Use MOOS Middleware to Simulate and Formal Methods to Verify the Fault Tolerance of Collaborative Autonomous Underwater Vehicles

Wenxin Wang
Embry-Riddle Aeronautical University
WANGW4@my.erau.edu

Abstract – Simulate and verify the fault tolerance of a real-time system using real objects are time and money consuming. In the processes of implementation and test, the objects might be damaged, thus the mission might fail due to the unexpected reasons such as severe weather problems. One way to improve the efficiency of the fault tolerance implementation and test is to simulate the object communities, communication, and actions of the real-time system and verify the tolerance logic of the fault tolerance by an integrated model checker tool using state machines. The purpose of this paper is to propose the verification and simulation method for the fault tolerance of the Eco-Dolphin, a fleet of collaborating AUVs, using an MOOS-IvP simulator and a UPPAAL verification model checker.

Keywords: Fault Tolerance, Simulation, Verification, collaborative, AUV, MOOS-IvP, UPPAAL, Formal Methods

1. Introduction

1.1 Background

Simulate and verify a real-time system using real objects are time and budget consuming. During the implementation and test phases, dozens of AUVs (Autonomous Underwater Vehicles) might be damaged or even lost. The immature or wrong design of the communication and behavior for the real AUV fleet might cause the mission's failure. Limitations of weather or environment also might negatively affect the implementation and test to a great extent. The researchers might have to wait for a long time before implementing or testing the AUV fleet in the real environment. One way to save cost and enhance the efficiency of the experiment is to simulate the communication and behavior of the AUV fleet in a virtual environment, as well as to verify the communication and behavior logic of the AUV fleet by an integrated virtual model checker with state machines. The fault tolerance would help the AUV fleet improve the mission's performance so it could be fulfilled with the minimal loss of the AUV objects.[1][5]

1.2 Purpose

The purpose of this paper is to verify the fault tolerance and simulate the communication and behavior of the AUV fleet, which belongs to the Eco-Dolphin project in the Society for Industrial & Applied Mathematics (SIAM) laboratory at Embry-Riddle Aeronautical University. Eco-Dolphin is a fleet of 3 collaborative real AUVs.

1.3 Methods

This paper implemented the simulation method using a MOOS (Mission Oriented Operating Suite)-IvP (Interval Programming) simulator, also used UPPAAL as the model checker to verify the behavior logic of the AUV fleet. Compared with the previous verification and simulation concerning the virtual mathematical models[4], this paper is established on a fleet of real AUVs. The C++ modules in the MOOS-IvP autonomy were used to simulate the communities, communication, and behavior of the Eco-Dolphin, as well as manipulate the behavior and activities of the AUV fleet especially when faults occur.

The paper has six parts. The first part just introduces background and targets of the project, as well as the verification and simulation methods. The second part introduces the architecture of the Eco-Dolphin fleet, the MOOS-IvP middleware and its modules, and the behavior models of the Eco-Dolphin fleet. The third part specifies fault tolerance mechanism and why consider fault tolerance for the design of a real-time system. The fourth part demonstrates the simulation of the communities and communication of the AUV fleet, mainly elaborates the behavior models in different situations. The fifth part presents the UPPAAL model checking tool and the state machines which were used to verify the behavior models for the simulated Eco-Dolphin AUV fleet. The sixth part concludes the work of this paper and discusses the future work about the Eco-Dolphin and MOOS-IvP.

2. Eco-Dolphin AUV Fleet

2.1 What is Eco-Dolphin

2.1.1 A Collaborative AUV fleet

The Eco-Dolphin project was designed and implemented in the Society for Industrial & Applied Mathematics (SIAM) laboratory at Embry-Riddle Aeronautical University. *The Eco-Dolphins are configured with an internal attitude control system that combined with propulsion from brushless DC thrusters allows the vehicle to ascend and descend easily. The design, production and assembly of the yellow Eco-Dolphin has been done in twelve months.*[5]

2.1.2 Electronic Modules

There are five electronic modules in the Eco-Dolphin system as shown in figure 1. They are Power Module, Sensor Module, Communication Module, Control Module, and Execution Module. The Power Module carries battery which is the source of electric power. The Sensor Module is divided into two parts. One is the pressure sensor and the other one is the temperature sensor. The Communication Module, where the message is sent back and forth, consists of a sonar modem and a positioning sonar modem. The Control Module could read the data from the Communication Module, thus control the thrusters in the Execution Module.[6]

