstacle to a preset safe retreat distance where a new manoeuvre and alternative obstacle avoidance route can be started.

The SafePositionMeter agent performs monitoring of the distance travelled by the vehicle during the “retreating” state of the HorizontalAvoidanceStrategy. During the “retreating” state a predetermined number of waypoint are retrieved from the retreat path array and sent as demands to the position control [5]. A safe position, marking the end of the retreat distance is reached, when the vehicle arrives to the last retreat path array retrieved waypoint from the predetermined number of retreat waypoints. At this point the RetreatPathArray agent will trigger a “collision avoided” event for the MissionScriptExecution agent.

The AvoidanceTimer agent monitors the length of time since the “retreating” state was first entered and hence the duration of the avoidance process since it became active. If a predetermined timeout value is exceeded, the AvoidanceTimer will generate an “avoidance failed” triggering event for the mission state machine formulated by the MissionScriptExecution agent. The RetryTrackCounter agent performs the counting of the number of times the “retreating” state is entered by the horizontal avoidance process. If the accumulated number of retry track count exceeds the allowed number of counts, the RetryTrackCounter will generate an “avoidance failed” event in order to trig-

Fig. 1. ISPL agents of the Autosub AUV model

Reference values defined in the mission control configuration.

The MissionTimer agent guards over the mission duration and is intended to provide a trigger to the EmergencyAbort node for mission termination. The timer agent is enabled at the launch of the vehicle and is scheduled to expire at a maximum mission duration length specified in the mission control configuration. If the mission has continued for too long, a mission timer timeout occurs and the EmergencyAbortSystem is triggered.

The LineTimer agent can provide a trigger to determine the evolution of the MissionScriptExecution agent. After issuing a mission element’s mode and demand instructions, the mission state machine iterates to the next mission element and waits for a triggering event to occur. A line timer timeout can become a triggering event for the execution of mission state machine mission elements.

The HorizontalAvoidanceStrategy agent models the multi-state process developed for the collision and obstacle avoidance for the horizontal control dimension [4]. “Waiting” is the active state during normal operation of the AUV, when the collision and obstacle avoidance is not taking any action. When triggered by “limited headroom” or “obstacle ahead” events, the state of the avoidance process switches to “retreating” during which the vehicle will backtrack. The actions that can be taken in this state consists of a 180 degree
ger mission alternative endings for the mission state machine [5]. An “avoidance failed” event will determine the MissionScriptExecution agent to evolve to a mission element belonging to a mission termination exception. Hence an alternative ending to the mission is triggered that will address the non-correct and faulty behaviour of the avoidance process. Both, the AvoidanceTimer and the RetryTrackCounter implement the functionalities of safeguards, against the vehicle becoming stuck in the avoidance process.

The ObstacleClearanceMeter agent performs monitoring of the distance travelled by the vehicle while the HorizontalAvoidanceStrategy is in the “trying new track” state. The travelled distance segments are added to the obstacle clearance meter’s accumulated distance. When the vehicle has covered the clearance distance on the alternative path, that is, when the accumulated distance reaches a pre-configured value, the obstacle avoidance process is finished and the MissionScriptExecution is signalled by triggering a “collision avoided” event for the mission state machine [5].

The NewDemandDetection agent determines if there is any new demand issued for the position control by the Mission Control node’s mission state machine, while the HorizontalAvoidanceStrategy is in the “retreating” state. At the end of the “retreating” state when the safe retreat position is reached, the RetreatTrackLengthMonitor agent will trigger the “collision avoided” event for the MissionScriptExecution [5]. The mission state machine can intervene at this stage by issuing a new demand to the position controller. The NewDemandDetection will sense the new position demand and will determine the HorizontalAvoidanceStrategy to abandon the avoidance strategy, but only after the horizontal avoidance has entered the “trying new track” state.

The VerticalAvoidanceStrategy agent addresses the operation of the avoidance strategy for the vertical control dimension [4]. The vertical avoidance strategy does not adopt a multi-state avoidance process, as taken for the implementation of the horizontal avoidance. Instead a continuous control solution is chosen, by imposing stricter control limits to the allowed range of the depth control process variables of depth and altitude. During normal operation, the “minimum depth”, “maximum depth”, “minimum altitude” and “minimum ice clearance” control limits are configured to their normal value, while in case of detected “limited headroom” or “obstacle ahead” the normal values are changed to a set of safe control limit values. After the obstacle is avoided the safe control limits will revert to their normal values.

The PositionControlOperation agent details the vehicle position control phases. The position control relies on waypoint guidance principles for travelling towards a target position or to carry out track following between a destination and target position [4]. Distinct phases in the vehicle position control are illustrated by the “head towards demand target position”, “arriving to acceptance circle region of target position” and “at demand target position” local states. The vehicle departs from its current position when a new demand is issued by the mission state machine, or a waypoint is retrieved from the retreat path array and issued as demand during backtracking. When the vehicle reaches the demand position a “got position” event is issued. At this point the mission state machine can issue new demands formulated in the subsequent mission element, or if in “retreating” the retreat path array can provide the following retreat waypoint. After the target position is reached, the vehicle will continue loitering until a new position demand is issued to the position control.

The DepthControlOperation agent addresses the various phases of the control process for the vertical dimension. The condition of the vehicle’s vertical position in the water column is described, from the perspective of the “range to the ice overhead” and “range to the seabed” variables status, relative to the “minimum ice clearance” and “minimum altitude” configuration limits. If both upward and downward measured ranges are greater than their configuration limits, the vehicle will be at a safe distance from the water column boundaries and will be using the mission state machine issued depth or altitude control demands. If the vehicle is located too close to overhead ice or seabed features, the “minimum ice clearance” or the “minimum altitude” configuration limits will override all depth and altitude based control actions. A hazardous confined space is indicated when the vehicle is too close to overhead ice and seabed features, in the same time. In this situation the DepthControlOperation will send a “limited headroom” event to the HorizontalAvoidanceStrategy and VerticalAvoidanceStrategy agents, determining the vehicle to backtrack along the same path that it had come along, and to use safe configuration limits for its vertical control actions.

The role of the GpsNavigation agent is to produce a “gps fix acquired” event for the mission state machine, when a calculated position information is available.

The ForwardLookingEchosounder agent generates the “obstacle ahead” event for the HorizontalAvoidanceStrategy and VerticalAvoidanceStrategy agents, when an obstacle is detected on the vehicle’s path.

The Environment agent indicates the hazardous physical environment that the vehicle operates in. Due to possible close proximity operation to seabed, ice overhead and other hazards, the vehicle can become trapped, or can get damaged and have its controllability affected to a varying extent. In these situations the vehicle may not be able to return to the support ship or to a predetermined rendezvous location for recovery. A delayed return of the vehicle may determine the Human agent to consider that the vehicle is in a faulty operational state.

The PowerNode agent indicates the status of the onboard battery power consumption. If the battery voltage falls below a preconfigured set level, the power node will notify the mission control node through a “power low” event. At the occurrence of this event, the mission state machine will determine an alternative ending to the mission.

The EmergencyAbortSystem agent models the functionality of the Emergency Abort node, illustrating its supporting role for the safe operation of the vehicle. The
local states of the EmergencyAbortSystem agent include the “normal operation” state for correct behaviour and the “mission aborted” state to illustrate the faulty behaviour. The trigger for the agent to evolve from “normal operation” to the “mission aborted” state, are the signalling by the MissionTimer agent that the “mission has continued for too long”, or if the MissionControlInternalEvents agent notifies the EmergencyAbortSystem that the “vehicle is too deep”. The HomingSystem agent model includes the “idling”, “homing signal detected” and “homing signal lost” operational states of the vehicle’s onboard homing system. The homing signal is received when an acoustic beacon is lowered from the support ship to trigger the vehicle’s onboard homing system. In this case the “homing signal detected” event is generated to the Mission Control node, where the mission state machine will trigger the “homing underway” mission element. While the homing signal is detected, the homing behaviour will guide the vehicle towards the ship’s position. If the signal is lost by the onboard homing system, this is also signalled to the mission control node. The “homing signal lost” event will trigger a “mission termination exception” in the mission state machine’s execution, providing an alternative ending to the mission.

A. Faults

The VehicleAppearsFaulty fault injection agent represents the faulty behaviour in which a human observes physically

```
VehicleAppearsFaulty → Environment
VehicleTooDeep → MissionControlInternalEvents
HomingLostBeforeAcquired → HomingSystem
HomingLostAfterAcquired → HomingSystem
PowerLow → PowerNode
MissionTooLong → MissionTimer
AvoidanceTimerFailed → AvoidanceTimer
RetryTrackAvoidanceFail → RetryTrackCounter
RetreatPathArrayAvoidanceFail → RetreatPathArray
HorizontalAvoidanceFailure → HorizontalAvoidanceStrategy
EchosounderDetects → ForwardLookingEchosounder
MinAltRange → DepthControlOperation
MinIceRange → DepthControlOperation
```