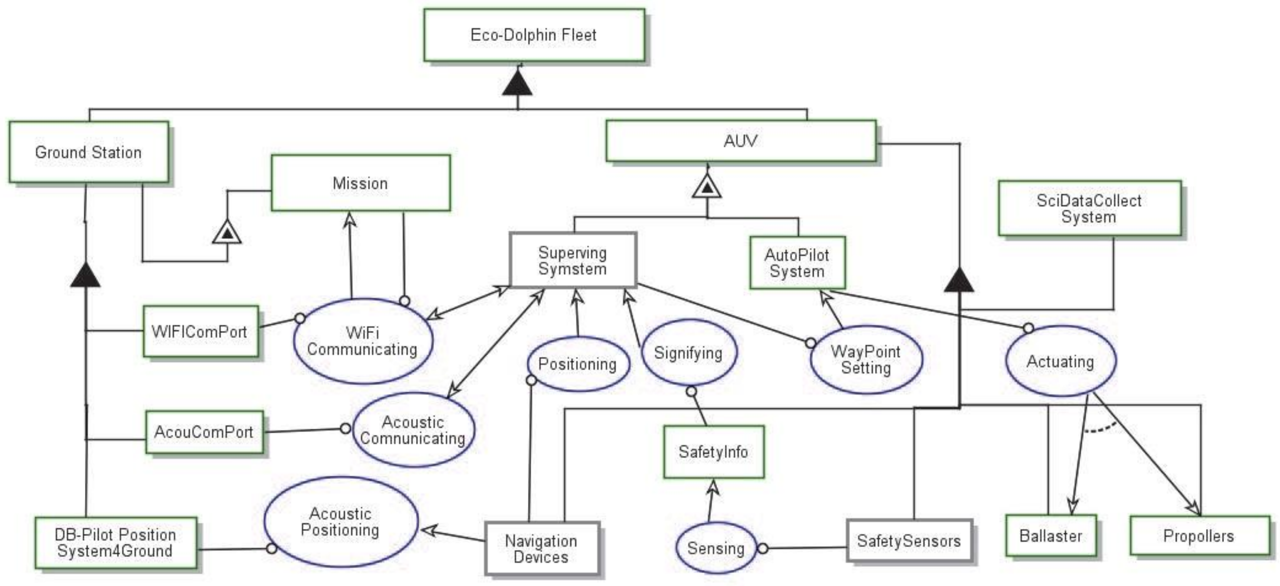


Figure 1. Conception Communication Modules of Eco-Dolphin System

2.2 Simulated AUV Statuses - Formal Method

The simulated Eco-Dolphin is a simulated fleet of three collaborative Autonomous Underwater Vehicles in

MOOS-IvP. The supervising system of Eco-Dolphin system is the first formal method that used to verify the fault tolerance behavior modes as shown in figure 2. [7] There are three different kinds of statuses of an AUV. They are diving, surfacing, and cruising statuses. Theoretically, AUVs could communicate with each other via wi-fi signals in the Eco-Dolphin system. However, when the AUV is far away from the ground station and get into the waters, the wi-fi signals might be hard to catch and read. At this point, the communication signals should be transferred in other forms such as acoustic and X-bee.

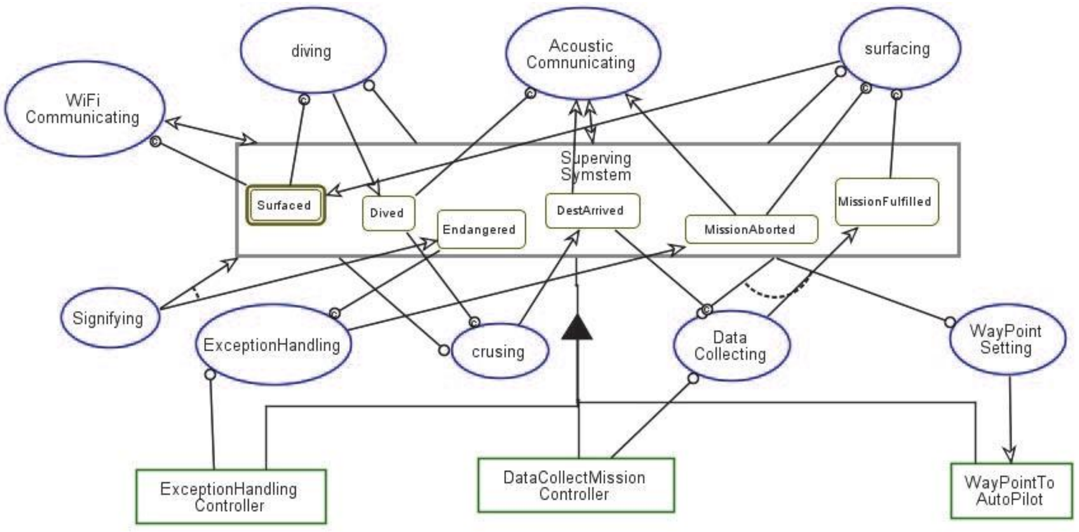


Figure 2. Supervising System of Eco-Dolphin Fleet

2.2.1 Diving (Submerged) Status

The AUV in diving mode will dive into the water to record videos, pictures or sound. Only when the Eco-Dolphin system is executing a mission, the specific AUV would be in this diving status. Otherwise, the AUV would be in other two kinds of conditions.

2.2.2 Surfacing Status

When an AUV is in surfacing status, it will stop on the surface of waters. There are four cases when the AUV should in this status.

- Before diving;
- After mission's abort;
- After completion of the mission;
- The mission supervises this AUV to be in the surfacing status.

2.2.3 Cruising Status

When an AUV is in cruising status, it will sail on the surface of waters. There are four cases when the AUV should in the cruising status.

- After diving task accomplished;
- Before arriving the destination;
- The mission supervises this AUV to be in the cruising status.

3. Fault tolerance Mechanism

3.1.1 Definitions

A system with faults may continue to provide its services, that is, not fail. Such a system is said to be fault tolerant. A fault does not lead the mission to fail unless the result is observable by the user and leads to the mission becoming unable to deliver its specific service. [8] Fault tolerance can be applied at three levels: hardware, software, and system. The purpose of applying fault tolerance is to achieve dependability which means availability, reliability, safety, and security. Approaches to achieve reliability requires efforts at all phases of a system's development, including design phase, implementation phase, execution phase, maintenance phase, and enhancement phase. For a system to be fault tolerant, it must be able to detect, diagnose, confine, mask, compensate and recover from faults. [9] The three situations are Normal, Hazard, and Critical. The situation type classified by the diagnosis results.

3.1.2 Normal Situation

The AUV fleet runs well.

3.1.3 Hazard Situation

If the diagnosis results think the suspended AUV could resume fulfilling the mission, the system will estimate this condition as Hazard.

3.1.4 Critical Situation

If the diagnosis results think the suspended AUV is lost or could not resume fulfilling the mission, the system will estimate this condition as Critical.

3.1.5 13 Behavior Scenes

The actual scenarios are much more than thirteen types Here in order to minimalize the situations and reduce the duplicated scenarios, we settled 13 basis behavior scenes. The Eco-Dolphin fleet contains three collaborative AUVs name *ERAU*, *Eco*, and *Dolphin*. Those three AUVs have different default identifications respectively. Only one of them could submerge using signals transmit via acoustic signals while other two AUVs sail (in cruising or surfacing status) on the surface. Because of the non-sustainability of the underwater transmission acoustic signals, one of the two surface AUVs shall be the communication agent in between to boost the communication signals. In the MOOS-IvP middleware, the type name of this kind of autonomous underwater is *auv (See the type table of the MOOS-IvP middleware)* as shown in the table 1.[10]

Table 1. Platform type values in MOOS-IvP

Platform Type					
kayak	*Mokai*	*uuv*	*Auv*	*Ship*	*glider*

The default identification status of *ERAU* is the main surface AUV. The default identification status of *Dolphin* is the submerged AUV. The default identification status of *Eco* is the second surface AUV which shall be the substitution of the *ERAU* when the main surface AUV out of service, or the *Dolphin* when the submerged AUV does not work well. The protocol model abstracts 13 basic scenes as shown in table 3 in the glossary.

4. MOOS-IvP Simulation

4.1 MOOS-IvP Simulation Tool

4.1.1 The Publish-Subscribe Middleware Architecture

Middleware is computer software that provides services to software applications beyond those available from the operating system. Middleware makes it easier to implement communication and input/output so that the software engineers could focus on the particular purpose of their application. The MOOS-IvP middleware consists of sets of open source C++ modules. [10] These modules simulate the communities including the ground station and the fleet of collaborative AUVs as shown in figure 9. The middleware also simulates the behaviors of each AUV and relatively communication between the ground station and the AUVs in normal, hazard, and critical situation. Simulated the community of the ground station and 3 AUVs name *ERAU*, *Eco*, and Dolphin of the Eco-Dolphin AUV fleet.

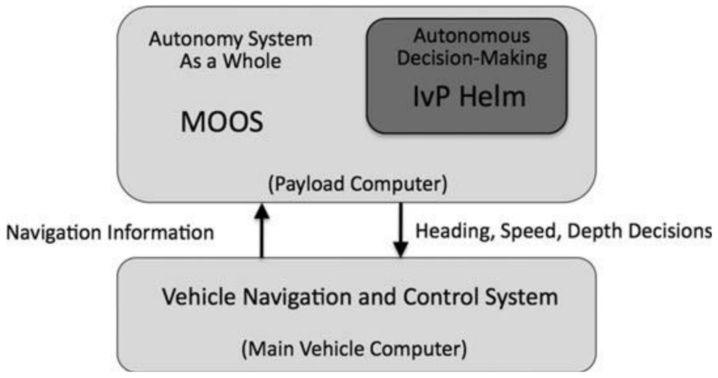

Autonomy System As a Whole
MOOS
Autonomous Decision-Making
IvP Helm
(Payload Computer)
Navigation Information
Heading, Speed, Depth Decisions
Vehicle Navigation and Control System
(Main Vehicle Computer)

Figure 3. The backseat driver paradigm

4.1.2 Components Structures

The MOOS-IvP middleware consists of one MOOS database, several customized applications, and one pHelmIvP core controller as shown in figure 4.[11] The pHelmIvP contains a C++ module which controls the basic communication and behaviors. The customized applications implemented in the project are shown in figure 5. Each application manipulates one corresponding communication or behavior model. The applications of the simulated fleet in MOOS-IvP GUI as shown on the right of figure 9. [12][13]