**Fig. 2.** Fault injection agents altering the evolution of the target agents from correct behaviour to non-correct or faulty behaviour

```
Table I: Definition of the fault injection agents

<table>
<thead>
<tr>
<th>Name</th>
<th>Agent</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleAppearsFaulty</td>
<td>Agent</td>
<td>Environment</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>Until</td>
<td>A “homing” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>VehicleTooDeep</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The mission script has started</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “vehicle is too deep” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>HomingLostBeforeAcquired</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace (on, off)</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>Until</td>
<td>A “homing lost” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>HomingLostAfterAcquired</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace (on, off)</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>None</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>A “homing acquired” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>The fault has occurred</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>PowerLow</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace (greater than, less than)</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The mission script has started</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “mission too long” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>MissionTooLong</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The mission script has started</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “mission too long” signal is sent</td>
<td></td>
</tr>
</tbody>
</table>

**Name** | **Abbrv** | **Variable** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EchosounderDetects</td>
<td>Agent</td>
<td>ForwardLookingEchosounder</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The obstacle avoidance process has begun idling or is maintaining a retreating state</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>An “obstacle ahead” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>MinIceRange</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace Greater than, less than</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The obstacle avoidance process has begun idling or is maintaining a retreating state</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “limited headroom” signal is sent</td>
<td></td>
</tr>
</tbody>
</table>

**Name** | **Abbrv** | **Variable** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MinAltRange</td>
<td>Agent</td>
<td>DepthControlOperation</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace Greater than, less than</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The obstacle avoidance process has begun idling or is maintaining a retreating state</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “limited headroom” signal is sent</td>
<td></td>
</tr>
</tbody>
</table>

**Definition of the fault injection agents**

<table>
<thead>
<tr>
<th>Name</th>
<th>Agent</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehicleAppearsFaulty</td>
<td>Agent</td>
<td>Environment</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>Until</td>
<td>A “homing” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>VehicleTooDeep</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The mission script has started</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “vehicle is too deep” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>HomingLostBeforeAcquired</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace (on, off)</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>Until</td>
<td>A “homing lost” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>HomingLostAfterAcquired</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace (on, off)</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>None</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>A “homing acquired” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>The fault has occurred</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>PowerLow</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace (greater than, less than)</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The mission script has started</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “mission too long” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>MissionTooLong</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The mission script has started</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “mission too long” signal is sent</td>
<td></td>
</tr>
</tbody>
</table>

**Name** | **Abbrv** | **Variable** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EchosounderDetects</td>
<td>Agent</td>
<td>ForwardLookingEchosounder</td>
</tr>
<tr>
<td>Type</td>
<td>Inversion</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The obstacle avoidance process has begun idling or is maintaining a retreating state</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>An “obstacle ahead” signal is sent</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>MinIceRange</td>
<td>Agent</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace Greater than, less than</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The obstacle avoidance process has begun idling or is maintaining a retreating state</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “limited headroom” signal is sent</td>
<td></td>
</tr>
</tbody>
</table>

**Name** | **Abbrv** | **Variable** |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MinAltRange</td>
<td>Agent</td>
<td>DepthControlOperation</td>
</tr>
<tr>
<td>Type</td>
<td>State Replace Greater than, less than</td>
<td>Variable</td>
</tr>
<tr>
<td>Options</td>
<td>Random Start</td>
<td>Abbrv</td>
</tr>
<tr>
<td>After</td>
<td>The obstacle avoidance process has begun idling or is maintaining a retreating state</td>
<td></td>
</tr>
<tr>
<td>Until</td>
<td>A “limited headroom” signal is sent</td>
<td></td>
</tr>
</tbody>
</table>
that the AUV appears to be faulty. When the fault is injected a boolean variable is inverted in the Environment agent that denotes a situation in which the AUV appears faulty. The fault can occur at any point after the mission script has started and stops occurring after the Human agent has sent a “homing” signal to the AUV.

The VehicleTooDeep fault injection agent describes a scenario in which the AUV is too deep. A boolean variable is inverted in the MissionControlInternalEvents agent to indicate that a depth threshold has been crossed when the fault is injected. The faulty behaviour can be initiated at any point after the mission script has started until a “vehicle is too deep” signal is sent by the MissionControlInternalEvents agent.

The HomingLostBeforeAcquired fault injection agent details a situation in which a homing signal is lost before the signal has been acquired by the AUV. The value of an enumerate variable indicating whether the beacon signal is on in the HomingSystem agent is replaced with off when the fault is injected in order to lose the signal. This behaviour can occur at any point during the operation of the AUV until a “homing lost” signal is sent by the HomingSystem agent.

The HomingLostAfterAcquired fault injection agent describes the faulty behaviour in which a homing signal is lost after the signal is acquired by the AUV. The faulty behaviour is defined in the same way as the HomingLostBeforeAcquired fault injection agent except that: the behaviour occurs after a “homing acquired” signal is sent by the HomingSystem agent; the behaviour stops occurring immediately after it has begun occurring.

The PowerLow fault injection agent describes a scenario in which the AUV onboard battery power is low. When the fault is injected the value of an enumerate variable of the PowerNode agent indicating that the battery power is greater than a particular threshold signifying low power is replaced with a value indicating it is less than that threshold. The fault can occur at any point after the mission script has started and stops occurring immediately after it has begun occurring. Once the value is less than the threshold signifying low power it remains at that threshold.

The MissionTooLong fault injection agent describes the faulty behaviour in which the AUV mission has continued for too long. The MissionTimer contains a boolean variable indicating when the mission duration has timed out which is inverted when the fault is injected. The fault can be injected at any point after the mission script has started and stops being injected after the MissionTimer has sent a “mission too long” signal.

The AvoidanceTimerFailed fault injection agent represents the scenario in which the duration of the avoidance process has continued for too long during obstacle avoidance. A boolean variable indicating when the avoidance process has timed out is inverted the AvoidanceTimer agent when the fault is injected. The fault can occur at any point after the AUV has begun retreating, and until an “avoidance failed” signal is sent by the AvoidanceTimer agent.

The RetryTrackAvoidanceFailure fault injection agent details a situation in which the AUV has attempted to retreat too many times during obstacle avoidance. A boolean variable in the RetryTrackCounter which indicates that the number of times the AUV has attempted to retreat has reached a present level is inverted when the fault is injected. The fault can occur at any point after the AUV has begun retreating, and until an “avoidance failed” signal is sent by the RetryTrackCounter agent.

The RPAAvoidanceFailure fault injection agent describes the scenario in which there are no waypoints to be sent as demands to the position control during obstacle avoidance. A boolean variable indicating whether there are waypoints to be sent as demands to the position control in the RetreatPathArray agent is inverted when the fault is injected. The fault can occur at any point after the AUV has begun retreating, and until an “avoidance failed” signal is sent by the RetreatPathArray agent.

The HorizontalAvoidanceFailure fault injection agent represents the faulty behaviour that occurs when the horizontal avoidance strategy fails to avoid an obstacle during the obstacle avoidance process. When the fault is injected a boolean variable in the HorizontalAvoidanceFailure agent indicating whether the horizontal avoidance strategy has failed to avoid the obstacle is inverted. The fault can occur at any point after the AUV has begun retreating, and until an “avoidance failed” signal is sent by the HorizontalAvoidanceFailure agent.

The EchosounderDetects fault injection agent details a situation in which the forward looking echosounder has detected an object on the vehicle’s path. A boolean variable in the ForwardLookingEchosounder agent indicating whether an obstacle has been detected is inverted when the fault is injected. The faulty behaviour can occur at any point after either the obstacle avoidance process has begun idling or is maintaining a retreating state, and until an “obstacle ahead” signal is sent by the ForwardLookingEchosounder agent.

The MinIceRange and MinAltRange fault injection agents represent scenarios in which the “minimum ice clearance” and “minimum altitude” configuration limits are reached respectively. When the MinIceRange fault is injected a boolean variable in the DepthControlOperation agent defining whether the “minimum ice clearance” configuration limit has been reached is inverted. Similarly, when the MinAltRange fault is injected a boolean variable in the DepthControlOperation agent defining when the “minimum altitude clearance” configuration limit has been reached is inverted. The faults can occur at any point after either the obstacle avoidance process has begun idling or is maintaining a retreating state, and until a “limited headroom” signal is sent by the DepthControlOperation agent.

III. Specifications

Several specifications were written to verify various properties of the AUV control system, including fault tolerance, recoverability, and diagnosability. We also verified specifications pertaining to additional properties of the system that we expect to be satisfied: all of the faults can be injected.
into the system; and that where appropriate, the faults will eventually stop being injected into the system.

Various properties of the mission control state machine, the homing system, the emergency abort system, and the avoidance strategy were verified. In the following we describe the specifications we wrote to reason about these properties and the results from running them. For the naming convention we use $f_i$ to indicate the "faulty" persistence of a fault; $s_i$ to indicate the "injecting" persistence; and $\lnot s_i$ to indicate the disjunction of the injecting persistence and stopped persistence;

A. Mission control state machine

The following specification verifies that the mission control state machine iterates to the mission elements where $n$ is the number of the mission element

$$AF(mission\_element_n)$$

This formula states that in all paths eventually mission_element_n will be reached. MCMAS verifies this formula as true for the first mission element, and false for the second, third, and fourth mission elements. For the second, third, and fourth mission elements MCMAS provides a counterexample showing that the mission control state machine enters mission termination exceptions instead of iterating to these elements.

To verify that these elements are reached under non-faulty behaviour we employed the following specification where for each Fault in Table I, nofaults is the conjunction of $\lnot Fault_i$.

$$AF(nofaults \rightarrow mission\_element_n)$$

This formula stipulates that in all paths in which there are no faults, eventually mission_element_n will be reached. MCMAS reports this specification as true for all of the mission elements. We can conclude that all of the mission elements are reached under non-faulty behaviour.

To verify that the mission control state machine enters the first mission termination exception mte1 when the power is low, or the AUV fails to avoid an obstacle, we used the following specifications.

$$AG(PL_s \rightarrow AF(mte1))$$
$$AG(ATF_s \rightarrow AF(mte1))$$
$$AG(RPAAF_t \rightarrow AF(mte1))$$
$$AG(HAF_s \rightarrow AF(mte1))$$

These specifications state that in all paths, after the (power low or avoidance failure) fault has occurred, mte1 is reached. MCMAS reports these specifications as false and displays a counterexample showing that the "homing underway" mission element can be reached instead of mte1. Thus, the behaviour of the mission state machine dictates that the occurrence of the homing signal takes priority over low power and obstacle avoidance failures.

We verified that first mission termination exception is reached in the same faulty situations but whenever the homing beacon is not lowered as follows.

$$AG(\lnot VAF_f \land PL_s \rightarrow AF(mte1))$$
$$AG(\lnot VAF_f \land ATF_s \rightarrow AF(mte1))$$

These specifications state that, in all paths in which the vehicle does not appear faulty, (power low or avoidance failure) fault has occurred, mte1 is reached. MCMAS reports these specifications as true. The first mission termination is therefore correctly entered when the vehicle does not appear faulty and the power is low or the AUV fails to avoid an obstacle.

The following specification verifies that the first mission termination exception is a diagnosis property of the system whereby the mission control node knows that the power is low or one of the avoidance failures has occurred.

$$AG(mte1 \rightarrow (K_{MissionScriptExecution}(ATF_s \lor \lnot K_{MissionScriptExecution}(PL_s) \lor \lnot K_{MissionScriptExecution}(RPAAF_t \lor \lnot RTAF_t \lor \lnot HAF_s)$$

This specification states that whenever mte1 is reached, the MissionScriptExecution agent knows that either a power low or obstacle avoidance failure is occurring or has occurred but does not know specifically which one of these faults is occurring or has occurred. MCMAS verifies this specification as true which testifies to the ability of the mission control state machine to diagnose the occurrence of low power or obstacle avoidance failures.

These specifications demonstrated that the AUV can tolerate low power and obstacle avoidance failures by entering a mission termination exception, and that these faults can be collectively diagnosed by the mission control node.

B. Homing system

The following specification verifies that the “homing underway” mission element is entered after the human observes that the vehicle appears faulty.

$$AG(VAF_f \rightarrow AF(homing\_underway))$$

This specification states that in all paths when the vehicle appears faulty the homing_underway state is always reached in the future. MCMAS verifies this specification as false and reports a counterexample showing that if the homing signal is lost before the signal is acquired by the AUV then the mission control does not enter the “homing underway” mission element.

The next specification verifies that if the human observes that the vehicle appears faulty and the homing signal is not lost before the signal is acquired by the AUV then the “homing underway” mission element is entered.

$$AG(\lnot HLBA_f \land VAF_f \rightarrow AF(homing\_underway))$$

This specification states that in all paths in which the homing signal is not lost before it is acquired, when the vehicle appears faulty the homing_underway state is always reached in the future. MCMAS verifies this specification as true. This means that the mission control state machine
always enters the “homing underway” mission element if the vehicle appears faulty and the signal is acquired.

The following specification verifies that if the homing signal is lost after it is acquired then the “homing signal lost” mte2 mission element is entered.

\[ AG(HLAA_i → AF(mte2)) \]

This specification states that in all paths when the homing signal is lost after it has been acquired the mte2 state is always reached in the future. MCMAS verifies this specification as true. Thus, the mission control state machine always enters a “homing lost” mission element after an acquired homing signal has been lost.

Once the homing signal has been lost, the AUV can re-acquire the homing signal. The following specification verifies recoverability to the “homing underway” mission element after the signal has been lost after it has been acquired.

\[ AG(HLAA_i → AF(homing\_underway)) \]

This specification states that in all paths when the homing signal is lost after it has been acquired the homing\_underway state is always reached in the future. MCMAS verifies this specification as false and provides a counterexample showing that the signal can be lost before it is re-acquired which prevents recovery.

The next specification verifies recovery to the “homing underway” mission element when the signal is not lost before it is re-acquired.

\[ AG((HLAA \land \neg HLBA_f) → AF(homing\_underway)) \]

This specification states that in all paths in which the homing signal is not lost before it is acquired, when the homing signal is lost after it has been acquired the homing\_underway state is always reached in the future. MCMAS verifies this specification as true. This testifies to the recoverability of the mission control state machine with regards to resuming homing when an acquired homing signal is lost.

These specifications demonstrated that the mission control state machine enters a “homing underway” element if a homing signal is not lost before it is acquired, and that the homing system can recover if the homing signal is lost after it is acquired.

C. Emergency abort system

The following specifications verify that the Emergency Abort node responds when the vehicle is too deep or the mission has continued for too long.

\[ AG(VT_{F_e} → AF(mission\_aborted)) \]
\[ AG(MT_{L_a} → AF(mission\_aborted)) \]

These specifications state that the in all paths in which there is an occurrence of a power low or mission too long fault a state is reached. MCMAS verifies these specifications as true. Thus the Emergency Abort node agent enters a “mission aborted state” when the vehicle is too deep or the mission has continued for too long.

The following specification verifies that the “mission aborted state” is a diagnosis property of the system whereby the emergency abort system node knows that the vehicle is too deep or that the mission has continued for too long.

\[ AG(mission\_aborted → (K_{EmergencyAbortSystem}(VT_{D_a} \lor MT_{L_a}) \land \neg K_{EmergencyAbortSystem}(VT_{D_a}) \land \neg K_{EmergencyAbortSystem}(MT_{L_a}))) \]

This specification states that whenever a mission\_aborted state is reached, the EmergencyAbortSystem agent knows that either a vehicle too deep or mission too long fault is occurring or has occurred but does not know specifically which one of these faults is occurring or has occurred. MCMAS verifies this specification as true which testifies to the ability of the Emergency Abort system to diagnose that the vehicle is too deep or the mission has continued for too long.

These specifications demonstrated that the AUV can tolerate situations in which it is too deep or the mission has continued for too long by entering a mission aborted state, and that these faults can be collectively diagnosed by the Emergency Abort node.

D. Avoidance strategy

The following specification verifies that the forward looking echosounder can diagnose the the detection of an object on the vehicle’s path.

\[ AG(ED_i → AF(K_{ForwardLookingEchosounder}(ED_i))) \]

This specification states that in all paths in which the echosounder detects fault occurs, the ForwardLookingEchosounder agent always eventually knows that the fault is occurring or has occurred. MCMAS verifies this specification as true. Thus, when an object has been detected on the vehicle’s path the forward looking echosounder diagnoses it.

The following specification verifies that the depth control node can diagnose that the “minimum ice clearance” and “minimum altitude” configuration limits have been reached.

\[ AG((MIR_i \land MAR_i) → AF(K_{DepthControlOperation}(MIR_i \land MAR_i))) \]

This specification states that in all paths in which the minimum ice clearance and minimum altitude faults occur, the DepthControlOperation agent always eventually knows that these faults are occurring or have occurred. MCMAS verifies this specification as true. We can conclude that when the “minimum ice clearance” and “minimum altitude” configuration limits are reached the depth control node diagnoses that these limits have been reached.

The following specification proves that when the forward looking echosounder diagnoses the detection of the object on the vehicle’s path, the knowledge of the fault is propagated to the horizontal control dimension avoidance strategy.

\[ \neg E(\neg K_{ForwardLookingEchosounder}(ED_i) \lor (K_{ForwardLookingEchosounder}(ED_i) \land \neg AF(K_{HorizontalAvoidanceStrategy}(ED_i \lor (MIR_i \land MAR_i)))) \]

This specification states that there is no path in which the ForwardLookingEchosounder agent comes to know that the echosounder detects fault has occurred or is occurring and it is not true that the HorizontalAvoidanceStrategy agent always comes to know that the echosounder detects fault has occurred or is occurring. MCMAS verifies this specification
as true. Thus, the forward looking echosounder propagates its knowledge of the detection of the object on the vehicles path to the horizontal control dimension avoidance strategy.

We also employed similar specifications which verified that: the forward looking echosounder propagates its knowledge of the detection of the object on the vehicles path to the vertical control dimension avoidance strategy; the forward depth control node propagates its knowledge that the “minimum ice clearance” and “minimum altitude” configuration limits have been reached to the horizontal and vertical control dimension avoidance strategy. MCMAS verified these specifications as true.

The following specification verifies that when the forward looking horizontal control dimension avoidance strategy diagnoses the detection of the object on the vehicles path or the reaching of the “minimum ice clearance” and “minimum altitude” configuration limits, the knowledge of that one of these faults has occurred is propagated to the obstacle avoidance timer.

\[\neg E(\neg K_{\text{HorizontalAvoidanceStrategy}}(ED_{ix} \lor (MIR_{ix} \land MIA_{ix}))) \land \neg AF(K_{\text{AvoidanceTimer}}(ED_{ix} \lor (MIR_{ix} \land MIA_{ix}))))\]

This specification states that there is no path in which the HorizontalAvoidanceStrategy comes to know that the echosounder detects fault has occurred or is occurring or the minimum ice clearance and minimum altitude configuration limits faults have occurred or are occurring and it is not true that the AvoidanceTimer agent always comes to know that the echosounder detects fault has occurred or is occurring or the minimum clearance and minimum altitude configuration limits faults have occurred or are occurring. MCMAS verifies this specification as true. This means that the horizontal control dimension avoidance strategy propagates its knowledge of the detection of the object on the vehicles path or the reaching of the “minimum ice clearance” and “minimum altitude” configuration limits to the obstacle avoidance timer.

We also employed similar specifications to verify that when the horizontal control dimension avoidance strategy diagnoses the detection of the object on the vehicles path or the reaching of the “minimum ice clearance” and “minimum altitude” configuration limits, the knowledge of that one of these faults has occurred is propagated to: The retry track counter; the safe position meter; the retry track length monitor; the new demand detection; the obstacle clearance meter. For the retry track counter, safe position meter, and retry track length monitor, MCMAS verified these specifications as true. For the new demand detection, obstacle clearance meter, and retreat track length monitor, MCMAS verified these specifications as false. Counterexamples were reported for these specifications showing that: for the retreat path array the safe position meter can put the agent can instruct the array to save a safe position as a waypoint when the safe distance has been covered which means it may not be retrieving a waypoint where it knows about the obstacle; the obstacle clearance meter needs the horizontal avoidance strategy to begin trying a new track before it knows about the obstacle; the new demand detection requires a position instruction before it can know about the obstacle.

The following specification verifies that when the forward looking horizontal control dimension avoidance strategy diagnoses the detection of the object on the vehicles path or the reaching of the “minimum ice clearance” and “minimum altitude” configuration limits, the knowledge of that one of these faults has occurred is propagated to avoidance timer, retry track counter, safe position meter, and retry track length monitor.

\[\neg E(\neg K_{\text{HorizontalAvoidanceStrategy}}(ED_{ix} \lor (MIR_{ix} \land MIA_{ix}))) \land \neg AF(K_{\text{AvoidanceTimer}}(ED_{ix} \lor (MIR_{ix} \land MIA_{ix})))\]

This specification states that there is no path in which the HorizontalAvoidanceStrategy agent comes to know that the echosounder detects fault has occurred or is occurring or the minimum ice clearance and minimum altitude configuration limits faults have occurred or are occurring and it is not true that everybody in the group ObstacleAvoidanceGroup (comprising of the AvoidanceTimer, RetryTrackCounter, SafePositionMeter, and RetryTrackLengthMonitor agents) always comes to know that the echosounder detects fault has occurred or is occurring or the minimum ice clearance and minimum altitude configuration limits faults have occurred or are occurring. MCMAS verifies this specification as true. Thus, the horizontal control dimension avoidance strategy propagates its knowledge of the detection of the object on the vehicles path or the reaching of the “minimum ice clearance” and “minimum altitude” configuration limits to the avoidance timer, retry track counter, safe position meter, and retry track length monitor.

We employed a similar specification to verify that when the forward looking horizontal control dimension avoidance strategy diagnoses the detection of the object on the vehicles path or the reaching of the “minimum ice clearance” and “minimum altitude” configuration limits, the knowledge of that one of these faults has occurred becomes common knowledge amongst the avoidance timer, retry track counter, safe position meter, and retry track length monitor. MCMAS verifies this specification as false. It follows that although these agents are required to know about the obstacle in order to allow the AUV to avoid it, they do not need to know that each other knows about the obstacle.

During the collision and obstacle avoidance the vehicle may try an indefinite number of retries during the avoidance, and can get stuck in this situation. The safeguard mechanisms will trigger an avoidance failed event that will alter the mission script execution. After sending the avoidance failed event to the mission script execution, the avoidance control will continue the retreat and try new track sequences, until a new demand is received from the mission script execution. The new demand detection occurs, as the result of the mission termination exception’s issued joint instruction, that is determined, though, by the avoidance failed event. The avoidance failed event determines the mission script execution to evolve from either of the mission elements (at the execution of which the obstacle is detected) to mte1’s joint
instruction issuing state. At this stage, though, if the vehicle is stuck in the avoidance, after the retreat stage the new demand detection will sense a new position control demand (send pctrl demand in position mode or send pctrl demand in track follow mode) and will notify the horizontal avoidance strategy through the new demand detection occurs action. This action will determine the horizontal avoidance strategy to resume the idling or waiting state. The following specification verifies that under these circumstances and in a situation where the AUV is not homing, the correct sequence of actions allows the horizontal avoidance strategy to resume the waiting state.

\[
\neg E((\text{tryingnewtrack} \land \neg VAF \land \neg ATF \land \neg RTAF) \\
\quad U(\neg ATF \lor RTAF) \land \neg AF(mte1) \\
\quad \land AF(\text{NewPositionControlDemand}) \\
\quad \land AF(\text{NewDemandDetection}) \\
\quad \land AF(\text{waiting}))
\]

This specification states that there is no path in which the vehicle does not appear faulty and the tryingnewtrack state does not always return to waiting after the following sequence of actions occurs: a RetryTrackAvoidanceFail or AvoidanceTimerFailed fault has occurred; a NewPositionControlDemand occurs; a NewDemandDetection occurs; the mte1 state is reached. MCMAS verifies this specification as true. This testifies to the ability of the obstacle avoidance system in executing the correct sequence of actions so that it does not get stuck in a situation where it indefinitely retries to avoid an obstacle.

These specifications have demonstrated that: the forward looking echosounder and depth control node can diagnose obstacles; the knowledge of these obstacles is distributed to the horizontal and vertical control dimension avoidance strategies; the horizontal control dimension avoidance strategy further distributes this knowledge to the obstacle avoidance agents that required it; the obstacle avoidance system does not get stuck in a situation where it indefinitely retries to avoid an obstacle.

REFERENCES


Appendix I

Stateflow diagrams illustrating the labelled transition system formulation of the multi-agent model
Emergency_abort_system

no_eas_act &... 
!mission_has_continued_for_too_long &... 
!vehicle_is_too_deep

normal_operation 
du: no_eas_act ();

mission_aborted 
du: abort_mission_and_rise_to_surface ()

function no_eas_act

function abort_mission_and_rise_to_surface
Horizontal_avoidance_strategy

State1
//has_stat=initialize_components_for_waiting_state
//has_avoidance_failed_faulty=false
du: start_waiting();

State2
//has_stat=waiting
du: avoidance_process_is_idling &... !send_limited_headroom_to_pctrl &... !send_obstacle_ahead_to_pctrl

State3
//has_stat=initialize_components_for_retreating_state
du: start_retreating();

State4
//has_stat=retreating
du: maintain_retreating &... !send_limited_headroom_to_pctrl &... !send_obstacle_ahead_to_pctrl &... !rpa_becomes_empty &... !end_of_safe_retreat_track_is_reached

State5
//has_stat=obstacle_detected_while_retreating
du: continue_initializing_components_for_trying_new_track_state

State6
//has_stat=initialize_components_for_trying_new_track_state
du: start_trying_new_track();

State7
//has_stat=avoidance_failed_event_sent_to_mission_control_by_hca
//has_avoidance_failed_faulty=false
du: no_has_act();

State8
//has_stat=obstacle_detected_while_retreating
du: maintain_retreating &... !send_limited_headroom_to_pctrl &... !send_obstacle_ahead_to_pctrl &... !rpa_becomes_empty &... !end_of_safe_retreat_track_is_reached

State9
//has_stat=trying_new_track
du: maintain_trying_new_track();

Function: start_waiting
Function: avoidance_process_is_idling
Function: continue_initializing_components_for_trying_new_track_state
Function: start_retreating
Function: maintain_retreating
Function: continue_initializing_components_for_trying_new_track_state
Function: start_trying_new_track
Function: maintain_trying_new_track
Function: hca_sends_avoidance_failed_event_to_msctrl
Function: no_has_act
Safe_position_meter

function add_travelled_distance_segment_to_spm_accumulated_distance

function no_spm_act

function safe_position_distance_has_been_covered

function spm_accumulated_distance_reaches_value_for_track_update

function start_spm_and_rpa_standby_operation
function allowed_number_of_rtc_exceeded

function rtc_sends_avoidance_failed_event_to_msctrl

function no_rtc_act
Avoidance_timer

State1
//at_stat=idling
//timeout=false
du: no_at_act ();

State2
//at_stat=operation_enabled
//timeout=false
du: no_at_act ();

State3
//at_stat=operation_enabled
//timeout=true
du: no_at_act ();

State4
//at_stat=avoidance_failed_event_sent_to_msctrl_by_at
du: at_sends_avoidance_failed_event_to_msctrl ();

State5
//at_stat=timeout_exceeded
du: at_overflows ();

function at_sends_avoidance_failed_event_to_msctrl
function at_overflows
function no_at_act
New_demand_detection

function new_demand_detection_occures

function no_ndd_act

has_retreating du: no_ndd_act ();

start_retreating du: new_demand_detection_occures ();

demand du: new_demand_detection_occures ();

send_pctrl_demand_in_position_mode

send_pctrl_demand_in_track_follow_mode

no_ndd_act &...

!send_pctrl_demand_in_position_mode &...

!send_pctrl_demand_in_track_follow_mode

new_demand_detection_occures &...

!start_retreating
Mission_timer

State1
//mt_stat=idling
du: no_mt_act ();

State2
//mt_stat=initialized
du: schedule_mt_to_expire ();

State3
//mt_stat=operation_enabled
//timeout=false
du: no_mt_act ();

State7
//mt_stat=operation_enabled
//timeout=true
du: no_mt_act ();

State5
//mt_stat=duration_too_long
du: mission_has_continued_for_too_long ();

State6
//mt_stat=expired
du: no_mt_act ();

Function: mission_has_continued_for_too_long
Function: no_mt_act
Function: schedule_mt_to_expire
Line_timer

function it_reset

function lt_reaching_timeout

function send_line_timeout_event_to_msctrl

function no_lt_act

function schedule_lt_to_expire
Mission_timer_instruction_parser

function no_mtip_act

function set_mission_timer

idle
du: no_mtip_act ()

set
du: set_mission_timer ()

issue_first_me_joint_instr

no_mtip_act &... !issue_first_me_joint_instr

set_mission Timer
Line_timer_instruction_parser

idle
du: no_ltip_act ();

function no_ltip_act
......

function set_line_timer

set

du: set_line_timer ();

issue_first_me_joint_instr

issue_third_me_joint_instr

set_line_timer

no_ltip_act &...

issue_first_me_joint_instr &...

issue_third_me_joint_instr
Depth_control_instruction_parser

1. idle du:
   no_dcip_act();

2. surface_mode du:
   send_dctrl_demand_in_surface_mode();

3. depth_mode du:
   send_dctrl_demand_in_depth_mode();

4. sternplane_angle_mode du:
   send_dctrl_demand_in_sternplane_angle_mode();

5. function no_dcip_act
6. function send_dctrl_demand_in_surface_mode
7. function send_dctrl_demand_in_depth_mode
8. function send_dctrl_demand_in_sternplane_angle_mode
Position_control_instruction_parser

idle
du: no_pcip_act();

function no_pcip_act

function send_pctrl_demand_in_position_mode

function send_pctrl_demand_in_track_follow_mode

position_mode
du: send_pctrl_demand_in_position_mode();

track_follow_mode
du: send_pctrl_demand_in_track_follow_mode();
```plaintext
Motor_control_instruction_parser

idle
du: no_mcip_act ();

power_mode
du: send_mctrl_demand_in_power_mode ();