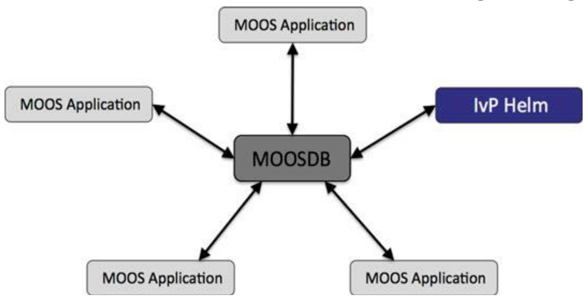


Figure 4. Structure of MOOS-IvP applications

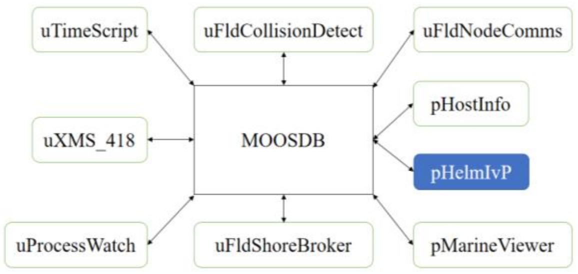
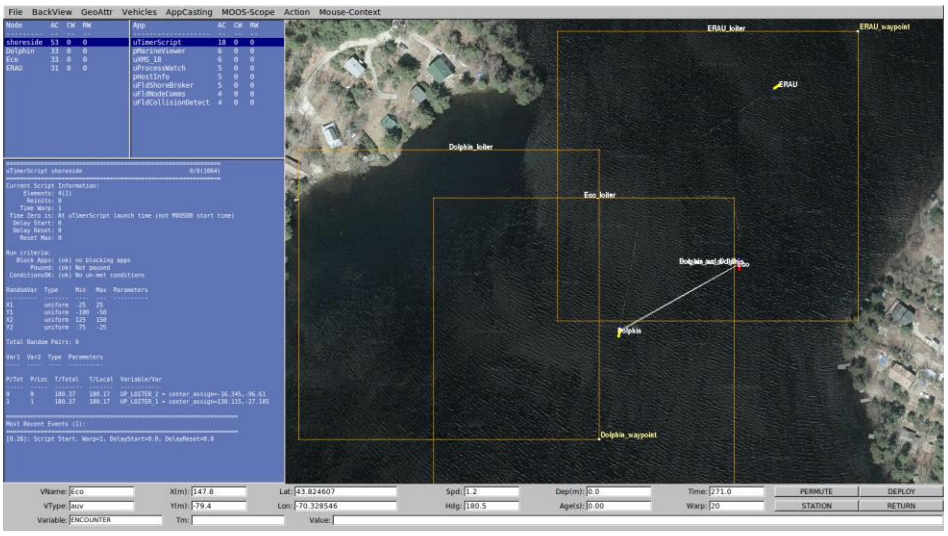


Figure 5. Customized applications of the MOOS-IvP

4.2 Implementation

The reliability has been considered during the early design strategy of the development of the Eco-Dolphin system. To complete the mission, the AUV fleet has to be available and reliable. For example, whenever one of the other two AUVs loses connection due to communication failure, the spare AUV *Eco* could change its tasks to substitute for the missing AUV so the mission could still be fulfilled. To be mission efficient, the system shall protect the safety of each AUV. For example, no matter which AUV loses connection with the ground station or other AUVs, the lost AUV should suspend the activity automatically, waiting for further self-diagnosis or instruction signals from the ground station. UPPAAL models help the AUV fleet verify the dependability of the logic behavior modes during the design strategy using state machines. It would be mission efficient because using the virtual models and environment instead of real AUVs.[14]

The simulated AUV fleet shall be able to detect, diagnose, and compensate the fault. Each AUV could detect the faults of other connected AUVs by the communication channels which are a set of applications between each AUV community as shown in figure 5.[15] At this time, the sensor shall also suspend the activity of the AUV where the fault occurred. Then the sensor of each AUV shall diagnose the fault to determine what caused the failure, or exactly which subsystem or component is dysfunctional. Depending on the diagnosis results, the sensor shall be able to recover the AUV's activities or send the message to the other AUVs to reschedule the activities. The user could manipulate the activities of the simulated AUV fleet by clicking the control buttons on the GUI as shown in figure 6. There are four types of manipulation. They are *permute*, *station*, *deploy*, and *return* as shown in figure 7. When the AUV is in hazard or critical status, the user could manipulate manually to substitute the AUV with other substitution AUVs. The user could compensate the AUV fleet’s mission by implementing those operations.



```
Node        AC  CW  RW
---------   --  --  --
shoreside   53  0   0
Dolphin     33  0   0
Eco         33  0   0
ERAU        31  0   0

App                    AC  CW  RW
-------------------    --  --  --
uTimerScript           18  0   0
pMarineViewer          6   0   0
uXMS_18                6   0   0
uProcessWatch          5   0   0
pHostInfo              5   0   0
uFldShoreBroker        5   0   0
uFldNodeComms          4   0   0
uFldCollisionDetect    4   0   0

==============================================================
uTimerScript shoreside                              0/0(1064)
==============================================================
Current Script Information:
     Elements: 4(2)
      Reinits: 0
    Time Warp: 1
 Time Zero is: At uTimerScript launch time (not MOOSDB start time)
  Delay Start: 0
  Delay Reset: 0
    Reset Max: 0

Run criteria:
   Block Apps: (ok) no blocking apps
       Paused: (ok) Not paused
 ConditionsOK: (ok) No un-met conditions

RandomVar  Type     Min   Max  Parameters
---------  -------  ----  ---  ----------
X1         uniform  -25   25
Y1         uniform  -100  -50
X2         uniform  125   150
Y2         uniform  -75   -25

Total Random Pairs: 0

Var1  Var2  Type  Parameters
----  ----  ----  ----------

P/Tot  P/Loc  T/Total   T/Local  Variable/Var
-----  -----  --------  -------  ------------
0      0      180.37    180.17   UP_LOITER_2 = center_assign=-16.345,-96.61
1      1      180.37    180.17   UP_LOITER_1 = center_assign=130.115,-27.185

==================================================================
Most Recent Events (1):
==================================================================
[0.26]: Script Start. Warp=1, DelayStart=0.0, DelayReset=0.0
```

```
VName: Eco        X(m): 147.8   Lat: 43.824607    Spd: 1.2     Dep(m): 0.0    Time: 271.0
VType: auv        Y(m): -79.4   Lon: -70.328546   Hdg: 180.5   Age(s): 0.00   Warp: 20
Variable: ENCOUNTER   Tm:        Value:
```

Figure 6. Figure - MOOS-IvP GUI

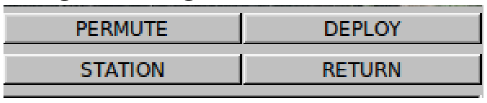
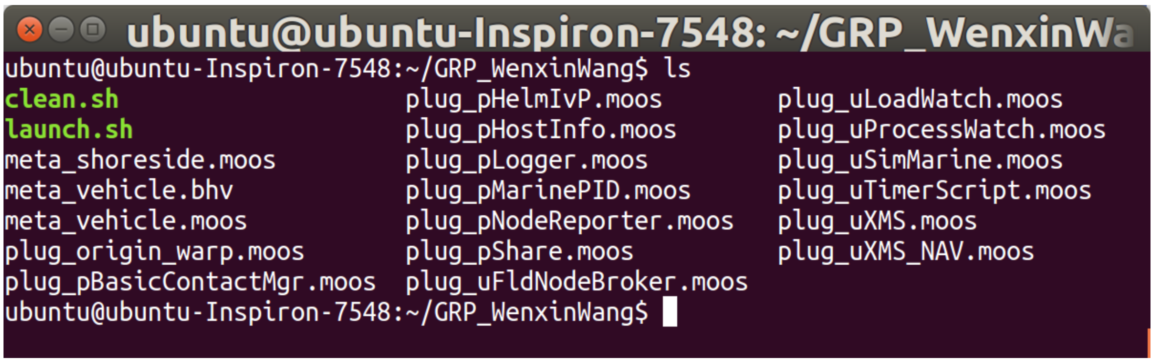


Figure 7. User Operation Buttons

4.2.1 Configuration Files

There are three primary types of configuration files. They are moos files, behavior files, and shell script files as shown in figure 8.[16] All configuration parameters and the corresponding values define different simulated communities, MOOS-IvP applications which control the AUVs’ communication and networking, and corresponding behavior modes under different situations. The shell script files enhance the execution efficiency by being run by the Unix Shell which is a command-line interpreter. [17]



```
ubuntu@ubuntu-Inspiron-7548: ~/GRP_WenxinWa
ubuntu@ubuntu-Inspiron-7548:~/GRP_WenxinWang$ ls
clean.sh                     plug_pHelmIvP.moos        plug_uLoadWatch.moos
launch.sh                    plug_pHostInfo.moos       plug_uProcessWatch.moos
meta_shoreside.moos          plug_pLogger.moos         plug_uSimMarine.moos
meta_vehicle.bhv             plug_pMarinePID.moos      plug_uTimerScript.moos
meta_vehicle.moos            plug_pNodeReporter.moos   plug_uXMS.moos
plug_origin_warp.moos        plug_pShare.moos          plug_uXMS_NAV.moos
plug_pBasicContactMgr.moos   plug_uFldNodeBroker.moos
ubuntu@ubuntu-Inspiron-7548:~/GRP_WenxinWang$
```

Figure 8. Files of the simulator in MOOS-IvP

4.2.1.1 Moos Files

The moos files end with .moos define all the MOOS-IvP configurations.

- Initialize the default behavior parameters,
- Define the behavior mode,
- Configure the corresponding behavior parameter values for each behavior mode.

4.2.1.2 Behavior Files

The behavior files end with .bhv define all the parameter values for different behavior modes.