function no_mcip_act

function send_mctrl_demand_in_power_mode
```
Power_node

State1
//pwr_stat=idle
//pwr_ref=safe_level & pwr_rel=greater_than
du: no_pn_act();

State2
//pwr_stat=power_normal
//pwr_ref=safe_level & pwr_rel=greater_than
du: no_pn_act();

State3
//pwr_stat=power_low_event_sent_to_msctrl
//pwr_ref=safe_level & pwr_rel=less_than
du: battery_voltage_fallen_below_set_level();

State4
//pwr_stat=power_low
//pwr_ref=safe_level & pwr_rel=less_than
du: battery_voltage_stays_below_set_level();

State5
//pwr_stat=power_normal
//pwr_ref=safe_level & pwr_rel=less_than
du: battery_voltage_stays_below_set_level();

function battery_voltage_stable_above_set_level
function battery_voltage_fallen_below_set_level
function battery_voltage_stays_below_set_level
function send_power_low_event_to_msctrl
function no_pn_act

mission_script_started

no_pn_act & !mission_script_started

battery_voltage_stays_below_set_level

send_power_low_event_to_msctrl

no_pn_act & !mission_script_started

battery_voltage_stays_below_set_level

send_power_low_event_to_msctrl
calculated_position_not_available

State1
//gps_stat=fix_not_acquired
du: calculated_position_available();

State2
//gps_stat=fix_acquired_event_sent_to_mission_control
du: send_got_gps_event_to_msctrl();

State3
//gps_stat=fix_acquired
du: calculated_position_available();

function calculated_position_available

function calculated_position_not_available

function send_got_gps_event_to_msctrl
Mission_control_internal_events

State 1
//intevent_stat=on_surface_event_sent_to_msctrl
//depth_ref=surface_depth & depth_rel=equal
du: send_on_surface_event_to_msctrl();

State 2
//intevent_stat=vehicle_on_surface
//depth_ref=surface_depth & depth_rel=equal
du: no_mcie_act();

State 3
//intevent_stat=dived_event_sent_to_msctrl
//depth_ref=surface_depth & depth_rel=greater_than
du: send_vehicle_dived_event_to_msctrl();

State 4
//intevent_stat=vehicle_dived
//depth_ref=surface_depth & depth_rel=greater_than
//threshold_crossed=false
du: no_mcie_act();

State 5
//intevent_stat=vehicle_dived
//depth_ref=surface_depth & depth_rel=greater_than
//threshold_crossed=true
du: send_too_deep_event_to_msctrl();

State 6
//intevent_stat=vehicle_too_deep
//depth_ref=max_depth & depth_rel=greater_than
//threshold_crossed=true
du: vehicle_is_too_deep();

State 7
//intevent_stat=got_depth_event_sent_to_msctrl
//depth_ref=dctrl_demand & depth_rel=equal
du: send_got_depth_event_to_msctrl();

State 8
//intevent_stat=vehicle_got_depth
//depth_ref=dctrl_demand & depth_rel=equal
//threshold_crossed=false
du: no_mcie_act();

State 9
//intevent_stat=vehicle_got_depth
//depth_ref=dctrl_demand & depth_rel=equal
//threshold_crossed=true
du: send_too_deep_event_to_msctrl();

function send_got_depth_event_to_msctrl
function vehicle_is_too_deep
function send_vehicle_dived_event_to_msctrl
function send_on_surface_event_to_msctrl
function send_too_deep_event_to_msctrl
function no_mcie_act
function fwddist_out_of_range

function send_obstacle_ahead_to_pctrl

State1
//echosounder_stat=idle
//fwddist_ref=max_range & fwddist_rel=greater_than
du: fwddist_out_of_range();

State2
//echosounder_stat=not_detected
//fwddist_ref=max_range & fwddist_rel=greater_than
du: fwddist_out_of_range();

State3
//echosounder_stat=not_detected
//fwddist_ref=max_range & fwddist_rel=less_than
du: send_obstacle_ahead_to_pctrl();

State4
//echosounder_stat=detected
//fwddist_ref=max_range & fwddist_rel=less_than
du: fwddist_out_of_range();

mission_script_started

fwddist_out_of_range
function no_vas_act

no_vas_act &... !send_obstacle_ahead_to_pctrl &...

!send_limited_headroom_to_pctrl &...

no_vas_act &... !send_obstacle_avoided_event_to_msctrl

send_obstacle_avoided_event_to_msctrl

send_obstacle_ahead_to_pctrl

send_limited_headroom_to_pctrl &...

!send_obstacle_ahead_to_pctrl &...

!send_limited_headroom_to_pctrl &...

State1
//max_depth=normal & min_depth=normal
//min_alt=normal & min_ice_clearance=normal
du: no_vas_act();

State2
//max_depth=safe & min_depth=safe
//min_alt=safe & min_ice_clearance=safe
du: no_vas_act();
Environment

State1
//vehicle_appears_faulty=false
du: no_env_act();

State2
//vehicle_appears_faulty=true
du: auv_appears_faulty();

function no_env_act
function auv_appears_faulty
Human

State 1
//human_stat=observing_auv
du: no_human_act ();

function no_human_act

function human_sends_homing_signal

State 2
//human_stat=sending_homing
auv_appears_faulty
du: human_sends_homing_signal ();
Appendix II

Multi-agent model
formulated in the interpreted system programming language
Agent Environment
Obsvars:
end Obsvars
Vars:
    vehicle_appears_faulty: boolean;
end Vars
RedStates:
end RedStates
Actions = { auv_appears_faulty, no_env_act };
Protocol:
    vehicle_appears_faulty=false: { no_env_act };
    vehicle_appears_faulty=true: { auv_appears_faulty };
end Protocol
Evolution:
    vehicle_appears_faulty = false if vehicle_appears_faulty=false and
    !VehicleAppearsFaulty.Action = inject;
    vehicle_appears_faulty = true if vehicle_appears_faulty=false and
    VehicleAppearsFaulty.Action = inject;
end Evolution
end Agent

Agent Depth_control_operation
Vars:
    limhdr_stat: { detected, idle, not_detected };
    rd_ref: { min_alt };
    rd_rel: { greater_than, less_than };
    ru_ref: { min_ice_clearance };
    ru_rel: { greater_than, less_than };
end Vars
RedStates:
end RedStates
Actions = { min_alt_cmd_override, min_ice_clr_cmd_override, no_dc_act,
    return_to_safe_watercolumn, send_limited_headroom_to_pctrl };
Protocol:
    (((limhdr_stat=idle and rd_ref=min_alt) and rd_rel=greater_than)
     and ru_ref=min_ice_clearance) and ru_rel=greater_than) or
    (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=greater_than)
     and ru_ref=min_ice_clearance) and ru_rel=greater_than);
    (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than)
     and ru_ref=min_ice_clearance) and ru_rel=greater_than): { min_alt_cmd_override,
     return_to_safe_watercolumn };
    (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than)
     and ru_ref=min_ice_clearance) and ru_rel=less_than): { send_limited_headroom_to_pctrl };
    (((limhdr_stat=detected and rd_ref=min_alt) and rd_rel=greater_than)
     and ru_ref=min_ice_clearance) and ru_rel=less_than) or
    (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than)
     and ru_ref=min_ice_clearance) and ru_rel=less_than): { min_ice_clr_cmd_override };
    (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than)
     and ru_ref=min_ice_clearance) and ru_rel=greater_than): { min_ice_clr_cmd_override,
     return_to_safe_watercolumn };
    (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than)
     and ru_ref=min_ice_clearance) and ru_rel=less_than): { min_ice_clr_cmd_override,
     return_to_safe_watercolumn };
end Protocol
Evolution:
    limhdr_stat = idle and rd_ref = min_alt and rd_rel = greater_than and ru_ref = min_ice_clearance
    and ru_rel = greater_than if ((Action=no_dc_act and !Mission_script_execution.Action=mission_script_started)
     and ((limhdr_stat=idle and rd_ref=min_alt) and rd_rel=greater_than) and ru_ref=min_ice_clearance) and
    ru_rel=greater_than) and !Min_Alt_Range.Action = inject and
    !Min_Ice_Range.Action = inject;
    limhdr_stat = idle and rd_ref = min_alt and rd_rel = greater_than and ru_ref = min_ice_clearance
    and ru_rel = less_than if ((Action=no_dc_act and !Mission_script_execution.Action=mission_script_started)
     and ((limhdr_stat=idle and rd_ref=min_alt) and rd_rel=greater_than) and ru_ref=min_ice_clearance) and
    ru_ref=min_ice_clearance);
ru_rel=greater_than)) and Min_Ice_Range.Action = inject and Min_Ice_Range.Action = inject;
  limhdr_stat = idle and rd_ref = min_alt and ru_rel = greater_than if ((Action=no_dc_act and !Mission_script_execution.Action=mission_script_started). and (((limhdr_stat=idle and rd_ref=min_alt) and ru_rel=greater_than) and ru_rel=greater_than)) and Min_Ice_Range.Action = inject and !Min_Ice_Range.Action = inject;
  limhdr_stat = idle and rd_ref = min_alt and rd_rel = less_than and ru_ref = min_ice_clearance and ru_rel = greater_than if ((Action=no_dc_act and !Mission_script_execution.Action=mission_script_started). and (((limhdr_stat=idle and rd_ref=min_alt) and ru_rel=greater_than) and ru_rel=greater_than)) and Min_Ice_Range.Action = inject.

  limhdr_stat = not_detected and rd_ref = min_alt and rd_rel = less_than and ru_ref = min_ice_clearance and ru_rel = greater_than if (Action=min_alt_cmd_override and (((limhdr_stat=not_detected and rd_rel=less_than) and ru_ref=min_ice_clearance) and ru_rel=greater_than) or (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than) and ru_ref=min_ice_clearance) and ru_rel=greater_than)) and !Min_Ice_Range.Action = inject;
  limhdr_stat = not_detected and rd_ref = min_alt and rd_rel = less_than and ru_ref = min_ice_clearance and ru_rel = less_than if (Action=min_alt_cmd_override and (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than) and ru_ref=min_ice_clearance) and ru_rel=less_than) or (Mission_script_execution.Action=mission_script_started and (((limhdr_stat=idle and rd_ref=min_alt) and rd_rel=greater_than) and ru_ref=min_ice_clearance) and ru_rel=greater_than)) or (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than) and ru_ref=min_ice_clearance) and ru_rel=less_than) and Min_Ice_Range.Action = inject;
  limhdr_stat = not_detected and rd_ref = min_alt and rd_rel = greater_than and ru_ref = min_ice_clearance and ru_rel = greater_than if (((Action=return_to_safe_watercolumn and (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=less_than) and ru_ref=min_ice_clearance) and ru_rel=greater_than) or (((limhdr_stat=not_detected and rd_ref=min_alt) and rd_rel=greater_than) and ru_ref=min_ice_clearance) and ru_ref=greater_than) or (((limhdr_stat=detected and rd_ref=min_alt) and rd_rel=less_than) and ru_ref=min_ice_clearance) and ru_rel=less_than))) and !Min_Alt_Range.Action = inject and !Min_Ice_Range.Action = inject.
ru_ref = min_ice_clearance and ru_rel = greater_than if 
((((Action=return_to_safe_watercolumn and (((limhdr_stat=not_detected and 
rd_ref=min_alt) and rd_rel=less_than) and ru_ref=min_ice_clearance) and 
ru_rel=greater_than) or (((limhdr_stat=not_detected and rd_ref=min_alt) and 
rd_rel=greater_than) and ru_ref=min_ice_clearance) and ru_ref=min_ice_clearance) and 
ru_rel=less_than)) or (((limhdr_stat=not_detected and rd_ref=min_alt) and 
rd_rel=greater_than) and ru_ref=min_ice_clearance) and ru_ref=greater_than))

Agent End

Agent Vertical_avoidance_strategy

Vars:
max_depth: { normal, safe };
min_alt: { normal, safe };
min_depth: { normal, safe };

end Evolution

end Agent
min_ice_clearance: { normal, safe }; 
end Vars 
RedStates:

Actions = { no_vas_act }; 
Protocol:
   (((max_depth=normal and min_alt=normal) and min_depth=normal) and min_ice_clearance=normal) or (((max_depth=safe and min_alt=safe) and min_depth=safe) and min_ice_clearance=safe)); { no_vas_act }; 
end Protocol
Evolution:
   max_depth = normal and min_alt = normal and min_depth = normal and min_ice_clearance = normal if (((Action=no_vas_act and !Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and !Depth_control_operation.Action=send_limited_headroom_to_pctrl) and !Obstacle_clearance_meter.Action=send_obstacle_avoided_event_to_msctrl and (((max_depth=normal and min_alt=normal) and min_depth=normal) and min_ice_clearance=normal)) or (Obstacle_clearance_meter.Action=send_obstacle_avoided_event_to_msctrl and (((max_depth=safe and min_alt=safe) and min_depth=safe) and min_ice_clearance=safe)));

max_depth = safe and min_alt = safe and min_depth = safe and min_ice_clearance = safe if (((Depth_control_operation.Action=send_limited_headroom_to_pctrl and !Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and !Depth_control_operation.Action=send_limited_headroom_to_pctrl) and !Obstacle_clearance_meter.Action=send_obstacle_avoided_event_to_msctrl and (((max_depth=normal and min_alt=normal) and min_depth=normal) and min_ice_clearance=normal)) or (Action=no_vas_act) and !EchosounderDetects.Action = inject; 
end Evolution
end Agent

Agent Forward_looking_echosounder
Vars:
   echosounder_stat: { detected, idle, not_detected }; 
   fwddist_ref: { max_range }; 
   fwddist_rel: { greater_than, less_than }; 
end Vars 
RedStates:

Actions = { fwddist_out_of_range, send_obstacle_ahead_to_pctrl }; 
Protocol:
   ((((echosounder_stat=not_detected and fwddist_ref=max_range) and fwddist_rel=less_than)) and send_obstacle_ahead_to_pctrl ); 
   ((((echosounder_stat=detected and fwddist_ref=max_range) and fwddist_rel=less_than) or ((echosounder_stat=not_detected and fwddist_ref=max_range) and fwddist_rel=greater_than)) or (echosounder_stat=idle and fwddist_ref=max_range) and fwddist_rel=greater_than)); { fwddist_out_of_range }
end Protocol
Evolution:
   echosounder_stat = detected and fwddist_ref = max_range and fwddist_rel = less_than if (Action=send_obstacle_ahead_to_pctrl and ((echosounder_stat=not_detected and fwddist_ref=max_range) and fwddist_rel=less_than) or ((echosounder_stat=not_detected and fwddist_ref=max_range) and fwddist_rel=greater_than)) or (Mission_script_execution.Action=mission_script_started and (echosounder_stat=idle and fwddist_ref=max_range) and fwddist_rel=greater_than)) and !EchosounderDetects.Action = inject; 
   echosounder_stat = not_detected and fwddist_ref = max_range and fwddist_rel = greater_than if ((Action=fwddist_out_of_range and ((echosounder_stat=detected and fwddist_ref=max_range) and fwddist_rel=less_than) or ((echosounder_stat=not_detected and fwddist_ref=max_range) and fwddist_rel=greater_than)) or (Mission_script_execution.Action=mission_script_started and (echosounder_stat=idle and fwddist_ref=max_range) and fwddist_rel=less_than) or ((echosounder_stat=not_detected and fu
greater_than if ((Action=fwddist_out_of_range and !Mission_script_execution.Action=mission_script_started) and ((echosounder_stat=idle and fwddist_ref=max_range) and fwddist_rel=greater_than)) and !EchosounderDetects.Action = inject;

end Evolution

end Agent

Agent Mission_control_internal_events

Vars:

  depth_ref: { dctrl_demand, max_depth, surface_depth };
  depth_rel: { equal, greater_than };
  interevent_stat: { dived_event_sent_to_mscrl, got_depth_event_sent_to_mscrl,
    on_surface_event_sent_to_mscrl, vehicle_dived, vehicle_got_depth,
    vehicle_on_surface, vehicle_too_deep };
  threshold_crossed: boolean;

end Vars

RedStates:

Actions = { no_mcie_act, send_got_depth_event_to_mscrl,
            send_on_surface_event_to_mscrl, send_too_deep_event_to_mscrl,
            send_vehicle_dived_event_to_mscrl, vehicle_is_too_deep };

Protocol:

  (depth_ref=surface_depth and depth_rel=equal) and interevent_stat=on_surface_event_sent_to_mscrl): { send_on_surface_event_to_mscrl };

  (((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth) and threshold_crossed=false) or
  (((depth_ref=surface_depth and depth_rel=equal) and interevent_stat=vehicle_got_depth) and threshold_crossed=true)) or
  (((depth_ref=dctrl_demand and depth_rel=equal) and threshold_crossed=false)); { no_mcie_act };

  (depth_ref=surface_depth and depth_rel=greater_than and interevent_stat=dived_event_sent_to_mscrl): { send_vehicle_dived_event_to_mscrl };

  (((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=got_depth_event_sent_to_mscrl) and threshold_crossed=false)); { send_got_depth_event_to_mscrl };

  (((depth_ref=surface_depth and depth_rel=greater_than) and interevent_stat=vehicle_got_depth) and threshold_crossed=true)); { send_too_deep_event_to_mscrl };

  (depth_ref=max_depth and depth_rel=greater_than and threshold_crossed=true)): { vehicle_is_too_deep };

end Protocol

Evolution:

  depth_ref = surface_depth and depth_rel = equal and interevent_stat = on_surface_event_sent_to_mscrl if (Depth_control_instruction_parser.Action=send_dctrl_demand_in_surface_mode and (((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth) and threshold_crossed=false)) or
  (((depth_ref=surface_depth and depth_rel=greater_than) and interevent_stat=vehicle_got_depth) and threshold_crossed=true)));

  depth_ref = dctrl_demand and depth_rel = equal and interevent_stat = vehicle_got_depth and threshold_crossed = false if
  ((Action=send_got_depth_event_to_mscrl and ((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=got_depth_event_sent_to_mscrl)) or
  ((Action=no_mcie_act and ! Depth_control_instruction_parser.Action=send_dctrl_demand_in_surface_mode) and ! Depth_control_instruction_parser.Action=send_dctrl_demand_in_depth_mode) and
  (((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth) and threshold_crossed=false)) and
  !VehicleTooDeep.Action = inject;

  depth_ref = dctrl_demand and depth_rel = equal and interevent_stat = vehicle_got_depth and threshold_crossed = true if
  ((Action=send_got_depth_event_to_mscrl and ((depth_ref=dctrl_demand and depth_rel=equal) and threshold_crossed=true)); { no_mcie_act };

  (depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth): { send_got_depth_event_to_mscrl };

  (((depth_ref=surface_depth and depth_rel=greater_than) and interevent_stat=vehicle_dived) and threshold_crossed=true)); { send_too_deep_event_to_mscrl };

  (depth_ref=max_depth and depth_rel=greater_than) and threshold_crossed=true)): { vehicle_is_too_deep };

end Protocol

Evolution:

  depth_ref = surface_depth and depth_rel = equal and interevent_stat = on_surface_event_sent_to_mscrl if (Depth_control_instruction_parser.Action=send_dctrl_demand_in_surface_mode and (((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth) and threshold_crossed=false)) or
  (((depth_ref=surface_depth and depth_rel=greater_than) and interevent_stat=vehicle_got_depth) and threshold_crossed=true)));

  depth_ref = dctrl_demand and depth_rel = equal and interevent_stat = vehicle_got_depth and threshold_crossed = false if
  ((Action=send_got_depth_event_to_mscrl and ((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=got_depth_event_sent_to_mscrl)) or
  ((Action=no_mcie_act and ! Depth_control_instruction_parser.Action=send_dctrl_demand_in_surface_mode) and ! Depth_control_instruction_parser.Action=send_dctrl_demand_in_depth_mode) and
  (((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth) and threshold_crossed=false)) and
  !VehicleTooDeep.Action = inject;

  depth_ref = dctrl_demand and depth_rel = equal and interevent_stat = vehicle_got_depth and threshold_crossed = true if
  ((Action=send_got_depth_event_to_mscrl and ((depth_ref=dctrl_demand and depth_rel=equal) and threshold_crossed=true)); { no_mcie_act };

  (depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth): { send_got_depth_event_to_mscrl };

  (((depth_ref=surface_depth and depth_rel=greater_than) and interevent_stat=vehicle_dived) and threshold_crossed=true)); { send_too_deep_event_to_mscrl };

  (depth_ref=max_depth and depth_rel=greater_than) and threshold_crossed=true)): { vehicle_is_too_deep };

end Protocol
depth_rel=equal) and intevent_stat=got_depth_event_sent_to_msctrl)) or 
((Action=no_mcie_act and ! Depth_control_instruction_parser.Action=send_dctrl_demand_in_surface_mode) and ! Depth_control_instruction_parser.Action=send_dctrl_demand_in_depth_mode) and ((depth_ref=dctrl_demand and depth_rel=equal) and intevent_stat=vehicle_got_depth) and threshold_crossed=false)) and 
VehicleTooDeep.Action = inject;

depth_ref = surface_depth and depth_rel = equal and intevent_stat = vehicle_on_surface if ((Action=send_on_surface_event_to_msctrl and ((depth_ref=surface_depth and depth_rel=equal) and inter

VehicleTooDeep.Action = inject;

depth_ref = max_depth and depth_rel = greater_than and intevent_stat = vehicle_too_deep and threshold_crossed = false if 
((depth_ref=surface_depth and depth_rel=equal) and interevent_stat=vehicle_got_depth) and 
threshold_crossed=true) or 
((depth_ref=dctrl_demand and depth_rel=equal) and interevent_stat=vehicle_got_depth) and 
threshold_crossed=true) and 
VehicleTooDeep.Action = inject;

end Evolution
end Agent

Agent Gps_navigation

Vars:
gps_stat: { fix_acquired, fix_acquired_event_sent_to_mission_control, fix_not_acquired };
end Vars

RedStates:

end RedStates
Actions = { calculated_position_available, calculated_position_not_available,
gps_stat = fix_acquired if (Action=send_got_gps_event_to_msctrl and gps_stat=fix_acquired_event_sent_to_mission_control);
gps_stat = fix_acquired_event_sent_to_mission_control if (Action=calculated_position_available and (gps_stat=fix_not_acquired or gps_stat=fix_acquired)); end Evolution end Agent
Agent Homing_system
Vars:
beacon_signal: { off, on }; homing_status: { homing_acquired_event_sent_to_msctrl,
homing_lost_event_sent_to_msctrl, homing_signal_detected, homing_signal_lost,
homing_system_idle, homing_system_off }; end Vars
RedStates:
end RedStates Actions = { homing_acquired, no_hs_act, resume_idle_stage_of_homing_system,
send_homing_acquired_event_to_msctrl, send_homing_lost_event_to_msctrl,