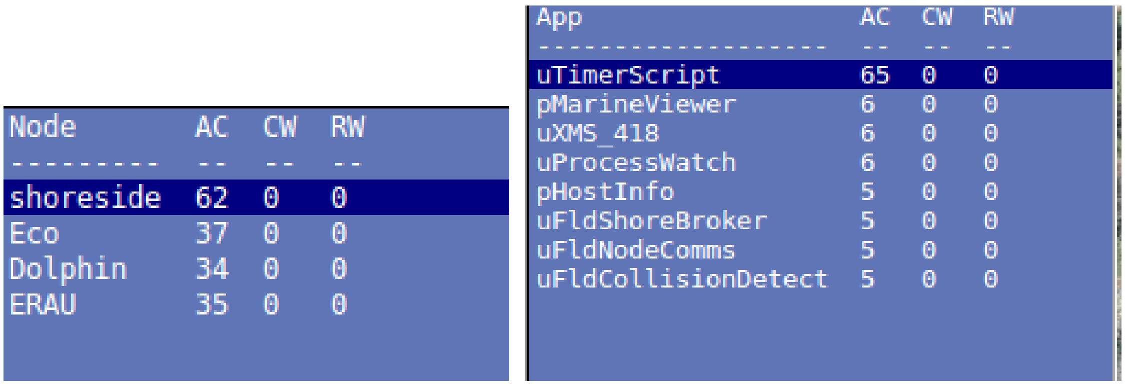
- Initialize the default behavior parameters,
- Define the behavior mode,
- Configure the corresponding behavior parameter values for each behavior mode.

4.2.1.3 Shell Script Files

The shell script files end with .*sh* manipulate the execution of the program.[18] The shell script files wrap multiple operations such as set up the environment, run the MOOS-IvP program, and display the GUI of the MOOS-IvP.

4.2.2 Communities and Applications

The communities and communications of the ground station and 3 AUVs in MOOS-IvP as shown on the left of figure 9. The applications of the simulated fleet in MOOS-IvP GUI as shown on the right of figure 9. The listening ports and sending ports of AUVs are listed as shown in table 2. Each AUV and the ground station have their own port for listening. In this case, all the fleet use the same host. In other cases, there could be multiple host addresses for different fleet communities. The more details and values of each application are listed in the figure 17-22 in the glossary at the end.[19]



```
Node        AC   CW   RW
---------   --   --   --
shoreside   62   0    0
Eco         37   0    0
Dolphin     34   0    0
ERAU        35   0    0
```

```
App                  AC   CW   RW
-------------------  --   --   --
uTimerScript         65   0    0
pMarineViewer        6    0    0
uXMS_418             6    0    0
uProcessWatch        6    0    0
pHostInfo            5    0    0
uFldShoreBroker      5    0    0
uFldNodeComms        5    0    0
uFldCollisionDetect  5    0    0
```

Figure 9. MOOS-IvP Communities and Applications

Table 2. Community Parameters

Name	VPORT	Share_Listen Port
ERAU	9001	9301
Eco	9002	9302
Dolphin	9003	9303
Shore	9000	9300

4.3 Simulated Behavior Statuses in 3 Situations

4.3.1 Normal Situation

In the normal situation, the AUV fleet shall execute the mission collaboratively as shown in figure 10 and figure 11.