signal_receiving_resumes }; Protocol: (((((beacon_signal=off and homing_status=homing_signal_detected) or (beacon_signal=off and homing_status=homing_system_idle)) or (beacon_signal=off and homing_status=homing_system_off)) or (beacon_signal=on and homing_status=homing_signal_detected)) or (beacon_signal=on and homing_status=homing_signal_lost)) or (beacon_signal=off and homing_status=homing_acquired_event_sent_to_msctrl)): { no_hs_act }; (beacon_signal=on and homing_status=homing_acquired_event_sent_to_msctrl): { send_homing_acquired_event_to_msctrl }; (beacon_signal=off and homing_status=homing_lost_event_sent_to_msctrl): { send_homing_lost_event_to_msctrl }; (beacon_signal=on and homing_status=homing_system_idle): { homing_acquired }; end Protocol Evolution:
beacon_signal = off and homing_status = homing_system_off if ((Action=no_hs_act and !Mission_script_execution.Action=mission_script_started) and (beacon_signal=off and homing_status=homing_system_off));
beacon_signal = off and homing_status = homing_system_idle if (((Mission_script_execution.Action=mission_script_started and (beacon_signal=off and homing_status=homing_system_off)) or (beacon_signal=off and homing_status=homing_signal_lost)) or (!Human.Action=human_sends_homing_signal and (beacon_signal=off and homing_status=homing_system_idle)));
beacon_signal = on and homing_status = homing_acquired_event_sent_to_msctrl if (Action=homing_acquired and (beacon_signal=on and homing_status=homing_system_idle)) and !HomingLostBeforeAcquired.Action = inject and !HomingLostAfterAcquired.Action = inject;
beacon_signal = off and homing_status = homing_acquired_event_sent_to_msctrl if (Action=homing_acquired and (beacon_signal=on and homing_status=homing_system_idle)) and !HomingLostBeforeAcquired.Action = inject and HomingLostAfterAcquired.Action = inject;
beacon_signal = on and homing_status = homing_signal_detected if (beacon_signal=off and homing_status = homing_signal_lost if (Action=send_homing_lost_event_to_msctrl and (beacon_signal=off and homing_status=homing_system_idle)) and HomingLostBeforeAcquired.Action = inject; beacon_signal = off and homing_status = homing_signal_lost if (beacon_signal=off and homing_status = homing_signal_detected if (beacon_signal=off and homing_status = homing_signal_lost if (beacon_signal=off and homing_status = homing_system_idle)) and home
homing_status=homing_acquired_event_sent_to_msctrl) ) or (beacon_signal=on and homing_status=homing_signal_detected) and !HomingLostBeforeAcquired.Action = inject and !HomingLostAfterAcquired.Action = inject;
  beacon_signal = off and homing_status = homing_signal_detected if ((Action=send_homing_acquired_event_to_msctrl and (beacon_signal=on and homing_status=homing_acquired_event_sent_to_msctrl)) or (beacon_signal=on and homing_status=homing_signal_detected)) and !HomingLostBeforeAcquired.Action = inject and HomingLostAfterAcquired.Action = inject;
  beacon_signal = off and homing_status = homing_signal_detected if ((beacon_signal=off and homing_status=homing_acquired_event_sent_to_msctrl) or (beacon_signal=off and homing_status=homing_signal_detected));
beacon_signal = on and homing_status = homing_system_idle if ((beacon_signal=on and homing_status=homing_signal_lost) or (Human.Action=human_sends_homing_signal and (beacon_signal=off and homing_status=homing_system_idle))) and !HomingLostBeforeAcquired.Action = inject and !HomingLostAfterAcquired.Action = inject;
  beacon_signal = off and homing_status = homing_system_idle if ((beacon_signal=on and homing_status=homing_signal_lost) or (Human.Action=human_sends_homing_signal and (beacon_signal=off and homing_status=homing_system_idle))) and HomingLostBeforeAcquired.Action = inject;
  beacon_signal = off and homing_status = homing_system_idle if ((beacon_signal=on and homing_status=homing_signal_lost) or (Human.Action=human_sends_homing_signal and (beacon_signal=off and homing_status=homing_system_idle))) and HomingLostBeforeAcquired.Action = inject;
end Evolution
end Agent

Agent Mission_timer_instruction_parser
Vars:
  state: { idle, set };
end Vars
RedStates:
end RedStates
Actions = { no_mtip_act, set_mission_timer };
Protocol:
  state=idle: { no_mtip_act };
  state=set: { set_mission_timer };
end Protocol
Evolution:
  state = idle if (((Action=no_mtip_act and !Mission_script_execution.Action=issue_first_me_join_instr) and state=idle) or (Action=set_mission_timer and state=set));
  state = set if (Mission_script_execution.Action=issue_first_me_join_instr and state=idle);
end Evolution
end Agent

Agent Power_node
Vars:
  pwr_ref: { safe_level };
  pwr_rel: { greater_than, less_than };
  pwr_stat: { idle, power_low, power_low_event_sent_to_msctrl, power_normal };
end Vars
RedStates:
end RedStates
Actions = { battery_voltage_fallen_below_set_level, battery_voltage_stable_above_set_level, battery_voltage_stays_below_set_level, no pn_act, send_power_low_event_to_msctrl };
Protocol:
  ( (pwr_ref=safe_level and pwr_rel=greater_than) and pwr_stat=idle) or((pwr_ref=safe_level and pwr_rel=greater_than) and pwr_stat=power_normal)): { no pn_act };
  (pwr_ref=safe_level and pwr_rel=less_than) and pwr_stat=power_low): {
battery_voltage_stays_below_set_level);
  (pwr_ref=safe_level and pwr_rel=less_than and pwr_stat=power_low_event_sent_to_msctrl);
  (pwr_ref=safe_level and pwr_rel=less_than and pwr_stat=power_normal):
  { battery_voltage_fallen_below_set_level);
end Protocol

Evolution:

pwr_ref = safe_level and pwr_rel = greater_than and pwr_stat = idle if
  ((Action=no_pn_act and !Mission_script_execution.Action=mission_script_started)
  and ((pwr_ref=safe_level and pwr_rel=greater_than) and pwr_stat=idle)) and
  !PowerLow.Action = inject;
  
pwr_ref = safe_level and pwr_rel = less_than and pwr_stat = idle if
  ((Action=no_pn_act and !Mission_script_execution.Action=mission_script_started)
  and ((pwr_ref=safe_level and pwr_rel=greater_than) and pwr_stat=idle)) and
  !PowerLow.Action = inject;
  
pwr_ref = safe_level and pwr_rel = greater_than and pwr_stat = power_normal
  if ((Mission_script_execution.Action=mission_script_started and
       (pwr_ref=safe_level and pwr_rel=greater_than) and pwr_stat=idle)) or
  (pwr_ref=safe_level and pwr_rel=less_than and pwr_stat=power_normal)
  and !PowerLow.Action = inject;

pwr_ref = safe_level and pwr_rel = greater_than and pwr_stat = idle if
  (pwr_ref=safe_level and pwr_rel=less_than and pwr_stat=power_low)
  or
  (Action=battery_voltage_stays_below_set_level and
   (pwr_ref=safe_level and pwr_rel=less_than) and pwr_stat=power_low_event_sent_to_msctrl));
end Evolution

end Agent

Agent Mission_script_execution
Vars:

dctrl_mode: { depth, sternplane_angle, surface };
line_timer: { off, on };
mctrl_mode: { power };
mse_stat: { first_me_event_proc, first_me_ongoing, fourth_me_event_proc, fourth_me_ongoing, homing_underway_event_proc, homing_underway_ongoing, ms_default_state, ms_start, mte1_event_proc, mte1_ongoing, mte2_event_proc, mte2_ongoing, second_me_event_proc, second_me_ongoing, third_me_event_proc, third_me_ongoing };
pctrl_mode: { position, track_follow };
end Vars

end RedStates

Actions = { issue_first_me_joint_instr, issue_fourth_me_joint_instr, issue_homing_underway_joint_instr, issue_mte2_joint_instr, issue_second_me_joint_instr, issue_third_me_joint_instr, maintain_first_me, maintain_fourth_me, maintain_homing_underway, maintain_mte1, maintain_mte2, maintain_second_me, maintain_third_me, mission_script_started, no_mse_act };

Protocol:

  mse_stat=ms_default_state:
  { no_mse_act };
  mse_stat=ms_start:
  { mission_script_started };
  { { (dctrl_mode=depth and line_timer=off) and mctrl_mode=power } and
    mse_stat=first_me_event_proc and pctrl_mode=position }:
    { issue_first_me_joint_instr };  
  { { (dctrl_mode=depth and line_timer=off) and mctrl_mode=power } and
    mse_stat=mte2_event_proc and pctrl_mode=track_follow }:
    { issue_mte2_joint_instr };

  { { (dctrl_mode=depth and line_timer=off) and mctrl_mode=power } and
    mse_stat=first_me_ongoing and pctrl_mode=position }:
    { maintain_first_me };
  { { (dctrl_mode=depth and line_timer=off) and mctrl_mode=power } and
    mse_stat=second_me_event_proc and pctrl_mode=position }:
    { issue_second_me_joint_instr };
  { ((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
mse_stat=mte1_ongoing) and pctrl_mode=track_follow): { maintain_mte1 };
    (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
    mse_stat=mte2_event_proc and pctrl_mode=position): { issue_mte2_joint_instr };
    (((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and
    mse_stat=second_me_ongoing) and pctrl_mode=position): { maintain_second_me };
    (((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and
    mse_stat=third_me_event_proc) and pctrl_mode=track_follow): { issue_third_me_joint_instr };
    (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
    mse_stat=homming_underway_event_proc) and pctrl_mode=position): { maintain_third_me };
    (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
    mse_stat=fourth_me_ongoing) and pctrl_mode=track_follow): { maintain_fourth_me };
end Protocol

Evolution:
    mse_stat = ms_default_state if ((Action=no_mse_act and !
Gps_navigation.Action=send_got_gps_event_to_msctrl) and
mse_stat=ms_default_state);    
    mse_stat = ms_start if (Gps_navigation.Action=send_got_gps_event_to_msctrl
and mse_stat=ms_default_state);    
    dctrl_mode = surface and line_timer = on and mctrl_mode = power and mse_stat =
first_me_event_proc and pctrl_mode = position if (Action=mission_script_started
and mse_stat=ms_start);    
    dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat =
second_me_event_proc and pctrl_mode=track_follow if
    (((Power_node.Action=send_power_low_event_to_mscctrl and !
Homing_system.Action=send_homing_acquired_event_to_mscctrl) and
    (((((dctrl_mode=surface and line_timer=on) and mctrl_mode=power) and
    mse_stat=first_me_ongoing) and pctrl_mode=position) or
    (((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and
    mse_stat=second_me_event_proc) and pctrl_mode=position)) or
    (((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and
    mse_stat=second_me_ongoing) and pctrl_mode=position )) or
    (((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and
    mse_stat=third_me_event_proc) and pctrl_mode=track_follow)) or
    (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
    mse_stat=fourth_me_event_proc) and pctrl_mode=track_follow)) or
    (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
    mse_stat=fourth_me_ongoing) and pctrl_mode=track_follow))) or
    (((Avoidance_timer.Action=at_sends_avoidance_failed_event_to_mscctrl and !
Power_node.Action=send_power_low_event_to_mscctrl) and !
Homing_system.Action=send_homing_acquired_event_to_mscctrl) and !
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_mscctrl) and !
Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_mscctrl) and !
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msc
ctrl and ! Power_node.Action=send_power_low_event_to_mscctrl) and !
Homing_system.Action=send_homing_acquired_event_to_mscctrl) and !
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_mscctrl) and !
Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_mscctrl) and !
Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and 
((((((((((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=fourth_me_ongoing) and pctrl_mode=track_follow) or (((dctrl_mode=depth 
and line_timer=off) and mctrl_mode=power) and mctrl_mode=track_follow)) or 
((((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and 
ms_stat=fourth_me_event_proc) and pctrl_mode=track_follow)) or 
((((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and 
ms_stat=third_me_ongoing) and pctrl_mode=track_follow)) or 
((((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and 
ms_stat=third_me_event_proc) and pctrl_mode=track_follow)) or 
((((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=second_me_ongoing) and pctrl_mode=track_follow)) or 
((((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=second_me_event_proc) and pctrl_mode=track_follow)) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=second_me_ongoing) and pctrl_mode=position)) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=second_me_event_proc) and pctrl_mode=position)) or 
((((dctrl_mode=surface and line_timer=on) and mctrl_mode=power) and 
ms_stat=first_me_ongoing) and pctrl_mode=position) ))) or 
((((dctrl_mode=surface and line_timer=on) and mctrl_mode=power) and 
ms_stat=first_me_event_proc) and pctrl_mode=position) ))) or 
((((dctrl_mode=surface and line_timer=on) and mctrl_mode=power) and 
ms_stat=first_me_ongoing) and pctrl_mode=position))) or 
(((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=fourth_me_ongoing) and pctrl_mode=track_follow) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=fourth_me_event_proc) and pctrl_mode=track_follow)) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=third_me_ongoing) and pctrl_mode=track_follow)) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=third_me_event_proc) and pctrl_mode=track_follow)) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=second_me_ongoing) and pctrl_mode=track_follow)) or 
((((dctrl_mode=sternplane_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=second_me_event_proc) and pctrl_mode=track_follow)) or 
((((dctrl_mode=surface and line_timer=on) and mctrl_mode=power) and 
ms_stat=first_me_ongoing) and pctrl_mode=position))))

dctrl_mode = surface and line_timer = on and mctrl_mode = power and ms_stat 
= first_me_ongoing and pctrl_mode = position if 
((Action=issue_first_me_joint_instr and (((dctrl_mode=surface and line_timer=on) 
and mctrl_mode=power) and ms_stat=first_me_event_proc) and pctrl_mode=position)) or 
((((((Action=maintain_first_me and ! 
Line_timer.Action=send_line_timeout_event_to_msctrl) and ! 
Power_node.Action=send_power_low_event_to_msctrl) and ! 
Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! 
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl 
and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and ! 
Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and ! 
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! 
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) 
and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and ! 
Power_node.Action=send_power_low_event_to_msctrl) and ! 
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! 
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) 
and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and ! 
Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and ! 
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! 
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) 
and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and !
Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl and 
((((dctrl_mode=surface and line_timer=on) and mctrl_mode=power) and 
ms_stat=first_me_ongoing) and 
pctrl_mode=position));

dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = 
mte1_ongoing and pctrl_mode = track_follow if (((Action=issue_mte1_joint_instr 
and Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=mte1_event_proc) and 
pctrl_mode=track_follow) or ((Action=maintain_mte1 
and Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=mte1_ongoing) and pctrl_mode=track_follow));

dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = 
mte2_event_proc and pctrl_mode = position if 
(Homing_system.Action=send_homing_lost_event_to_msctrl and 
((((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
mse_stat=mte1_ongoing) and pctrl_mode=position) or 
(((dctrl_mode=depth and line_timer=off) and 
mctrl_mode=power) and 
ms_stat=homing_underway_event_proc) and 
pctrl_mode=position));

dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = 
second_me_ongoing and pctrl_mode = position if 
(((Action=issue_second_me_joint_instr and 
Power_node.Action=send_power_low_event_to_msctrl) and 
! Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and 
! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) 
and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and 
! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and 
(dctrl_mode=second_me_event_proc and mte1_mode=power) and 
ms_stat=send_second_me_ongoing and pctrl_mode=position));

dctrl_mode = depth and line_timer = on and mctrl_mode = power and mse_stat = 
third_me_event_proc and pctrl_mode = track_follow if 
(((Mission_control_internal_events.Action=send_vehicle_dived_event_to_msctrl) and 
! Power_node.Action=send_power_low_event_to_msctrl) and 
! Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and 
! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) 
and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and 
! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and 
((dctrl_mode=second_me_angle and line_timer=off) and mctrl_mode=power) and 
ms_stat=send_second_me_ongoing and pctrl_mode=position));

dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = 
mte2_ongoing and pctrl_mode = position if 
(((Action=issue_mte2_joint_instr and 
Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
(mse_stat=mte1_ongoing) and pctrl_mode=position) or 
(((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
mse_stat=mte2_event_proc) and pctrl_mode=position)) or 
((Action=maintain_mte2 and 
Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=mte2_ongoing) and pctrl_mode=position));

dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = 
homing_underway_event_proc and pctrl_mode = position if 
(Homing_system.Action=send_homing_acquired_event_to_msctrl) and 
(((((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
mse_stat=mte2_ongoing) and pctrl_mode=position) or 
((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and 
ms_stat=first_me_ongoing) and 
pctrl_mode=position)) or 
(((dctrl_mode=depth and line_timer=off) and 
mctrl_mode=power) and 
mse_stat=send_second_me_event_proc) and pctrl_mode=position)) or 
((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and 
ms_stat=send_second_me_ongoing) and pctrl_mode=position));
mctrl_mode=power) and mse_stat=mte1_ongoing) and pctrl_mode=track_follow)) or (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and mse_stat=fourth_me_event_proc) and pctrl_mode=track_follow)) or (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and mse_stat=fourth_me_ongoing) and pctrl_mode=track_follow)) or (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and mse_stat=third_me_ongoing) and pctrl_mode=track_follow));
dctrl_mode = depth and line_timer = on and mctrl_mode = power and mse_stat = third_me_ongoing and pctrl_mode = track_follow if (((((((Action=issue_third_me_joint_instr and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and (((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and mse_stat=third_me_event_proc) and pctrl_mode=track_follow)) or ((((((((Action=maintain_third_me and ! Position_control_operation.Action=send_go t_position_event_to_msctrl) and ! Line_timer.Action=send_line_timeout_event_to_msctrl) and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and (((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and mse_stat=third_me_ongoing) and pctrl_mode=track_follow));
dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = fourth_me_event_proc and pctrl_mode = track_follow if (((((((Retreat_track_length_monitor.Action=send_collision_avoided_event_to_msctrl) and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) or ((((((((Position_control_operation.Action=send_get_position_event_to_msctrl) and ! Line_timer.Action=send_line_timeout_event_to_msctrl) and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl)) or ((((((((Line_timer.Action=send_line_timeout_event_to_msctrl) and ! Retrace_track_length_monitorn.Action=send_collision_avoided_event_to_msctrl) and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl)) and (((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and mse_stat=third_me_ongoing) and pctrl_mode=track_follow));
dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = homing_underway_ongoing and pctrl_mode = position if (((Action=issue_homing_underway_joint_instr and ! Homing_system.Action=send_homing_lost_event_to_msctrl) and (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and mse_stat=homing_underway_event_proc) and pctrl_mode=position)) or (((Action=maintain_homing_underway and ! Homing_system.Action=send_homing_lost_event_to_msctrl) and ! Retreat_track_path_array.Action=pta_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl)) or (((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and mse_stat=homing_underway_ongoing) and pctrl_mode=position));
dctrl_mode = depth and line_timer = off and mctrl_mode = power and mse_stat = fourth_me_ongoing and pctrl_mode = track_follow if (((((Action=issue_fourth_me_joint_instr and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl)) or ((((((((Line_timer.Action=send_line_timeout_event_to_msctrl) and ! Retrace_track_length_monitorn.Action=send_collision_avoided_event_to_msctrl) and ! Power_node.Action=send_power_low_event_to_msctrl) and ! Homing_system.Action=send_homing_acquired_event_to_msctrl) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl)) and (((dctrl_mode=depth and line_timer=on) and mctrl_mode=power) and mse_stat=third_me_ongoing) and pctrl_mode=track_follow));

---
Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and !
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl)
and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl) and !
((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
mse_stat=fourth_me_event_proc) and pctrl_mode=track_follow)) or
((((dctrl_mode=depth and line_timer=off) and mctrl_mode=power) and
mse_stat=fourth_me_ongoing) and pctrl_mode=track_follow)));
end Evolution
end Agent

Agent Human
Vars:
    human_stat: { observing_auv, sending_homing };
end Vars

end RedStates
Actions = { human_sends_homing_signal, no_human_act };
Protocol:
    human_stat=observing_auv: { no_human_act };
    human_stat=sending_homing: { human_sends_homing_signal };
end Protocol

Evolution:
    human_stat = sending_homing if ((Environment.Action=auv_appears_faulty and
human_stat=observing_auv) or human_stat=sending_homing);
end Evolution
end Agent

Agent Motor_control_instruction_parser
Vars:
    state: { idle, power_mode };
end Vars

end RedStates
Actions = { no_mcip_act, send_mctrl_demand_in_power_mode };
Protocol:
    state=idle: { no_mcip_act };
    state=power_mode: { send_mctrl_demand_in_power_mode };
end Protocol

Evolution:
    state = idle if (((((Action=no_mcip_act and !
Mission_script_execution.Action=issue_first_me_joint_instr) and !
Mission_script_execution.Action=issue_second_me_joint_instr) and !
Mission_script_execution.Action=issue_third_me_joint_instr) and !
Mission_script_execution.Action=issue_fourth_me_jointInstr) and !
Mission_script_execution.Action=issue_mte1_joint_instr) and !
Mission_script_execution.Action=issue_mte2_joint_instr) and !
Mission_script_execution.Action=issue_homing_underway_joint_instr) and
state=idle) or (Action=send_mctrl_demand_in_power_mode and state=power_mode));
    state = power_mode if (((((Mission_script_execution.Action=issue_first_me_joint_instr
or Mission_script_execution.Action=issue_fourth_me_joint_instr) or
Mission_script_execution.Action=issue_homing_underway_joint_instr) or
Mission_script_execution.Action=issue_mte1_joint_instr) or
Mission_script_execution.Action=issue_mte2_joint_instr) or
Mission_script_execution.Action=issue_second_me_joint_instr) or
Mission_script_execution.Action=issue_third_me_joint_instr) and state=idle);
end Evolution
end Agent

Agent Position_control_instruction_parser
Vars:
    state: { idle, position_mode, track_follow_mode };
end Vars
RedStates:
end RedStates
Actions = { no_pcip_act, send_pctrl_demand_in_position_mode, send_pctrl_demand_in_track_follow_mode }; Protocol:
  state=idle: { no_pcip_act }
  state=position_mode: { send_pctrl_demand_in_position_mode }
  state=track_follow_mode: { send_pctrl_demand_in_track_follow_mode }
end Protocol
Evolution:
  state = idle if (((((((((Action=no_pcip_act and ! Mission_script_execution.Action=issue_first_me_joint_instr) and ! Mission_script_execution.Action=issue_second_me_joint_instr) and ! Mission_script_execution.Action=issue_third_me_joint_instr) and ! Mission_script_execution.Action=issue_fourth_me_joint_instr) and ! Mission_script_execution.Action=issue_mte1_joint_instr) and ! Mission_script_execution.Action=issue_mte2_joint_instr) and ! Mission_script_execution.Action=issue_homing_underway_joint_instr) and state=idle) or (Action=send_pctrl_demand_in_track_follow_mode and state=track_follow_mode)) or (Action=send_pctrl_demand_in_position_mode and state=position_mode));
  state = position_mode if ((((Mission_script_execution.Action=issue_first_me_joint_instr or Mission_script_execution.Action=issue_homing_underway_joint_instr) or Mission_script_execution.Action=issue_mte2_joint_instr) or Mission_script_execution.Action=issue_second_me_joint_instr) and state=idle);
  state = track_follow_mode if (((Mission_script_execution.Action=issue_fourth_me_joint_instr or Mission_script_execution.Action=issue_mte1_joint_instr) or Mission_script_execution.Action=issue_mte2_joint_instr) or Mission_script_execution.Action=issue_third_me_joint_instr) and state=idle);
end Evolution
end Agent

Agent Depth_control_instruction_parser
Vars:
  state: { depth_mode, idle, sternplane_angle_mode, surface_mode }
end Vars
RedStates:
end RedStates
Actions = { no_dcip_act, send_dctrl_demand_in_depth_mode, send_dctrl_demand_in_sternplane_angle_mode, send_dctrl_demand_in_surface_mode }; Protocol:
  state=idle: { no_dcip_act }
  state=surface_mode: { send_dctrl_demand_in_surface_mode }
  state=depth_mode: { send_dctrl_demand_in_depth_mode }
  state=sternplane_angle_mode: { send_dctrl_demand_in_sternplane_angle_mode }
end Protocol
Evolution:
  state = idle if (((((((((Action=no_dcip_act and ! Mission_script_execution.Action=issue_first_me_joint_instr) and ! Mission_script_execution.Action=issue_second_me_joint_instr) and ! Mission_script_execution.Action=issue_third_me_joint_instr) and ! Mission_script_execution.Action=issue_fourth_me_joint_instr) and ! Mission_script_execution.Action=issue_mte2_joint_instr) and ! Mission_script_execution.Action=issue_homing_underway_joint_instr) and state=idle) or (Action=send_dctrl_demand_in_sternplane_angle_mode and state=sternplane_angle_mode)) or (Action=send_dctrl_demand_in_depth_mode and state=depth_mode)) or (Action=send_dctrl_demand_in_surface_mode and state=surface_mode));
  state = surface_mode if (Mission_script_execution.Action=issue_first_me_joint_instr and state=idle);
  state = depth_mode if ((((Mission_script_execution.Action=issue_fourth_me_joint_instr or Mission_script_execution.Action=issue_mte1_joint_instr) or Mission_script_execution.Action=issue_mte2_joint_instr) or Mission_script_execution.Action=issue_homing_underway_joint_instr) or Mission_script_execution.Action=issue_third_me_joint_instr) and state=idle);
  state = sternplane_angle_mode if (Mission_script_execution.Action=issue_second_me_joint_instr and state=idle);
end Evolution
end Agent
Agent Line_timer_instruction_parser
Vars:
    state: { idle, set }; 
end Vars 
RedStates:
end RedStates 
Actions = { no_ltip_act, set_line_timer }; 
Protocol: 
    state=idle: { no_ltip_act }; 
    state=set: { set_line_timer }; 
end Protocol 
Evolution:
    state = idle if (((Action=no_ltip_act and !
Mission_script_execution.Action=issue_first_me_joint_instr) and !
Mission_script_execution.Action=issue_third_me_joint_instr) and state=idle) or
    (Action=set_line_timer and state=set));
    state = set if ((Mission_script_execution.Action=issue_first_me_joint_instr
or Mission_script_execution.Action=issue_third_me_joint_instr) and state=idle);
end Evolution 
end Agent 
Agent Line_timer
Vars: 
    state: { lt_event_sent_to_msctrl, lt_expired, lt_idling, lt_initialized, 
    lt_operation_enabled }; 
end Vars 
end RedStates 
Actions = { lt_reaching_timeout, lt_reset, no_ltd_act, schedule_lt_to_expire, 
send_line_timeout_event_to_msctrl }; 
Protocol: 
    state=lt_idling: { no_ltd_act }; 
    state=lt_initialized: { schedule_lt_to_expire }; 
    state=lt_operation_enabled: { lt_reaching_timeout }; 
    state=lt_event_sent_to_msctrl: { send_line_timeout_event_to_msctrl }; 
    state=lt_expired: { lt_reset }; 
end Protocol 
Evolution:
    state = lt_idling if (((Action=no_ltd_act and !
Line_timer_instruction_parser.Action=set_line_timer) and state=lt_idling) or 
(Line_timer_instruction_parser.Action=set_line_time
r and state=lt_idling));
    state = lt_initialized if (Line_timer_instruction_parser.Action=set_line_time
r and state=lt_idling);
    state = lt_operation_enabled if (Action=schedule_lt_to_expire and 
state=lt_initialized);
    state = lt_event_sent_to_msctrl if (Action=lt_reaching_timeout and 
state=lt_operation_enabled);
    state = lt_expired if (Action=send_line_timeout_event_to_msctrl and 
state=lt_event_sent_to_msctrl);
end Evolution 
end Agent 
Agent New_demand_detection
Vars: 
    state: { demand, has_retreating, nodemand }; 
end Vars 
end RedStates 
Actions = { new_demand_detection_occures, no_ndd_act }; 
Protocol: 
    state=demand: { new_demand_detection_occures }; 
    (state=nodemand or state=has_retreating): { no_ndd_act }; 
end Protocol 
Evolution:
    state = demand if ((((Position_control_instruction_parser.Action=send_pctrl_de
mand_in_position_mode or Position_control_instruction_parser.Action=send_pctrl_de
mand_in_track_follow_mode) and state=nodemand) or
(((Action=new_demand_detection_occures and ! Horizontal_avoidance_strategy.Action=start_retreating) and state=demand));
  state = nodemand if (((Action=no_ndd_act and ! Position_control_instruction_parser.Action=start_send_pctrl_demand_in_track_follow_mode) and state=nodemand) or state=has_retreating);
end Evolution
end Agent

Agent Obstacle_clearance_meter
Vars:
  state: { clearance_distance_accumulated, clearance_distance_covered, has_retreating, has_waiting, idling, obstacle_avoided_event_sent_to_msctrl, reset};
end Vars
end RedStates

Actions = { add_travelled_distance_segment_to_ocm_accumulated_distance, clearance_distance_has_been_covered, no_ocm_act, ocm_accumulated_distance_reaches_configured_value, send_obstacle_avoided_event_to_msctrl, set_clearance_run_distance_to_zero};
Protocol:
  state=reset: { set_clearance_run_distance_to_zero };  
  ((state=idling or state=has_retreating) or state=has_waiting): { no_ocm_act };
  state=clearance_distance_accumulated: { add_travelled_distance_segment_to_ocm_accumulated_distance, ocm_accumulated_distance_reaches_configured_value };  
  state=obstacle_avoided_event_sent_to_msctrl: { send_obstacle_avoided_event_to_msctrl };  
  state=clearance_distance_covered: { clearance_distance_has_been_covered };  
end Protocol
Evolution:
  state = reset if (Horizontal_avoidance_strategy.Action=start_trying_new_track and state=idling);
  state = idling if (((Action=no_ocm_act and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=idling) or (state=has_waiting or state=has_retreating));
  state = clearance_distance_accumulated if (((Action=set_clearance_run_distance_to_zero and state=reset) or (((Action=add_travelled_distance_segment_to_ocm_accumulated_distance and ! Horizontal_avoidance_strategy.Action=start_retreating) and ! Horizontal_avoidance_strategy.Action=start_waiting) and state=clearance_distance_accumulated)));  
  state = obstacle_avoided_event_sent_to_msctrl if (((Action=ocm_accumulated_distance_reaches_configured_value and ! Horizontal_avoidance_strategy.Action=start_retreating) and ! Horizontal_avoidance_strategy.Action=start_waiting) and ! Horizontal_avoidance_strategy.Action=start_waiting) and state=clearance_distance_accumulated);
  state = clearance_distance_covered if (((Action=send_obstacle_avoided_event_to_msctrl and ! Horizontal_avoidance_strategy.Action=start_retreating) and ! Horizontal_avoidance_strategy.Action=start_waiting) and ! Horizontal_avoidance_strategy.Action=start_waiting) and ! Horizontal_avoidance_strategy.Action=start_retreating and ! Horizontal_avoidance_strategy.Action=start_waiting and state=clearance_distance_covered);  
  state = has_retreating if (Horizontal_avoidance_strategy.Action=start_retreating and ((state=clearance_distance_covered or state=obstacle_avoided_event_sent_to_msctrl) or state=clearance_distance_accumulated));
  state = has_waiting if (Horizontal_avoidance_strategy.Action=start_waiting and ((state=clearance_distance_covered or state=obstacle_avoided_event_sent_to_msctrl) or state=clearance_distance_accumulated));
end Evolution
end Agent
Agent Avoidance_timer
Vars:
  at_stat: { avoidance_failed_event_sent_to_msctrl_by_at, idling, 
operation_enabled, timeout_exceeded }; 
  timeout: boolean;
end Vars
RedStates:

end RedStates
Actions = { at_overflows, at_sends_avoidance_failed_event_to_msctrl, no_at_act };

Protocol:
  {((at_stat=idling and timeout=false) or (at_stat=operation_enabled and 
timeout=false)) or (at_stat=operation_enabled and timeout=true)):
    { no_at_act };
  at_stat=avoidance_failed_event_sent_to_msctrl_by_at: { 
    at_sends_avoidance_failed_event_to_msctrl }; 
  at_stat=timeout_exceeded: { at_overflows };
end Protocol
Evolution:
  at_stat = idling and timeout = false if ((Action=no_at_act and !
Horizontal_avoidance_strategy.Action=start_retreating) and (at_stat=idling and
timeout=false)) and 
Avoidance_timer_failed.Action = inject;
  at_stat = idling and timeout = true if ((Action=no_at_act and !
Horizontal_avoidance_strategy.Action=start_retreating) and (at_stat=idling and
timeout=false)) and (Vertical_avoidance_strategy.Action=start_waiting and
(at_stat=operation_enabled and timeout=false)) and Avoidance_timer_failed.Action
= inject;
  at_stat = operation_enabled and timeout = false if 
((Horizontal_avoidance_strategy.Action=start_retreating and (at_stat=idling and
timeout=false)) or (! Horizontal_avoidance_strategy.Action=start_waiting and
(at_stat=operation_enabled and timeout=false)))
  at_stat = operation_enabled and timeout = true if 
((Horizontal_avoidance_strategy.Action=start_retreating and (at_stat=idling and
timeout=false)) or (! Horizontal_avoidance_strategy.Action=start_waiting and
(at_stat=operation_enabled and timeout=false)))
end Evolution

end Agent

Agent Retry_track_counter
Vars:
  allowed_no_rtc_reached: boolean;
  rtc_stat: { avoidance_failed_event_sent_to_msctrl_by_rtc, counter_overflow, 
idling, number_of_count_accumulated };
end Vars
RedStates:

end RedStates
Actions = { allowed_number_of_rtc_exceeded, no_rtc_act, 
rtc_sends_avoidance_failed_event_to_msctrl };

Protocol:
  ((((allowed_no_rtc_reached=false and rtc_stat=idling) or 
(allowed_no_rtc_reached=false and rtc_stat=number_of_count_accumulated)) or 
(allowed_no_rtc_reached=true and rtc_stat=number_of_count_accumulated)):
    { no_rtc_act };
  rtc_stat=avoidance_failed_event_sent_to_msctrl_by_rtc: { 
    rtc_sends_avoidance_failed_event_to_msctrl }; 
  rtc_stat=counter_overflow: { allowed_number_of_rtc_exceeded };
end Protocol
Evolution:
  allowed_no_rtc_reached = false and rtc_stat = idling if (((Action=no_rtc_act
and ! Horizontal_avoidance_strategy.Action=start_retreating) and
(allowed_no_rtc_reached=false and rtc_stat=idling)) or
(Horizontal_avoidance_strategy.Action=start_waiting and
(allowed_no_rtc_reached=false and rtc_stat=number_of_count_accumulated)) and
!RetryTrackAvoidanceFail.Action = inject;
allowed_no_rtc_reached = true and rtc_stat = idling if (((Action=no_rtc_act and
Horizontal_avoidance_strategy.Action=start_retreating) and
(allowed_no_rtc_reached=false and rtc_stat=idling)) or
Horizontal_avoidance_strategy.Action=start_waiting and
(allowed_no_rtc_reached=false and rtc_stat=number_of_count_accumulated))) and
RetryTrackAvoidanceFail.Action = inject;
allowed_no_rtc_reached = false and rtc_stat = number_of_count_accumulated if
((Action=no_rtc_act and (allowed_no_rtc_reached=false and
rtc_stat=number_of_count_accumulated)) or
Horizontal_avoidance_strategy.Action=start_retreating and
(allowed_no_rtc_reached=false and rtc_stat=idling)) or
(allowed_no_rtc_reached=false and rtc_stat=number_of_count_accumulated)))) and
!RetryTrackAvoidanceFail.Action = inject;
allowed_no_rtc_reached = true and rtc_stat = number_of_count_accumulated if
((Action=no_rtc_act and (allowed_no_rtc_reached=false and
rtc_stat=number_of_count_accumulated)) or
Horizontal_avoidance_strategy.Action=start_retreating and
(allowed_no_rtc_reached=false and rtc_stat=idling)) or
(allowed_no_rtc_reached=false and rtc_stat=number_of_count_accumulated)))) and
RetryTrackAvoidanceFail.Action = inject;
nrtc_stat = avoidance_failed_event_sent_to_msctrl_by_rtc if
(allowed_no_rtc_reached=true and rtc_stat=number_of_count_accumulated);
nrtc_stat = counter_overflow if ((Action=rtc_sends_avoidance_failed_event_to_msctrl
and rtc_stat=avoidance_failed_event_sent_to_msctrl_by_rtc) or
((Action=allowed_number_of_rtc_exceeded and rtc_stat=counter_overflow));
end Evolution
end Agent

Agent Retreat_track_length_monitor

Vars:
state: { check_availability_of_remaining_retreat_wp,
collision_avoided_event_sent_to_mission_control, has.trying_new_track, idling,
no_retreat_wp_left_for_retreat_track_ie_safe_waypoint_reached,
reaching_last_retreat_wp_after_demand_issue, retreat_wp_decreased_at_demand_issue
};
end Vars

RedStates:

end RedStates

Actions = { end_of_safe_retreat_track_is_reached,
idle_at_reaching_last_retreat_wp_after_demand_issue,
idle_at_retreat_wp_decreased_at_demand_issue, no_more_retreat_wp_are_available,
no_rtlm_act, send_collision_avoided_event_to_msctrl
};

Protocol:

(\text{state=idling or state=has_trying_new_track}): \{ \text{no_rtlm_act } \};
state=retreat_wp_decreased_at_demand_issue: \{
state=check_availability_of_remaining_retreat_wp: \{
state=reaching_last_retreat_wp_after_demand_issue: \{
state=collision_avoided_event_sent_to_mission_control: \{
state=no_retreat_wp_left_for_retreat_track_ie_safe_waypoint_reached: \{
end of_safe_retreat_track_is_reached
};
end Protocol

Evolution:

(\text{state=idling if }} (((Action=\text{no_rtlm_act and !}
\text{Horizontal_avoidance_strategy.Action=start_retreating) and state=idling}) or
\text{state=has\_trying\_new\_track});
state=retreat_wp_decreased_at_demand_issue if
(\text{Horizontal_avoidance_strategy.Action=start_retreating and state=idling}) or
(\text{Horizontal_avoidance_strategy.Action=start\_trying\_new\_track}) and !
\text{Retreat\_path\_array.Action=avctrl\_sends\_demand\_to\_pcctrl\_in\_track\_follow\_mode} and
\text{state=retreat_wp_decreased_at_demand_issue}) or
(\text{\text{\text{Position\_control\_operation.Action=vehicle\_arrives\_to\_position\_circle\_error\_region\_of\_retreat\_track\_target\_position}} and !
\text{Horizontal\_avoidance\_strategy.Action=start\_trying\_new\_track}) and !
Action=no_more_retreat_wp_are_available) and state=check_availability_of_remaining_retreat_wp));
    state = check_availability_of_remaining_retreat_wp if ((Retreat_path_array.Action=avctrl_sends_demand_to_pctrl_in_track_follow_mode and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=retreat_wp_decreased_at_demand_issue);
    state = reaching_last_retreat_wp_after_demand_issue if (((Action=no_more_retreat_wp_are_available and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=check_availability_of_remaining_retreat_wp) or (((Action=idle_at_reaching_last_retreat_wp_after_demand_issue and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and ! Position_control_operation.Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position and state=reaching_last_retreat_wp_after_demand_issue));
    state = collision_avoided_event_sent_to_mission_control if ((Position_control_operation.Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=collision_avoided_event_sent_to_mission_control); state = no_retreat_wp_left_for_retreat_track_ie_safe_waypoint_reached if (((Action=send_collision_avoided_event_to_msctrl and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=collision_avoided_event_sent_to_mission_control) or (! Horizontal_avoidance_strategy.Action=start_trying_new_track and state=no_retreat_wp_left_for_retreat_track_ie_safe_waypoint_reached));
    state = has_trying_new_track if (Horizontal_avoidance_strategy.Action=start_trying_new_track and ((((state=no_retreat_wp_left_for_retreat_track_ie_safe_waypoint_reached or state=retreat_wp_decreased_at_demand_issue) or state=collision_avoided_event_sent_to_mission_control) or state=check_availability_of_remaining_retreat_wp) or state=reaching_last_retreat_wp_after_demand_issue));
end Evolution
end Agent

Agent Retreat_path_array
Vars:
    rpa_empty: boolean;
    rpa_stat: { avoidance_failed_event_sent_to_msctrl_by_rpa, initialized, retrieving_available_waypoint, safe_position_saved_as_waypoint, standby_to_store_or_provide_safe_positions };
end Vars
RedStates:

end RedStates

Actions = { avctrl_sends_demand_to_pctrl_in_track_follow_mode, fill_content_of_rpa_with_vehicle_current_position, idle_at_safe_position_saved_as_waypoint, maintain_rpa_standby_mode, no_rpa_act, rpa_becomes_empty, rpa_sends_avoidance_failed_event_to_msctrl };

Protocol:
    (rpa_empty=true and rpa_stat=retrieving_available_waypoint): { rpa_becomes_empty };
    (rpa_empty=false and rpa_stat=initialized): { fill_content_of_rpa_with_vehicle_current_position };
    rpa_stat=avoidance_failed_event_sent_to_msctrl_by_rpa: { rpa_becomes_empty };
    rpa_stat=safe_position_saved_as_waypoint: { rpa_sends_avoidance_failed_event_to_msctrl };
    rpa_stat=idle_at_safe_position_saved_as_waypoint: { maintain_rpa_standby_mode };
end Protocol

Evolution:
    rpa_stat = avoidance_failed_event_sent_to_msctrl_by_rpa if (rpa_empty=true and rpa_stat=retrieving_available_waypoint);
    rpa_stat = safe_position_saved_as_waypoint if (((Action=fill_content_of_rpa_with_vehicle_current_position and (rpa_empty=false and rpa_stat=initialized)) or ((Action=fill_content_of_rpa_with_vehicle_current_position and ! Safe_position_meter.Action=start_spm_and_rpa_standby_operation) and ! Safe_position_meter.Action=start_spm_and_rpa_standby_operation) and rpa_stat=safe_position_saved_as_waypoint)) or