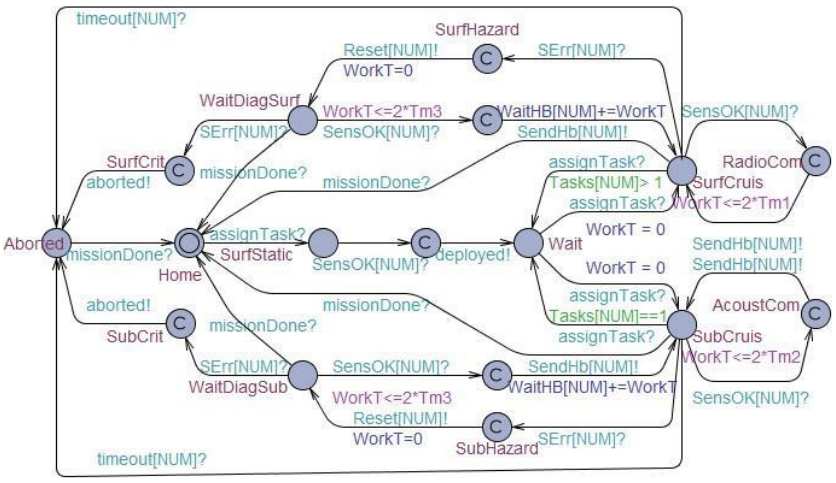


Figure 10. The protocols of 3 AUVs' activities

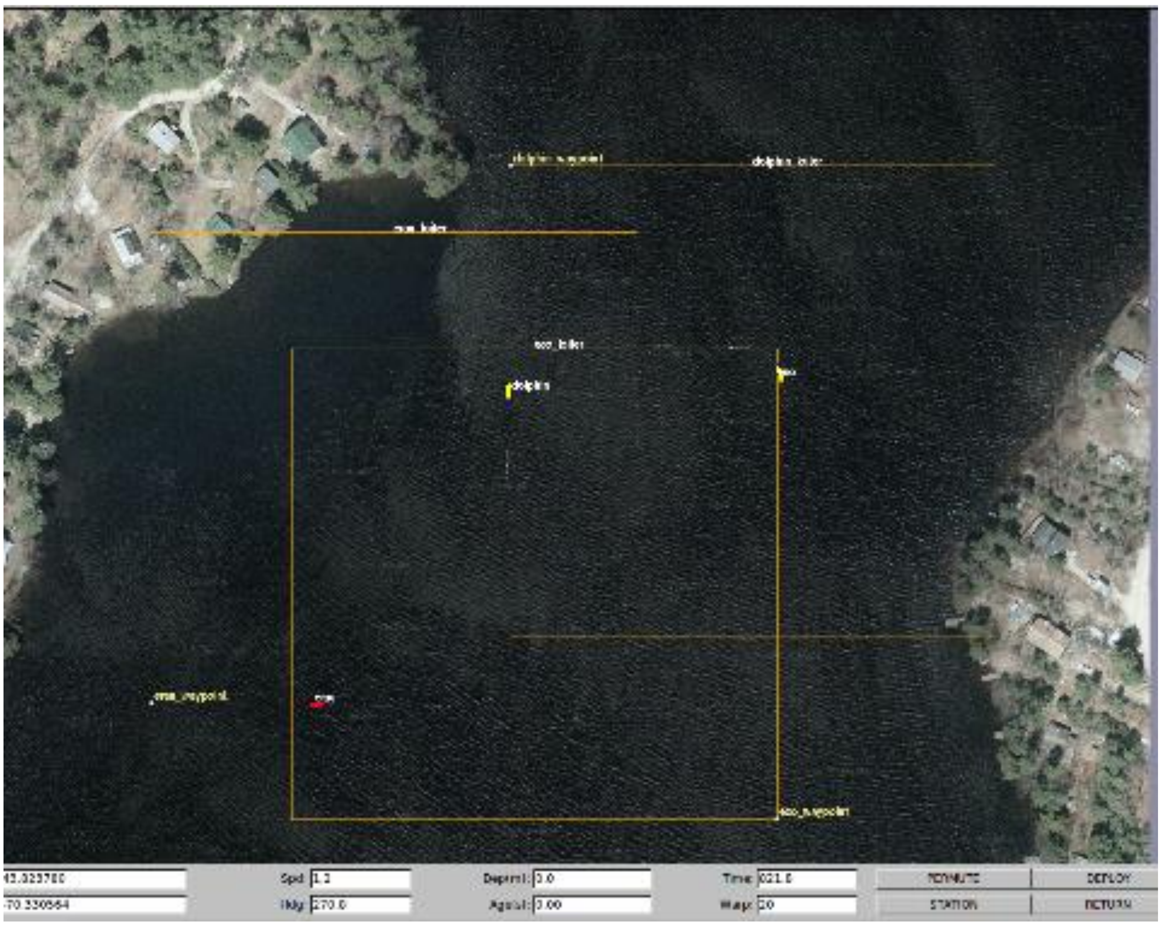


Figure 11. AUVs works in the regular mission

4.3.2 Hazard Situation

There are three main kinds of hazard fault in the simulation. They are *temperature*, *humidity*, and *timeout*. For example, when an AUV is in SubHazard situation, the state of the sensor would go to the hazard point correspondingly as shown in figure 12 and figure 13.