((Safe_position_meter.Action=safe_position_distance_has_been_covered and !Horizontal_avoidance_strategy.Action=start_retreating) and !Position_control_operation.Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position) and rpa_stat=standby_to_store_or_provide_safe_positions);

rpa_empty = false and rpa_stat = retrieving_available_waypoint if (!Action=avctrl_sends_demand_to_pctrl_in_track_follow_mode and (rpa_empty=false and rpa_stat=retrieving_available_waypoint)) or ((Horizontal_avoidance_strategy.Action=start_retreating or (Position_control_operation.Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position and !Horizontal_avoidance_strategy.Action=start_retreating) and rpa_stat=standby_to_store_or_provide_safe_positions)) and !RetreatPathArrayAvoidanceFailure.Action = inject;

rpa_empty = true and rpa_stat = retrieving_available_waypoint if ((!Action=avctrl_sends_demand_to_pctrl_in_track_follow_mode and (rpa_empty=false and rpa_stat=retrieving_available_waypoint)) or ((Horizontal_avoidance_strategy.Action=start_retreating or (Position_control_operation.Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position and !Horizontal_avoidance_strategy.Action=start_retreating)) and rpa_stat=standby_to_store_or_provide_safe_positions)) and !RetreatPathArrayAvoidanceFailure.Action = inject;

rpa_stat = standby_to_store_or_provide_safe_positions if (((Action=rpa_sends_avoidance_failed_event_to_msctrl and rpa_stat=avoidance_failed_event_sent_to_msctrl_by_rpa) or (Horizontal_avoidance_strategy.Action=start_waiting or Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=idling) or ((Action=maintain_rpa_standby_mode and !Horizontal_avoidance_strategy.Action=start_retreating) and !Position_control_operation.Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position) and !Safe_position_meter.Action=safe_position_distance_has_been_covered) and rpa_stat=standby_to_store_or_provide_safe_positions);

end Evolution

end Agent

Agent Safe_position_meter

Vars:

state: { has_retreating, idling, reset, safe_position_distance_accumulated, safe_position_distance_covered };

end Vars

end RedStates

Actions = { add_travelled_distance_segment_to_spm_accumulated_distance, no_spm_act, safe_position_distance_has_been_covered, spm_accumulated_distance_reaches_value_for_track_update, start_spm_and_rpa_standby_operation };

Protocol:

(state=idling or state=has_retreating): { no_spm_act };
state=reset: { start_spm_and_rpa_standby_operation };
state=safe_position_distance_accumulated: { add_travelled_distance_segment_to_spm_accumulated_distance, spm_accumulated_distance_reaches_value_for_track_update };
state=safe_position_distance_covered: { safe_position_distance_has_been_covered };

end Protocol

Evolution:

state = idling if (((Action=no_spm_act and !Horizontal_avoidance_strategy.Action=start_waiting) and !Horizontal_avoidance_strategy.Action=start_trying_new_track) and state=idling) or state=has_retreating);
state = reset if (((Horizontal_avoidance_strategy.Action=start_trying_new_track or Horizontal_avoidance_strategy.Action=start_waiting) and state=idling) or ((Action=safe_position_distance_has_been_covered and !Horizontal_avoidance_strategy.Action=start_retreating) and state=safe_position_distance_covered));
state = safe_position_distance_accumulated if
((Action=start_spm_and_rpa_standby_operation and !
Horizontal_avoidance_strategy.Action=start_retreating) and state=reset) or
((Action=add_travelled_distance_segment_to_spm_accumulated_distance and !
Horizontal_avoidance_strategy.Action=start_retreating) and
state=safe_position_distance_accumulated() and
state=safe_position_distance_covered if
((Action=spm_accumulated_distance_reaches_value_for_track_update and !
Horizontal_avoidance_strategy.Action=start_retreating) and
state=safe_position_distance_accumulated); state = has_retreating if (Horizontal_avoidance_strategy.Action=start_retreating
and ((state=reset or state=safe_position_distance_covered) or
state=safe_position_distance_accumulated));
end Evolution
end Agent

Agent Position_control_operation

Vars:

state: { at_mission_start, at_original_demand_target_position,
at_retreat_track_target_position, got_position_event_sent_to_msctrl,
head_towards_original_demand_target_position,
head_towards_retreat_track_target_position }; end Vars

RedStates:

Actions = { continue_with_loitering_at_original_demand_target_position,
idle_at_retreat_track_target_position, no_pc_act,
send_got_position_event_to_msctrl, vehicle_arrives_to_position_circle_error_region_of_original_demand_target_position,
vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position, vehicle_moves_to_original_demand_target_position,
vehicle_moves_to_retreat_track_target_position }; Protocol:
state=at_mission_start: { no_pc_act }
state=head_towards_original_demand_target_position: {
vehicle_arrives_to_position_circle_error_region_of_original_demand_target_position,
vehicle_moves_to_original_demand_target_position }
state=at_retreat_track_target_position: {
vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position,
vehicle_moves_to_retreat_track_target_position }
state=got_position_event_sent_to_msctrl: { send_got_position_event_to_msctrl }
state=at_original_demand_target_position: {
continue_with_loitering_at_original_demand_target_position }
state=at_retreat_track_target_position: {
idle_at_retreat_track_target_position }
end Protocol

Evolution:

state = at_mission_start if (((Action=no_pc_act and !
Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode)
and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode) and state=at_mission_start);
state = head_towards_original_demand_target_position if
(((Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode and
((state=at_mission_start or state=head_towards_original_demand_target_position) or
state=at_retreat_track_target_position))
or
(Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode
d and ((state=at_mission_start or state=head_towards_original_demand_target_position) or
state=at_retreat_track_target_position))
or
(((Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode
and state=at_mission_start or state=head_towards_original_demand_target_position) or
state=at_retreat_track_target_position))
or
(((Action=vehicle_moves_to_original_demand_target_position and !
Retreat_path_array.Action=avctrl_sends_demand_to_pctrl_in_track_follow_mode) and !
Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode)
or
(((Horizontal_avoidance_strategy.Action=start_trying_new_track and !
Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode)
and state=at_mission_start or state=head_towards_original_demand_target_position))
or
(((Horizontal_avoidance_strategy.Action=start_retreating and !
Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode)
and state=at_mission_start or state=head_towards_retreat_track_target_position) and
state=at_mission_start) or state=at_retreat_track_target_position) or state=at_mission_start);

state = at_retreat_track_target_position if
(((Retreat_path_array.Action=avctrl_sends_demand_to_pctrl_in_track_follow_mode and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode) or (Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode and state=head_towards_retreat_track_target_position)) or
(((Action=vehicle_moves_to_retreat_track_target_position and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode) and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode) and state=at_retreat_track_target_position));

state = got_position_event_sent_to_msctrl if
((Action=vehicle_arrives_to_position_circle_error_region_of_retreat_track_target_position and ! Horizontal_avoidance_strategy.Action=start_trying_new_track) and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode) and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode) and state=head_towards_retreat_track_target_position)) or
(((Action=continue_with_loitering_at_original_demand_target_position and ! Retreat_path_array.Action=avctrl_sends_demand_to_pctrl_in_track_follow_mode) and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_position_mode) and ! Position_control_instruction_parser.Action=send_pctrl_demand_in_track_follow_mode) and state=at_retreat_track_target_position));

end Agent

Agent Horizontal_avoidance_strategy

Vars:
    has_avoidance_failed_faulty: boolean;
    has_stat: { avoidance_failed_event_sent_to_msctrl_by_hca,
        initialize_components_for_retreating_state,
        initialize_components_for_trying_new_track_state,
        initialize_components_for_waiting_state, obstacle_detected_while_retreating,
        retreating, trying_new_track, waiting }; end Vars

end States:

Actions = { avoidance_process_is_idling,
    continue_initializing_components_for_trying_new_track_state,
    hca_sends_avoidance_failed_event_to_msctrl, maintain_retreating,
    maintain_trying_new_track, no_has_act, start_retreating, start_trying_new_track,
    start_waiting }; end Actions

Protocol:
    {has_avoidance_failed_faulty=true}
has_stat=avoidance_failed_event_sent_to_mission_control_by_hca): {
    hca_sends_avoidance_failed_event_to_msctrl;
    (has_avoidance_failed_faulty=false and
    has_stat=initialize_components_for_waiting_state): { start_waiting }
    has_stat=waiting: { avoidance_process_is_idling }
    has_stat=initialize_components_for_retreating_state: { start_retreating }
    has_stat=retreating: { maintain_retreting }
    has_stat=initialize_components_for_trying_new_track_state: {
    start_trying_new_track }
    (has_avoidance_failed_faulty=false and
    has_stat=avoidance_failed_event_sent_to_mission_control_by_hca): { no_has_act }
    has_stat=trying_new_track: { maintain_trying_new_track }
    has_stat=obstacle_detected_while_retreating: {
      continue_initializing_components_for_trying_new_track_state }
    end Protocol
  
  Evolution: has_avoidance_failed_faulty = false and has_stat =
  initialize_components_for_waiting_state if
  (((Obstacle_clearance_meter.Action=clearance_distance_has_been_covered or
  (New_demand_detection.Action=new_demand_detection_occures and !
  Obstacle_clearance_meter.Action=clearance_distance_has_been_covered)) and
  has_stat=trying_new_track) and !HorizontalAvoidanceFailure.Action = inject;
  has_avoidance_failed_faulty = true and has_stat =
  initialize_components_for_waiting_state if
  (((Obstacle_clearance_meter.Action=clearance_distance_has_been_covered or
  (New_demand_detection.Action=new_demand_detection_occures and !
  Obstacle_clearance_meter.Action=clearance_distance_has_been_covered)) and
  has_stat=trying_new_track) and HorizontalAvoidanceFailure.Action = inject;
  has_stat = waiting if ((Action=start_waiting and
  (has_avoidance_failed_faulty=false and has_stat=initialize_components_for_waiting
  _state)) or (((Action=avoidance_process_is_idling and !
  Depth_control_operation.Action=send_limited_headroom_to_pctrl) and !
  Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and
  has_stat=waiting));
  has_stat = initialize_components_for_retreating_state if
  (((Depth_control_operation.Action=send_limited_headroom_to_pctrl and !
  Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) or
  Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and
  has_stat=waiting) or (((((Depth_control_operation.Action=send_limited_headroom_to
  _pctrl and ! Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and
  ! Obstacle_clearance_meter.Action=clearance_distance_has_been_covered) and !
  New_demand_detection.Action=new_demand_detection_occures) or
  ((Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl and !
  Obstacle_clearance_meter.Action=clearance_distance_has_been_covered) and !
  New_demand_detection.Action=new_demand_detection_occures)) and
  has_stat=trying_new_track);
  has_stat = retreating if ((Action=start_retreating and
  has_stat=initialize_components_for_retreating_state) or
  (((Action=maintain_retreating and ! Depth_control_operation.Action=send_limited
  _headroom_to_pctrl) and ! Forward_looking_echosounder.Action=send_obstacle_ahead_to
  _pctrl) and ! Retreat_path_array.Action=rpa_becomes_empty) and !
  Retreat_track_length_monitor.Action=end_of_safe_retreat_track_is_reached) and
  has_stat=retreating));
  has_stat = initialize_components_for_trying_new_track_state if
  (((Retreat_track_length_monitor.Action=end_of_safe_retreat_track_is_reached and
  ! Retreat_path_array.Action=rpa_becomes_empty) or
  Retreat_path_array.Action=rpa_becomes_empty) and has_stat=retreating) or
  (Action=continue_initializing_components_for_trying_new_track_state and
  has_stat=obstacle_detected_while_retreating));
  has_avoidance_failed_faulty = false and has_stat =
  avoidance_failed_event_sent_to_mission_control_by_hca if
  (((Depth_control_operation.Action=send_limited_headroom_to_pctrl and !
  Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and !
  Retreat_path_array.Action=rpa_becomes_empty) and !
  Retreat_track_length_monitor.Action=end_of_safe_retreat_track_is_reached) or
  ((Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl and !
  Retreat_path_array.Action=rpa_becomes_empty) and !
  Retreat_track_length_monitor.Action=end_of_safe_retreat_track_is_reached)) and
  has_stat=retreating) and !HorizontalAvoidanceFailure.Action = inject;
  has_avoidance_failed_faulty = true and has_stat =
  avoidance_failed_event_sent_to_mission_control_by_hca if
(((Depth_control_operation.Action=send_limited_headroom_to_pctrl and !
Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and!
Retreat_path_array.Action=rpa_becomes_empty) and!
Retreat_track_length_monitor.Action=end_of_safe_retreat_track_is_reached) or
((Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl and!
Retreat_path_array.Action=rpa_becomes_empty) and!
Retreat_track_length_monitor.Action=end_of_safe_retreat_track_is_reached)) and
has_stat=retreating) and HorizontalAvoidanceFailure.Action = inject;
  has_stat = trying_new_track if ((Action=start_trying_new_track and
has_stat=initialize_components_for_trying_new_track_state) or
(((((Action=maintain_trying_new_track and!
Depth_control_operation.Action=send_limited_headroom_to_pctrl) and!
Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and!
Obstacle_clearance_meter.Action=clearance_distance_has_been_covered) and!
New_demand_detection.Action=new Demand_detection_occurs) and
has_stat=try new track));
  has_stat = obstacle_detected_while_retreating if
((Action=hca_sends_avoidance_failed_event_to_misctrl and
(has_avoidance_failed_faulty=true and has_stat=avoidance_failed_event_sent_to_mission_control_by_hca)) or (has_avoidance_failed_faulty=false and
has_stat=avoidance_failed_event_sent_to_mission_control_by_hca));
end Evolution
end Agent
Agent Mission_timer
Vars:
  mt_stat: { duration_too_long, expired, idling, initialized, operation_enabled
};
timeout: boolean;
end Vars
RedStates:
end RedStates
Actions = { mission_has_continued_for_too_long, no_mt_act, schedule_mt_to_expire
};
Protocol:
  (((mt_stat=idling or (mt_stat=operation_enabled and timeout=false)) or
mt_stat=expired) or (mt_stat=operation_enabled and timeout=true)): { no_mt_act
};
  mt_stat=initialized: { schedule_mt_to_expire
};
  mt_stat=duration_too_long: { mission_has_continued_for_too_long
};
end Protocol
Evolution:
  mt_stat = idling if ((Action=no_mt_act and
Mission_timer_instruction_parser.Action=set_mission_timer_action and mt_stat=idling);
  mt_stat = initialized if (Mission_timer_instruction_parser.Action=set_mission
_timer and mt_stat=idling);
  mt_stat = operation_enabled and timeout = false if
((Action=schedule_mt_to_expire and mt_stat=initialized) or
(mt_stat=operation_enabled and timeout=false)) and !MissionTooLong.Action =
inject;
  mt_stat = operation_enabled and timeout = true if
((Action=schedule_mt_to_expire and mt_stat=initialized) or
(mt_stat=operation_enabled and timeout=false)) and MissionTooLong.Action =
inject;
  mt_stat = duration_too_long if (mt_stat=operation_enabled and timeout=true);
  mt_stat = expired if ((Action=mission_has_continued_for_too_long and
mt_stat=duration_too_long) or mt_stat=expired);
end Evolution
end Agent
Agent Emergency_abort_system
Vars:
  state: { mission_aborted, normal_operation
};
end Vars
RedStates:
end RedStates
Actions = { abort_mission_and_rise_to_surface, no_eas_act
};
Protocol:
  state=normal operation: { no_eas_act
};
  state=mission_aborted: { abort_mission_and_rise_to_surface
};
end Protocol
Evolution:
    state = normal_operation if (((Action=no_eas_act and ! Mission_timer.Action=mission_has_continued_for_too_long) and ! Mission_control_internal_events.Action=vehicle_is_too_deep) and state=normal_operation);
    state = mission_aborted if (((Mission_timer.Action=mission_has_continued_for_too_long or (Mission_control_internal_events.Action=vehicle_is_too_deep and ! Mission_timer.Action=mission_has_continued_for_too_long)) and state=normal_operation) or state=mission_aborted);
end Evolution
end Agent

Agent VehicleAppearsFaulty
Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rsst }
end Vars
RedStates:
end RedStates
Actions = { inject, notinject, start }
Protocol:
    status=fault_i: { inject };  
    status=fault_ni: { inject };  
    status=nofault: { noinject };  
    status=stop_i: { noinject };  
    status=stop_ni: { noinject };  
    status=w_astt: { noinject };  
    status=w_rsst: { noinject, start };  
end Protocol
Evolution:
    status = fault_ni if (status=w_rsst and Action=start);  
    status = w_rsst if (status=w_astt and Mission_script_execution.Action=mission_script_started);  
    status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Human.Action=human_sends_homing_signal);  
    status = stop_i if (((status=fault_i or status=fault_ni) and Human.Action=human_sends_homing_signal) and Action=inject);  
    status = stop_ni if status=stop_i;  
end Evolution
end Agent

Agent VehicleTooDeep
Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rsst }
end Vars
RedStates:
end RedStates
Actions = { inject, notinject, start }
Protocol:
    status=fault_i: { noinject };  
    status=fault_ni: { inject };  
    status=nofault: { noinject };  
    status=stop_i: { noinject };  
    status=stop_ni: { noinject };  
    status=w_astt: { noinject };  
    status=w_rsst: { noinject, start };  
end Protocol
Evolution:
    status = w_rsst if (status=w_rsst and Action=start);  
    status = fault_ni if (status=w_astt and Mission_control_internal_events.Action=send_vehicle_dived_event_to_msctrl);  
    status = fault_i if ((status=fault_i or status=fault_ni) and ! Mission_control_internal_events.Action=vehicle_is_too_deep);  
    status = stop_i if ((status=fault_i or status=fault_ni) and Mission_control_internal_events.Action=vehicle_is_too_deep);  
    status = stop_ni if status=stop_i;  
end Evolution
end Agent

Agent HomingLostBeforeAcquired

Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_rstt }; 
end Vars 
RedStates: 
end RedStates
Actions = { inject, notinject, start };
Protocol: 
    status=fault_i: { inject }; 
    status=fault_ni: { inject }; 
    status=nofault: { notinject }; 
    status=stop_i: { notinject }; 
    status=stop_ni: { notinject }; 
    status=w_rstt: { notinject, start }; 
end Protocol 
Evolution: 
    status = fault_ni if (status=w_rstt and Action=start); 
    status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject)
            and ! Homing_system.Action=send_homing_lost_event_to_msctrl); 
    status = stop_i if (((status=fault_i or status=fault_ni) and 
            Homing_system.Action=send_homing_lost_event_to_msctrl) and Action=inject); 
    status = stop_ni if status=stop_i; 
end Evolution
end Agent

Agent HomingLostAfterAcquired

Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt }; 
end Vars 
RedStates: 
end RedStates
Actions = { inject, notinject }; 
Protocol: 
    status=fault_i: { inject }; 
    status=fault_ni: { inject }; 
    status=nofault: { notinject }; 
    status=stop_i: { notinject }; 
    status=stop_ni: { notinject }; 
    status=w_astt: { notinject }; 
end Protocol 
Evolution: 
    status = fault_ni if (status=w_astt and 
            Homing_system.Action=send_homing_acquired_event_to_msctrl); 
    status = stop_i if status=fault_ni; 
    status = stop_ni if status=stop_i; 
end Evolution
end Agent

Agent PowerLow

Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt }; 
end Vars 
RedStates: 
end RedStates
Actions = { inject, notinject, start };
Protocol: 
    status=fault_i: { inject }; 
    status=fault_ni: { inject }; 
    status=nofault: { notinject }; 
    status=stop_i: { notinject }; 
    status=stop_ni: { notinject }; 
    status=w_astt: { notinject }; 
    status=w_rstt: { notinject, start }; 
end Protocol 
Evolution:
status = fault_ni if (status=w_rstt and Action=start);
status = w_rstt if (status=w_astt and Mission_script_execution.Action=mission_script_started);
status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Mission_timer.Action=mission_has_continued_for_too_long);
status = stop_i if ((status=fault_i or status=fault_ni) and Action=inject);
status = stop_ni if status=stop_i;
end Evolution
end Agent

Agent MissionTooLong
Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt };
end Vars
RedStates:
Actions = { inject, notinject, start };
Protocol:
    status=fault_i: { inject };
    status=fault_ni: { inject };
    status=nofault: { notinject };
    status=stop_i: { notinject };
    status=stop_ni: { notinject };
    status=w_astt: { notinject, inject };
    status=w_rstt: { notinject, start };
end Protocol
Evolution:
    status = fault_ni if (status=w_rstt and Action=start);
    status = w_rstt if (status=w_astt and Mission_script_execution.Action=mission_script_started);
    status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Mission_timer.Action=mission_has_continued_for_too_long);
    status = stop_i if ((status=fault_i or status=fault_ni) and Mission_timer.Action=mission_has_continued_for_too_long) and Action=inject);
    status = stop_ni if status=stop_i;
end Evolution
end Agent

Agent Avoidance_timer_failed
Vars:
    status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt };
end Vars
RedStates:
Actions = { inject, notinject, start };
Protocol:
    status=fault_i: { inject };
    status=fault_ni: { inject };
    status=nofault: { notinject };
    status=stop_i: { notinject };
    status=stop_ni: { notinject };
    status=w_astt: { notinject };
    status=w_rstt: { notinject, start };
end Protocol
Evolution:
    status = fault_ni if (status=w_rstt and Action=start);
    status = w_rstt if (status=w_astt and Mission_script_execution.Action=mission_script_started);
    status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl);
    status = stop_i if ((status=fault_i or status=fault_ni) and Avoidance_timer.Action=at_sends_avoidance_failed_event_to_msctrl) and Action=inject);
    status = stop_ni if status=stop_i;
end Evolution
end Agent

Agent RetryTrackAvoidanceFail
Vars:
status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt }; end Vars
RedStates:
Actions = { inject, notinject, start }; Protocol:
status=fault_i: { inject }
status=fault_ni: { inject }
status=nofault: { notinject }
status=stop_i: { notinject }
status=stop_ni: { notinject }
status=w_astt: { notinject }
status=w_rstt: { notinject, start }
end Protocol
Evolution:
status = fault_ni if (status=w_astt and Horizontal_avoidance_strategy.Action=start_retreating);
status = w_astt if (status=w_rstt and Action=start);
status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Retry_track_counter.Action=rtc_sends_avoidance_failed_event_to_msctrl);
status = stop_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and Action=inject);
status = stop_ni if status=stop_i;
end Evolution
end Agent
Agent RetreatPathArrayAvoidanceFailure Vars:
status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt }; end Vars
RedStates:
Actions = { inject, notinject, start }; Protocol:
status=fault_i: { inject }
status=fault_ni: { inject }
status=nofault: { notinject }
status=stop_i: { notinject }
status=stop_ni: { notinject }
status=w_astt: { notinject }
status=w_rstt: { notinject, start }
end Protocol
Evolution:
status = fault_ni if (status=w_astt and Horizontal_avoidance_strategy.Action=start_retreating);
status = w_astt if status=w_rstt;
status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl);
status = stop_i if (((status=fault_i or status=fault_ni) and Action=inject) and ! Retreat_path_array.Action=rpa_sends_avoidance_failed_event_to_msctrl) and Action=inject);
status = stop_ni if status=stop_i;
end Evolution
end Agent
Agent HorizontalAvoidanceFailure Vars:
status: { fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt }; end Vars
RedStates:
Actions = { inject, notinject, start }; Protocol:
status=fault_i: { inject }
status=fault_ni: { inject }
status=nofault: { notinject }
status=stop_i: { notinject }

status=stop_ni: { notinject }
status=w_astt: { notinject }
status=w_rstt: { notinject, start }
end Protocol

Evolution:
status = fault_ni if (status=w_astt and
Horizontal_avoidance_strategy.Action=start_retreating);
status = w_astt if status=w_rstt;
status = fault_i if (((status=fault_i or status=fault_ni) and Action=inject)
and ! Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl);
status = stop_i if (((status=fault_i or status=fault_ni) and
Horizontal_avoidance_strategy.Action=hca_sends_avoidance_failed_event_to_msctrl)
and Action=inject);
status = stop_ni if status=stop_i;
end Evolution
end Agent

Agent EchosounderDetects

Vars:
status: {fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt};
end Vars

RedStates:
end RedStates

Actions = {inject, notinject, start};
end Protocol

Evolution:
status = w_astt if status = w_rstt and EchosounderDetects.Action = start;
status = fault_ni if status = w_astt and (Horizontal_avoidance_strategy.Action=avoidance_process_is_idling or Horizontal_avoidance_strategy.Action=maintain_retreating);
status = fault_i if (status = fault_i or status = fault_ni) and EchosounderDetects.Action = inject and !Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl;
status = stop_i if ((status = fault_i or status = fault_ni) and Forward_looking_echosounder.Action=send_obstacle_ahead_to_pctrl) and EchosounderDetects.Action = inject;
status = stop_ni if status = stop_i;
end Evolution
end Agent

Agent Min_Alt_Range

Vars:
status: {fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt};
end Vars

RedStates:
end RedStates

Actions = {inject, notinject, start};
end Protocol
status = stop_ni : {notinject};
status = w_astt : {notinject};
status = w_rstt : {notinject, start};
end Protocol

Evolution:

status = w_astt if status = w_rstt and Min_Alt.Range.Action = start;
status = fault_ni if status = w_astt and (Horizontal_avoidance_strategy.Action=avoidance_process_is_idling or Horizontal_avoidance_strategy.Action=maintain_retreating);
status = fault_i if (status = fault_i or status = fault_ni) and Min_Alt.Range.Action = inject and !Depth_control_operation.Action=send_limited_headroom_to_pctrl;
status = stop_i if ((status = fault_i or status = fault_ni) and Depth_control_operation.Action=send_limited_headroom_to_pctrl) and Min_Alt.Range.Action = inject;
status = stop_ni if status = stop_i;
end Evolution

end Agent

Agent Min_Ice_Range

Vars:
status: {fault_i, fault_ni, nofault, stop_i, stop_ni, w_astt, w_rstt};
end Vars

RedStates:
end RedStates

Actions = {inject, notinject, start};

Protocol:
status = fault_i : {inject};
status = fault_ni : {inject};
status = nofault : {notinject};
status = stop_i : {notinject};
status = stop_ni : {notinject};
status = w_astt : {notinject};
status = w_rstt : {notinject, start};
end Protocol

Evolution:

status = w_astt if status = w_rstt and Min_Ice_Range.Action = start;
status = fault_ni if status = w_astt and (Horizontal_avoidance_strategy.Action=avoidance_process_is_idling or Horizontal_avoidance_strategy.Action=maintain_retreating);
status = fault_i if (status = fault_i or status = fault_ni) and Min_Ice_Range.Action = inject and !Depth_control_operation.Action=send_limited_headroom_to_pctrl;
status = stop_i if ((status = fault_i or status = fault_ni) and Depth_control_operation.Action=send_limited_headroom_to_pctrl) and Min_Ice_Range.Action = inject;
status = stop_ni if status = stop_i;
end Evolution

end Agent

Evaluation

Avoidance_timer_failed_faulty if !Avoidance_timer_failed.status=nofault;
Avoidance_timer_failed_injected if (Avoidance_timer_failed.status=fault_i or Avoidance_timer_failed.status=stop_i);
Avoidance_timer_failed_injecting if ((Avoidance_timer_failed.status=nofault or Avoidance_timer_failed.status=stop_ni) or Avoidance_timer_failed.status=w_astt) or Avoidance_timer_failed.status=w_rstt);
Avoidance_timer_failed_stopped if Avoidance_timer_failed.status=stop_ni;
HomingLostAfterAcquired_faulty if !HomingLostAfterAcquired.status=nofault;
HomingLostAfterAcquired_injected if (HomingLostAfterAcquired.status=fault_i or HomingLostAfterAcquired.status=stop_i) or HomingLostAfterAcquired.status=stop_ni);
HomingLostAfterAcquired injecting if !
((HomingLostAfterAcquired.status=nofault or
HomingLostAfterAcquired.status=stop_ni) or
HomingLostAfterAcquired.status=w_astt);
HomingLostAfterAcquired stopped if HomingLostAfterAcquired.status=stop_ni;
HomingLostBeforeAcquired faulty if ! HomingLostBeforeAcquired.status=nofault;
HomingLostBeforeAcquired injected if (HomingLostBeforeAcquired.status=fault_i
or HomingLostBeforeAcquired.status=stop_i);
HomingLostBeforeAcquired injecting if !
((HomingLostBeforeAcquired.status=nofault or
HomingLostBeforeAcquired.status=stop_ni) or
HomingLostBeforeAcquired.status=w_rstt);
HomingLostBeforeAcquired stopped if HomingLostBeforeAcquired.status=stop_ni;
HomingLostBeforeAcquired injecting if ! (((HomingLostBeforeAcquired.status=nofault or
HomingLostBeforeAcquired.status=stop_ni) or
HomingLostBeforeAcquired.status=w_astt) or
HomingLostBeforeAcquired.status=w_rstt);
HorizontalAvoidanceFailure faulty if !
HorizontalAvoidanceFailure_injected if
(HorizontalAvoidanceFailure.status=fault_i or
HorizontalAvoidanceFailure.status=stop_i);
HorizontalAvoidanceFailure injecting if !
(((HorizontalAvoidanceFailure.status=nofault or
HorizontalAvoidanceFailure.status=stop_i) or
HorizontalAvoidanceFailure.status=w_rstt);
HorizontalAvoidanceFailure stopped if HorizontalAvoidanceFailure.status=stop_ni;
MissionTooLong faulty if ! MissionTooLong.status=nofault;
MissionTooLong injected if (MissionTooLong.status=fault_i or
MissionTooLong.status=stop_i);
MissionTooLong injecting if ! (((MissionTooLong.status=nofault or
MissionTooLong.status=stop_i) or
MissionTooLong.status=w_rstt);
MissionTooLong stopped if MissionTooLong.status=stop_ni;
NoFaults if (((((((MissionTooLong.status=nofault and
PowerLow.status=nofault) and HomingLostAfterAcquired.status=nofault) and
HomingLostBeforeAcquired.status=nofault) and VehicleTooDeep.status=nofault) and
RetryTrackAvoidanceFail.status=nofault) and RetreatPathArrayAvoidanceFailure.status=nofault)
and HorizontalAvoidanceFailure.status=nofault) and
Avoidance_timer_failed.status=nofault);
PowerLow faulty if ! PowerLow.status=nofault;
PowerLow injected if (PowerLow.status=fault_i or PowerLow.status=stop_i);
PowerLow injecting if ! (((PowerLow.status=nofault or
PowerLow.status=stop_ni) or PowerLow.status=w_astt) or PowerLow.status=w_rstt);
PowerLow stopped if PowerLow.status=stop_ni;
RetreatPathArrayAvoidanceFailure faulty if !
RetreatPathArrayAvoidanceFailure.injected if
(RetreatPathArrayAvoidanceFailure.status=fault_i or
RetreatPathArrayAvoidanceFailure.status=stop_i);
RetreatPathArrayAvoidanceFailure injecting if !
(((RetreatPathArrayAvoidanceFailure.status=nofault or
RetreatPathArrayAvoidanceFailure.status=stop_ni) or
RetreatPathArrayAvoidanceFailure.status=w_astt) or
RetreatPathArrayAvoidanceFailure.status=w_rstt);
RetreatPathArrayAvoidanceFailure stopped if
RetreatPathArrayAvoidanceFailure.status=stop_ni;
RetryTrackAvoidanceFail faulty if ! RetryTrackAvoidanceFail.status=nofault;
RetryTrackAvoidanceFail injected if (RetryTrackAvoidanceFail.status=fault_i
or RetryTrackAvoidanceFail.status=stop_i);
RetryTrackAvoidanceFail injecting if !
(((RetryTrackAvoidanceFail.status=nofault or
RetryTrackAvoidanceFail.status=stop_ni) or RetryTrackAvoidanceFail.status=w_astt) or
RetryTrackAvoidanceFail.status=w_rstt);
RetryTrackAvoidanceFail stopped if RetryTrackAvoidanceFail.status=stop_ni;
VehicleAppearsFaulty faulty if ! VehicleAppearsFaulty.status=nofault;
VehicleAppearsFaulty injected if (VehicleAppearsFaulty.status=fault_i or
VehicleAppearsFaulty.status=stop_i);
VehicleAppearsFaulty injecting if ! (((VehicleAppearsFaulty.status=nofault or
VehicleAppearsFaulty.status=stop_ni) or VehicleAppearsFaulty.status=w_astt) or
VehicleAppearsFaulty.status=w_rstt);
VehicleAppearsFaulty.status = w_rstt);
    VehicleAppearsFaulty_stopped if VehicleAppearsFaulty.status = stop_ni;
    VehicleTooDeep_faulty if !VehicleTooDeep.status = nofault;
    VehicleTooDeep_injected if (VehicleTooDeep.status = fault_i or
    VehicleTooDeep.status = stop_i);
    VehicleTooDeep_injecting if !(VehicleTooDeep.status = nofault or
    VehicleTooDeep.status = stop_ni) or VehicleTooDeep.status = w_astt) or
    VehicleTooDeep_stopped if VehicleTooDeep.status = stop_ni;
    got_position if Position_control_operation.state = got_position_event_sent_to_msctrl;
    gps_fix_acquired if Gps_navigation.gps_stat = fix_acquired;
    homing_underway_event_proc If Mission_script_execution.mse_stat = homing_underway_event_proc;
    linetimerenabled if Line_timer.state = lt_operation_enabled;
    linetimersent if Line_timer.state = lt_event_sent_to_msctrl;
    mission_aborted if Emergency_abort_system.state = mission_aborted;
    mission_script_first_me if Mission_script_execution.mse_stat = first_me_ongoing;
    mission_script_fourth_me if Mission_script_execution.mse_stat = fourth_me_ongoing;
    mission_script_second_me if Mission_script_execution.mse_stat = second_me_ongoing;
    mission_script_start if Mission_script_execution.mse_stat = ms_start;
    mission_script_third_me if Mission_script_execution.mse_stat = third_me_ongoing;
    mte1_ongoing if Mission_script_execution.mse_stat = mte1_ongoing;
    mte2_event_proc if Mission_script_execution.mse_stat = mte2_event_proc;
    send_event if (Mission_control_internal_events.depth_ref = surface_depth and
    Mission_control_internal_events.depth_rel = greater_than) and
    EchosounderDetects_faulty if !EchosounderDetects.status = nofault;
    EchosounderDetects_injected if EchosounderDetects.status = fault_i or
    EchosounderDetects_injecting if !(EchosounderDetects.status = nofault or
    EchosounderDetects.status = stop_ni or EchosounderDetects.status = w_astt or
    EchosounderDetects.status = w_rstt);
    EchosounderDetects_stopped if EchosounderDetects.status = stop_ni;
    Min_Alt_Range_faulty if !Min_Alt_Range.status = nofault;
    Min_Alt_Range_injected if Min_Alt_Range.status = fault_i or
    Min_Alt_Range_injecting if !(Min_Alt_Range.status = nofault or
    Min_Alt_Range.status = stop_ni) or Min_Alt_Range.status = w_astt or
    Min_Alt_Range_stopped if Min_Alt_Range.status = stop_ni;
    Min_Ice_Range_faulty if !Min_Ice_Range.status = nofault;
    Min_Ice_Range_injected if Min_Ice_Range.status = fault_i or
    Min_Ice_Range_injecting if !(Min_Ice_Range.status = nofault or
    Min_Ice_Range.status = stop_ni) or Min_Ice_Range.status = w_astt or
    Min_Ice_Range_stopped if Min_Ice_Range.status = stop_ni;
    VASsafe if Vertical_avoidance_strategy.max_depth = safe;
    HASretreating if Horizontal_avoidance_strategy.has_stat = retreating;
    HASwaiting if Horizontal_avoidance_strategy.has_stat = waiting;
    ATretreating if Avoidance_timer.at_stat = operation_enabled;

Echosounderfault if Forward_looking_echosounder.echosounder_stat = not_detected and
Forward_looking_echosounder.fwddist_ref = max_range
and Forward_looking_echosounder.fwddist_rel = less_than;
DepthControlFault if Depth_control_operation.limhdr_stat=not_detected and Depth_control_operation.ru_rel=less_than and Depth_control_operation.ru_ref=min_ice_clearance and Depth_control_operation.rd_rel=less_than and Depth_control_operation.rd_ref=min_alt;
NewPositionControlDemand if Position_control_instruction_parser.state = track_follow_mode;
    HAS_new_track if Horizontal_avoidance_strategy.has_stat = trying_new_track;
NewDemandDetectionOccurs if New_demand_detection.state = demand;
end Evaluation
InitStates
   (((((((((((((((((((((((((((((((((((((((((((((Gps_navigation.gps_stat = fix_not_acquired and Obstacle_clearance_meter.state=has_waiting and Avoidance_timer.timeout=false and Retry_track_counter.allowed_no_rtc_reached=false and Retreat_path_array.rpa_empty=false and Safe_position_meter.state=idling and Environment.vehicle_appears_faulty=false and Depth_control_operation.limhdr_stat=idle) and Depth_control_operation.rd_ref=min_alt) and Depth_control_operation.rd_rel=greater_than) and Depth_control_operation.ru_ref=min_ice_clearance) and Depth_control_operation.ru_rel=greater_than) and Vertical_avoidance_strategy.max_depth=normal) and Vertical_avoidance_strategy.min_alt=normal) and Vertical_avoidance_strategy.min_ice_clearance=normal) and Vertical_avoidance_strategy.min_depth=normal) and Vertical_avoidance_strategy.min_ice_clearance=normal) and Forward_looking_echosounder.echosounder_stat=idle) and Forward_looking_echosounder.fwddist_ref=max_range) and Forward_looking_echosounder.fwddist_rel=greater_than) and Mission_control_internal_events.depth_ref=surface_depth) and Mission_control_internal_events.depth_rel=equal) and Mission_control_internal_events.depth_ref=vehicle_on_surface) and Homing_system.beacon_signal=off) and Homing_system.homing_status=homing_system_off) and Mission_timer_instruction_parser.state=idle) and Power_node.pwr_ref=safe_level) and Power_node.pwr_rel=greater_than) and Power_node.pwr_stat=idle) and Mission_script_execution.mse_stat=ms_default_state) and Motor_control_instruction_parser.state=idle) and Position_control_instruction_parser.state=idle) and Line_timerInstruction_parser.state=idle) and Line_timer.state=lt_idling) and New_demand_detection.state=nodemand) and Avoidance_timer.at_stat=idling) and Position_control_operation.state=at_mission_start) and Horizontal_avoidance_strategy.has_stat=initialize_components_for_waiting_state and Horizontal_avoidance_strategy.has_avoidance_failed_faulty=false) and Mission_timer.mt_stat=idling) and Emergency_abort_system.state=normal_operation) and Human.human_stat=observing_auv) and (VehicleAppearsFaulty.status=nofault or VehicleAppearsFaulty.status=wastt) and (VehicleTooDeep.status=nofault or VehicleTooDeep.status=wrstt) and (HomingLostBeforeAcquired.status=nofault or HomingLostBeforeAcquired.status=wastt) and (HomingLostAfterAcquired.status=nofault or HomingLostAfterAcquired.status=wastt) and (PowerLow.status=nofault or PowerLow.status=wastt) and (MissionTooLong.status=nofault or MissionTooLong.status=wastt) and (Avoidance_timer_failed.status=nofault or Avoidance_timer_failed.status=wastt)) and (RetryTrackAvoidanceFail.status=nofault or RetryTrackAvoidanceFail.status=wrstt) and (RetreatPathArrayAvoidanceFailure.status=nofault or RetreatPathArrayAvoidanceFailure.status=wrstt) and (HorizontalAvoidanceFailure.status=nofault or HorizontalAvoidanceFailure.status=wrstt) and (EchosounderDetects.status = nofault or EchosounderDetects.status = wrstt) and (Min_Alt_Range.status = nofault or Min_Alt_Range.status = wrstt) and
(Min_Ice_Range.status = nofault or Min_Ice_Range.status = w_rstt)
end InitStates

Groups
  group1 = {Avoidance_timer, Retry_track_counter, Safe_position_meter, Retreat_track_length_monitor};
end Groups

Fairness
end Fairness

Formulae

-- verify the script behaves normally with faults
AF(mission_script_start);
AF(mission_script_first_me);
AF(mission_script_second_me);
AF(mission_script_third_me);
AF(mission_script_fourth_me);
AF(AG(mission_script_fourth_me));

-- verify the script behaves normally without faults
AF(NoFaults -> mission_script_start);
AF(NoFaults -> mission_script_first_me);
AF(NoFaults -> mission_script_second_me);
AF(NoFaults -> mission_script_third_me);
AF(NoFaults -> mission_script_fourth_me);
AF(NoFaults -> AG(mission_script_fourth_me));

-- verify whether we always enter particular script termination states
!AF(mte1_ongoing);
!AF(mte2_event_proc);
!AF(homing_underway_event_proc);

-- verify the faults get injected
AG(!VehicleAppearsFaulty_injected);
AG(!MissionTooLong_injected);
AG(!VehicleTooDeep_injected);
AG(!PowerLow_injected);
AG(!HomingLostAfterAcquired_injected);
AG(!HomingLostBeforeAcquired_injected);
AG(!Avoidance_timer_failed_injected);
AG(!RetryTrackAvoidanceFail_injected);
AG(!HorizontalAvoidanceFailure_stopped); -- we use stopped here as this is the case where the fault has the desired impact
AG(!RetreatPathArrayAvoidanceFailure_stopped);

-- verify the power low and avoidance failures end up in mte1
AG(PowerLow_injected->AF(mte1_ongoing));
AG(Avoidance_timer_failed_injected->AF(mte1_ongoing));
AG(RetryTrackAvoidanceFail_injected->AF(mte1_ongoing));
AG(RetreatPathArrayAvoidanceFailure_stopped->AF(mte1_ongoing));
AG(HorizontalAvoidanceFailure_stopped->AF(mte1_ongoing));

-- now verify they end up in mte1 if there is no homing
AG(!VehicleAppearsFaulty_faulty and PowerLow_injected->AF(mte1_ongoing));
AG(!VehicleAppearsFaulty_faulty and Avoidance_timer_failed_injected->AF(mte1_ongoing));
AG(!VehicleAppearsFaulty_faulty and RetryTrackAvoidanceFail_injected->AF(mte1_ongoing));
ongoing));
   AG(!VehicleAppearsFaulty_faulty and RetreatPathArrayAvoidanceFailure_stopped->AF(mte1_ongoing));
   AG(!VehicleAppearsFaulty_faulty and HorizontalAvoidanceFailure_stopped->AF(mte1_ongoing));

   -- verify knowledge of the power low and avoidance failures are correct

   AG (mte1_ongoing -> ( K(Mission_script_execution,
   ((Avoidance_timer_failed_stopped or Avoidance_timer_failed_injecting) or
   (PowerLow_stopped or PowerLow_injecting) or
   (RetreatPathArrayAvoidanceFailure_stopped or
   RetreatPathArrayAvoidanceFailure_injecting) or (RetryTrackAvoidanceFail_stopped
   or RetryTrackAvoidanceFail_injecting) or (HorizontalAvoidanceFailure_stopped
   or HorizontalAvoidanceFailure_injecting) ))
   and (! K(Mission_script_execution, (Avoidance_timer_failed_stopped or
   Avoidance_timer_failed_injecting)))) and (! K(Mission_script_execution,
   (RetreatPathArrayAvoidanceFailure_stopped or
   RetreatPathArrayAvoidanceFailure_injecting))) and (! K(Mission_script_execution,
   (RetryTrackAvoidanceFail_stopped or RetryTrackAvoidanceFail_injecting)))
   and (! K(Mission_script_execution, (PowerLow_stopped or PowerLow_injecting))) and (!
   K(Mission_script_execution, (RetreatPathArrayAvoidanceFailure_stopped or
   RetreatPathArrayAvoidanceFailure_injecting)));

   -- verify homing signal behaviour

   -- vehicle appears faulty always enters a homing underway state
   AG(VehicleAppearsFaulty_injected -> AF(homing_underway_event_proc));

   -- vehicle appears faulty always enters a homing underway state if there is no
   beacon loss before it is acquired
   AG(VehicleAppearsFaulty_injected -> AF(homing_underway_event_proc));
   AG(VehicleAppearsFaulty_injected and !HomingLostBeforeAcquired_faulty ->
   AF(homing_underway_event_proc));

   -- if the homing is lost after it is acquired always enter a homing lost state
   AG( HomingLostAfterAcquired_injected -> AF(mte2_event_proc));

   -- always recover back to a homing underway state
   AG( HomingLostAfterAcquired_injected and !HomingLostBeforeAcquired_faulty ->
   AF(homing_underway_event_proc));

   -- verify emergency abort

   -- always enter a mission abort state for vehicle too deep or mission too long
   AG(!VehicleTooDeep_stopped);
   AG(!MissionTooLong_stopped);
   AG(VehicleTooDeep_stopped -> AF(mission_aborted ));
   AG(MissionTooLong_stopped -> AF(mission_aborted ));

   -- verify emergency abort knowledge
   (AG (mission_aborted -> ((K(Emergency_abort_system, ((VehicleTooDeep_stopped or
   VehicleTooDeep_injecting) or (MissionTooLong_stopped or
   MissionTooLong_injecting))) and (! K(Emergency_abort_system,
   (VehicleTooDeep_stopped or VehicleTooDeep_injecting))) and (!
   K(Emergency_abort_system, (MissionTooLong_stopped or
   MissionTooLong_injecting))))));
-------------- next round of faults distribution of knowledge

AG(! Min_Ice_Range_injected);
AG(! Min_Alt_Range_injected);
AG(! EchosounderDetects_injected);

AG(EchosounderDetects_injected -> AF(K(Forward_looking_echosounder,(EchosounderDetects_injecting or EchosounderDetects_stopped))));

AG((Min_Ice_Range_injected and Min_Alt_Range_injected) -> AF(K(Depth_control_operation,(Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped) ))));

AG(VASsafe -> K(Vertical_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)) )));
AG(VASsafe -> !K(Vertical_avoidance_strategy, (EchosounderDetects_injecting or EchosounderDetects_stopped)));
AG(VASsafe -> !K(Vertical_avoidance_strategy, ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped) ) ));

AG(EchosounderDetects_injected -> AF(HASretreating));
AG((Min_Ice_Range_injected and Min_Alt_Range_injected) -> AF(HASretreating));
AG((Min_Ice_Range_injected) -> AF(HASretreating));
AG((Min_Alt_Range_injected) -> AF(HASretreating));

AG(HASretreating -> K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)) )));
AG(HASretreating -> !K(Horizontal_avoidance_strategy, (EchosounderDetects_injecting or EchosounderDetects_stopped)));
AG(HASretreating -> !K(Horizontal_avoidance_strategy, ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped) ) ));

AG(ATretreating -> K(Avoidance_timer, ((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)) )));
AG(ATretreating -> !K(Avoidance_timer, (EchosounderDetects_injecting or EchosounderDetects_stopped)));
AG(ATretreating -> !K(Avoidance_timer, ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)) ) });

---

-- check track state and fault injection
AG(!HASnewtrack);
AG(!HASnewtrack and VehicleAppearsFaulty_faulty and Avoidance_timer_failed_injected);
AG((HASnewtrack and VehicleAppearsFaulty_faulty) and RetryTrackAvoidanceFail_injected);

-- These prove that the stuck situation occurs
AG(HASnewtrack -> AF(HASwaiting ));
AG((HASnewtrack and !VehicleAppearsFaulty_faulty) -> AF(HASwaiting ));

-- These prove that the avoidance failure and retry track lead back to waiting state
AG((HASnewtrack and Avoidance_timer_failed_injected and !VehicleAppearsFaulty_faulty) -> AF(HASwaiting ));
AG((HASnewtrack and RetryTrackAvoidanceFail_injected and !VehicleAppearsFaulty_faulty) -> AF(HASwaiting ));
!E((HASnewtrack and !VehicleAppearsFaulty_faulty and !Avoidance_timer_failed_injected and !RetryTrackAvoidanceFail_injected) U (RetryTrackAvoidanceFail_injected or Avoidance_timer_failed_injected) and !AF(mte1_ongoing and AF(NewPositionControlDemand and AF(NewDemandDetectionOccurs and AF(HASwaiting)))) )

---Prove echosounder distributes to horizontal and vertical avoidance
!E( !K(Forward_looking_echosounder,(EchosounderDetects_injecting or EchosounderDetects_stopped)) U K(Forward_looking_echosounder,(EchosounderDetects_injecting or EchosounderDetects_stopped)) and !AF(K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))) )));
!E( !K(Forward_looking_echosounder,(EchosounderDetects_injecting or EchosounderDetects_stopped)) U K(Forward_looking_echosounder,(EchosounderDetects_injecting or EchosounderDetects_stopped)) and !AF(K(Vertical_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))) )

---Prove Depth_control_operation distributes to horizontal and vertical avoidance
!E( !K(Depth_control_operation,((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))) U K(Depth_control_operation,((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))) and !AF(K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))) )

!E( !K(Depth_control_operation,((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))) U K(Depth_control_operation,((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))) and !AF(K(Vertical_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))) )

--- individual distribution of knowledge
!E( !K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))) U K(Horizontal_avoidance_strategy,((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))) and !AF(K(Obstacle_clearance_meter,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))))

common knowledge

!E(!K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))))
U K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))))
and !AF(GK(group1,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))));

!E(!K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))))
U K(Horizontal_avoidance_strategy,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped))))
and !AF(GCK(group1,((EchosounderDetects_injecting or EchosounderDetects_stopped) or ((Min_Ice_Range_injecting or Min_Ice_Range_stopped) and (Min_Alt_Range_injecting or Min_Alt_Range_stopped)))));

end Formulae
Appendix III

Verification property results provided by the MCMAS model checker
MCMAS v1.0.0

This software comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

Please check http://www-lai.doc.ic.ac.uk/mcmas/ for the latest release.
Report bugs to <mcmas@imperial.ac.uk>

/home/jezekiel/work/AUVaug20/model19augfi.ispl has been parsed successfully.

Global syntax checking...
Done
Encoding BDD parameters...
Building partial transition relation...
Building OBDD for initial states...
Building reachable state space...
Checking formulae...
Verifying properties...

Formula number 1: (AF mission_script_start), is TRUE in the model
Formula number 2: (AF mission_script_first_me), is TRUE in the model
Formula number 3: (AF mission_script_second_me), is FALSE in the model
Formula number 4: (AF mission_script_third_me), is FALSE in the model
Formula number 5: (AF mission_script_fourth_me), is FALSE in the model
Formula number 6: (AF (AG mission_script_fourth_me)), is FALSE in the model
Formula number 7: (AF (NoFaults -> mission_script_start)), is TRUE in the model
Formula number 8: (AF (NoFaults -> mission_script_first_me)), is TRUE in the model
Formula number 9: (AF (NoFaults -> mission_script_second_me)), is TRUE in the model
Formula number 10: (AF (NoFaults -> mission_script_third_me)), is TRUE in the model
Formula number 11: (AF (NoFaults -> mission_script_fourth_me)), is TRUE in the model
Formula number 12: (AF (NoFaults -> (AG mission_script_fourth_me))), is TRUE in the model

Formula number 13: (! (AF mte1_ongoing)), is TRUE in the model

Formula number 14: (! (AF mte2_event_proc)), is TRUE in the model

Formula number 15: (! (AF homing_underway_event_proc)), is TRUE in the model

Formula number 16: (AG (! VehicleAppearsFaulty_injected)), is FALSE in the model

Formula number 17: (AG (! MissionTooLong_injected)), is FALSE in the model

Formula number 18: (AG (! VehicleTooDeep_injected)), is FALSE in the model

Formula number 19: (AG (! PowerLow_injected)), is FALSE in the model

Formula number 20: (AG (! HomingLostAfterAcquired_injected)), is FALSE in the model

Formula number 21: (AG (! HomingLostBeforeAcquired_injected)), is FALSE in the model

Formula number 22: (AG (! Avoidance_timerFailed_injected)), is FALSE in the model

Formula number 23: (AG (! RetryTrackAvoidanceFailed_injected)), is FALSE in the model

Formula number 24: (AG (! HorizontalAvoidanceFailure_stopped)), is FALSE in the model

Formula number 25: (AG (! RetreatPathArrayAvoidanceFailure_stopped)), is FALSE in the model

Formula number 26: (AG (PowerLow_injected -> (AF mte1_ongoing))), is FALSE in the model

Formula number 27: (AG (Avoidance_timerFailed_injected -> (AF mte1_ongoing))), is FALSE in the model

Formula number 28: (AG (RetryTrackAvoidanceFailed_injected -> (AF mte1_ongoing))), is FALSE in the model

Formula number 29: (AG (RetreatPathArrayAvoidanceFailure_stopped -> (AF mte1_ongoing))), is FALSE in the model

Formula number 30: (AG (HorizontalAvoidanceFailure_stopped -> (AF mte1_ongoing))), is FALSE in the model

Formula number 31: (AG (((! VehicleAppearsFaulty_faulty) && PowerLow_injected) ->
(AF mte1_ongoing)), is TRUE in the model

Formula number 32: (AG (((! VehicleAppearsFaulty_faulty) && Avoidance_timer_failed_injected) -> (AF mte1_ongoing))), is TRUE in the model

Formula number 33: (AG (((! VehicleAppearsFaulty_faulty) && RetryTrackAvoidanceFail_injected) -> (AF mte1_ongoing))), is TRUE in the model

Formula number 34: (AG (((! VehicleAppearsFaulty_faulty) && RetreatPathArrayAvoidanceFailure_stopped) -> (AF mte1_ongoing))), is TRUE in the model

Formula number 35: (AG (((! VehicleAppearsFaulty_faulty) && HorizontalAvoidanceFailure_stopped) -> (AF mte1_ongoing))), is TRUE in the model

Formula number 36: (AG (mte1_ongoing -> (((((K(Mission_script_execution, (((((Avoidance_timer_failed_stopped || Avoidance_timer_failed_injecting) || (PowerLow_stopped || PowerLow_injecting)) || (RetreatPathArrayAvoidanceFailure_stopped || RetryTrackAvoidanceFail_injecting)) || (HorizontalAvoidanceFailure_stopped || HorizontalAvoidanceFailure_injecting))))) && (! K(Mission_script_execution, (RetreatPathArrayAvoidanceFailure_stopped || RetryTrackAvoidanceFail_injecting)))) && (! K(Mission_script_execution, (RetreatPathArrayAvoidanceFailure_stopped || RetryTrackAvoidanceFail_injecting))))))) && (! K(Mission_script_execution, (RetreatPathArrayAvoidanceFailure_stopped || RetryTrackAvoidanceFail_injecting))))))), is TRUE in the model

Formula number 37: (AG (VehicleAppearsFaulty_injected -> (AF homing_underway_event_proc))), is FALSE in the model

Formula number 38: (AG (VehicleAppearsFaulty_injected -> (AF homing_underway_event_proc))), is FALSE in the model

Formula number 39: (AG ((VehicleAppearsFaulty_injected && (! HomingLostBeforeAcquired_faulty)) -> (AF homing_underway_event_proc))), is TRUE in the model

Formula number 40: (AG (HomingLostAfterAcquired_injected -> (AF mte2_event_proc))), is TRUE in the model

Formula number 41: (AG ((HomingLostAfterAcquired_injected && (! HomingLostBeforeAcquired_faulty)) -> (AF homing_underway_event_proc))), is TRUE in the model

Formula number 42: (AG (! VehicleTooDeep_stopped)), is FALSE in the model

Formula number 43: (AG (! MissionTooLong_stopped)), is FALSE in the model

Formula number 44: (AG (VehicleTooDeep_stopped -> (AF mission_aborted))), is TRUE in the model

Formula number 45: (AG (MissionTooLong_stopped -> (AF mission_aborted))), is TRUE in the model

Formula number 46: (AG (mission_aborted -> ((K(Emergency_abort_system, (AF mte1_ongoing))))), is TRUE in the model
((VehicleTooDeep_stopped || VehicleTooDeep_injecting) || (MissionTooLong_stopped
|| MissionTooLong_injecting)) && (! K(Emergency_abort_system,
(VehicleTooDeep_stopped || VehicleTooDeep_injecting))) && (!
K(Emergency_abort_system, (MissionTooLong_stopped ||
MissionTooLong_injecting))))), is TRUE in the model

Formula number 47: (AG (! Min_Ice_Range_injected)), is FALSE in the model

Formula number 48: (AG (! Min_Alt_Range_injected)), is FALSE in the model

Formula number 49: (AG (! EchosounderDetects_injected)), is FALSE in the model

Formula number 50: (AG (EchosounderDetects_injected -> (AF
K(Forward_looking_echosounder, (EchosounderDetects_injecting ||
EchosounderDetects_stopped))))), is TRUE in the model

Formula number 51: (AG ((Min_Ice_Range_injected && Min_Alt_Range_injected) -> (AF
K(Depth_control_operation, ((Min_Ice_Range_injecting || Min_Ice_Range_stopped) &&
(Min_Alt_Range_injecting || Min_Alt_Range_stopped))))), is TRUE in the model

Formula number 52: (AG (VASsafe -> K(Vertical_avoidance_strategy,
((EchosounderDetects_injecting || EchosounderDetects_stopped) ||
(Min_Ice_Range_injecting || Min_Ice_Range_stopped) &&
(Min_Alt_Range_injecting || Min_Alt_Range_stopped))))), is TRUE in the model

Formula number 53: (AG (VASsafe -> (! K(Vertical_avoidance_strategy,
(EchosounderDetects_injecting || EchosounderDetects_stopped))))), is TRUE in the model

Formula number 54: (AG (VASsafe -> (! K(Vertical_avoidance_strategy,
((Min_Ice_Range_injecting || Min_Ice_Range_stopped) &&
(Min_Alt_Range_injecting || Min_Alt_Range_stopped))))), is TRUE in the model

Formula number 55: (AG (EchosounderDetects_injected -> (AF HASretreating))), is FALSE in the model

Formula number 56: (AG ((Min_Ice_Range_injected && Min_Alt_Range_injected) -> (AF
HASretreating))), is FALSE in the model

Formula number 57: (AG (Min_Ice_Range_injected -> (AF HASretreating))), is FALSE in the model

Formula number 58: (AG (Min_Alt_Range_injected -> (AF HASretreating))), is FALSE in the model

Formula number 59: (AG (HASretreating -> K(Horizontal_avoidance_strategy,
((EchosounderDetects_injecting || EchosounderDetects_stopped) ||
(Min_Ice_Range_injecting || Min_Ice_Range_stopped) &&
(Min_Alt_Range_injecting || Min_Alt_Range_stopped))))), is TRUE in the model

Formula number 60: (AG (HASretreating -> (! K(Horizontal_avoidance_strategy,
(EchosounderDetects_injecting || EchosounderDetects_stopped))))), is TRUE in the model

Formula number 61: (AG (HASretreating -> (! K(Horizontal_avoidance_strategy,
((Min_Ice_Range_injecting || Min_Ice_Range_stopped) &&
(Min_Alt_Range_injecting || Min_Alt_Range_stopped))))), is TRUE in the model
Formula number 62: (AG (ATretreating -> K(Avoidance_timer, (EchosounderDetects_injecting || EchosounderDetects_stopped) || ((Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))), is TRUE in the model

Formula number 63: (AG (ATretreating -> (! K(Avoidance_timer, (EchosounderDetects_injecting || EchosounderDetects_stopped)))), is TRUE in the model

Formula number 64: (AG (ATretreating -> (! K(Avoidance_timer, (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))), is TRUE in the model

Formula number 65: (AG (! HASnewtrack)), is FALSE in the model

Formula number 66: (AG ((! (HASnewtrack && VehicleAppearsFaulty_faulty)) && Avoidance_timer_failed_injected)), is FALSE in the model

Formula number 67: (AG ((! (HASnewtrack && VehicleAppearsFaulty_faulty)) && RetryTrackAvoidanceFail_injected)), is FALSE in the model

Formula number 68: (AG (HASnewtrack -> (AF HASwaiting))), is FALSE in the model

Formula number 69: (AG ((HASnewtrack && (! VehicleAppearsFaulty_faulty)) -> (AF HASwaiting))), is FALSE in the model

Formula number 70: (AG (((HASnewtrack && Avoidance_timer_failed_injected) && (! VehicleAppearsFaulty_faulty)) -> (AF HASwaiting))), is TRUE in the model

Formula number 71: (AG (((HASnewtrack && RetryTrackAvoidanceFail_injected) && (! VehicleAppearsFaulty_faulty)) -> (AF HASwaiting))), is TRUE in the model

Formula number 72: (! E((((HASnewtrack && (! VehicleAppearsFaulty_faulty)) && (! Avoidance_timer_failed_injected)) && (! RetryTrackAvoidanceFail_injected)) U ((RetryTrackAvoidanceFail_injected || Avoidance_timer_failed_injected) && (! (AF mte1_ongoing && (AF (NewPositionControlDemand && (AF (NewDemandDetectionOccurs && (AF HASwaiting)))))))))))), is TRUE in the model

Formula number 73: (! E(! K(Forward_looking_echosounder, (EchosounderDetects_injecting || EchosounderDetects_stopped)) U (K(Forward_looking_echosounder, (EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Horizontal_avoidance_strategy, (EchosounderDetects_injecting || EchosounderDetects_stopped) || ((Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))))), is TRUE in the model

Formula number 74: (! E(! K(Forward_looking_echosounder, (EchosounderDetects_injecting || EchosounderDetects_stopped)) U (K(Forward_looking_echosounder, (EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Vertical_avoidance_strategy, (EchosounderDetects_injecting || EchosounderDetects_stopped) || ((Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))))), is TRUE in the model

Formula number 75: (! E(! K(Depth_control_operation, (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))) U (K(Forward_looking_echosounder, (EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Horizontal_avoidance_strategy, (EchosounderDetects_injecting || EchosounderDetects_stopped) || ((Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))))), is TRUE in the model
Formula number 76: (! E((! K(Depth_control_operation, ((Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))) U (K(Forward_looking_echosounder, (EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Vertical_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))))))), is TRUE in the model

Formula number 77: (! E((! K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))) U (K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Obstacle_clearance_meter, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))))))), is FALSE in the model

Formula number 78: (! E((! K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))) U (K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(New_demand_detection, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))))))), is FALSE in the model

Formula number 79: (! E((! K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))) U (K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Safe_position_meter, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))))))), is TRUE in the model

Formula number 80: (! E((! K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))) U (K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Retreat_path_array, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))))))), is FALSE in the model

Formula number 81: (! E((! K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped)))) U (K(Horizontal_avoidance_strategy, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) && (! (AF K(Avoidance_timer, ((EchosounderDetects_injecting || EchosounderDetects_stopped)) || (Min_Ice_Range_injecting || Min_Ice_Range_stopped) && (Min_Alt_Range_injecting || Min_Alt_Range_stopped))))))), is TRUE in the model
Formula number 82: \(! E(\neg K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))) U (K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))))), is TRUE in the model

Formula number 83: \(! E(\neg K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))) U (K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped)))) && (! (AF K(Retreat\_track\_length\_monitor, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))))), is TRUE in the model

Formula number 84: \(! E(\neg K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))) U (K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped)))) && (! (AF GK(group1, ((EchosounderDetects\_injecting \\
|| EchosounderDetects\_stopped) || ((Min\_Ice\_Range\_injecting \\
|| Min\_Ice\_Range\_stopped) && (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))))))), is TRUE in the model

Formula number 85: \(! E(\neg K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))) U (K(Horizontal\_avoidance\_strategy, \\
(EchosounderDetects\_injecting || EchosounderDetects\_stopped) || \\
((Min\_Ice\_Range\_injecting || Min\_Ice\_Range\_stopped) \&\& (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped)))) && (! (AF GCK(group1, ((EchosounderDetects\_injecting \\
|| Echosounder\_stopped) || ((Min\_Ice\_Range\_injecting \\
|| Min\_Ice\_Range\_stopped) && (Min\_Alt\_Range\_injecting \\
|| Min\_Alt\_Range\_stopped))))))), is FALSE in the model

done, 85 formulae successfully read and checked
execution time = 6548
number of reachable states = 4.65726e+10
BDD memory in use = 212558900