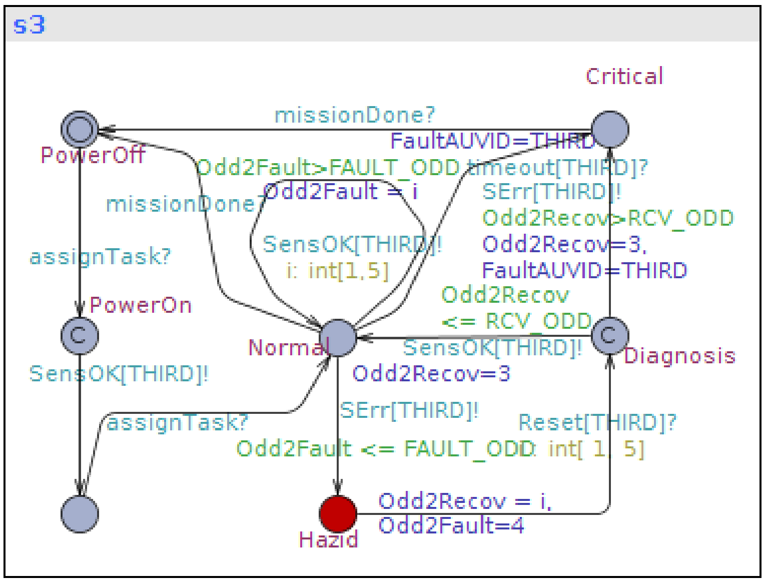


Figure 12. Hazard situation in UPPAAL model

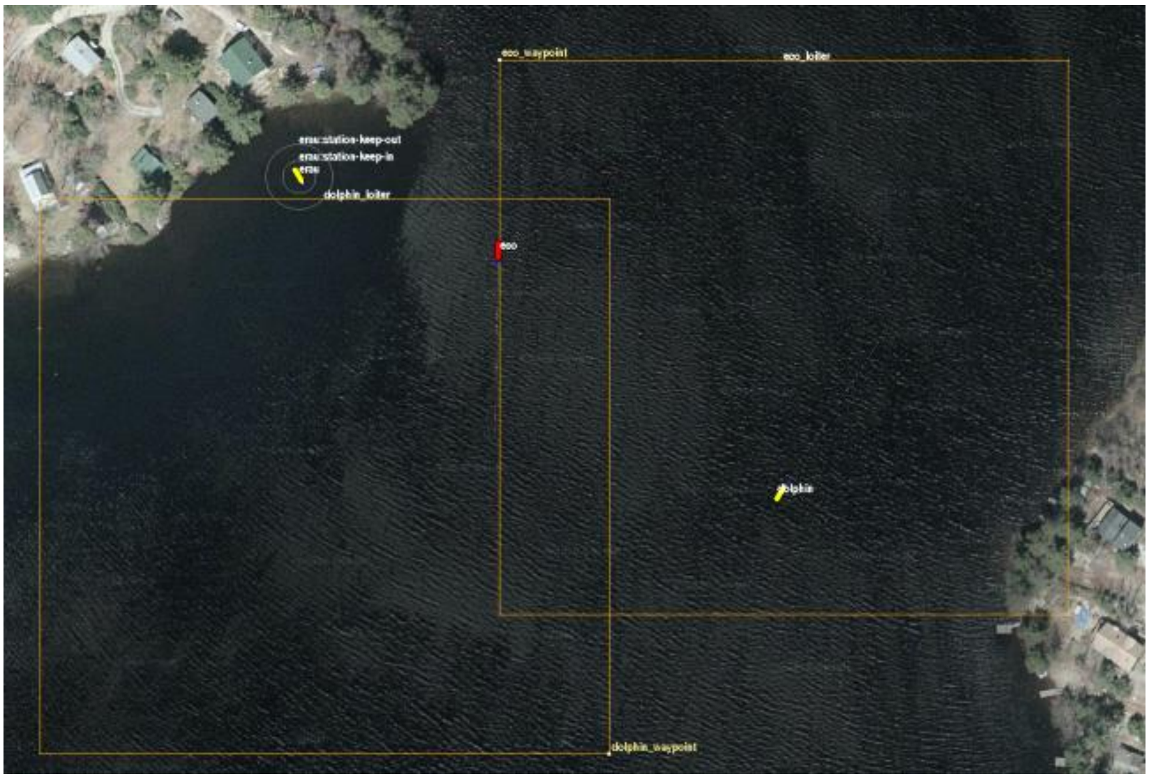


Figure 13. One AUV stops while other two reassign the mission.

4.3.3 Critical Situation

Once the AUV is under the hazard situation, the system would wait for another time period to diagnose the fault and figure out whether the fault could be recovered or not as shown in figure 14 below. The length of the time period depends on the time clock and the design of the scenarios. If there is no further response which means the communication lost or the diagnosis results display the fault could not recover, the state of the AUV would be turned into the critical situation. At this time, the AUV would be aborted as shown in figure 15 below. Later the mission will be aborted if there is not enough AUVs left. If there are more than one AUV left, the mission could still be executed as shown in figure 16.

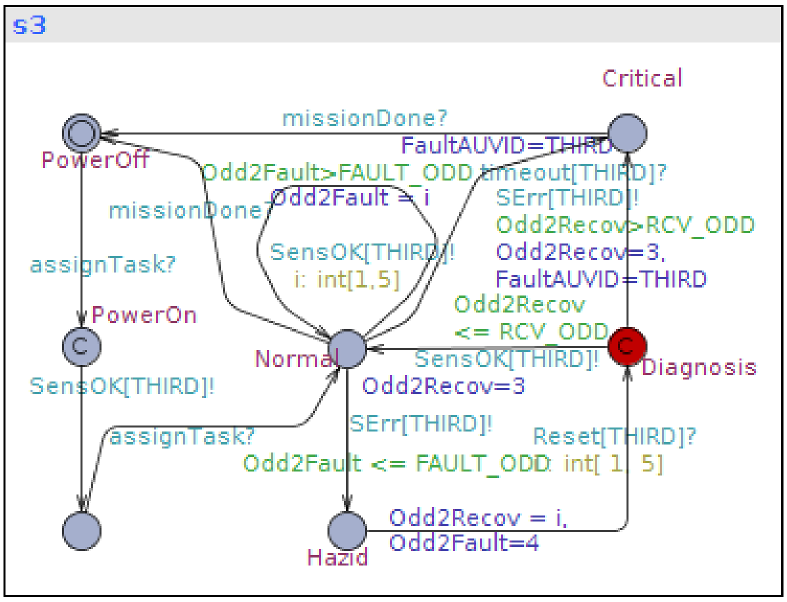


Figure 14. Diagnosis time period in UPPAAL model

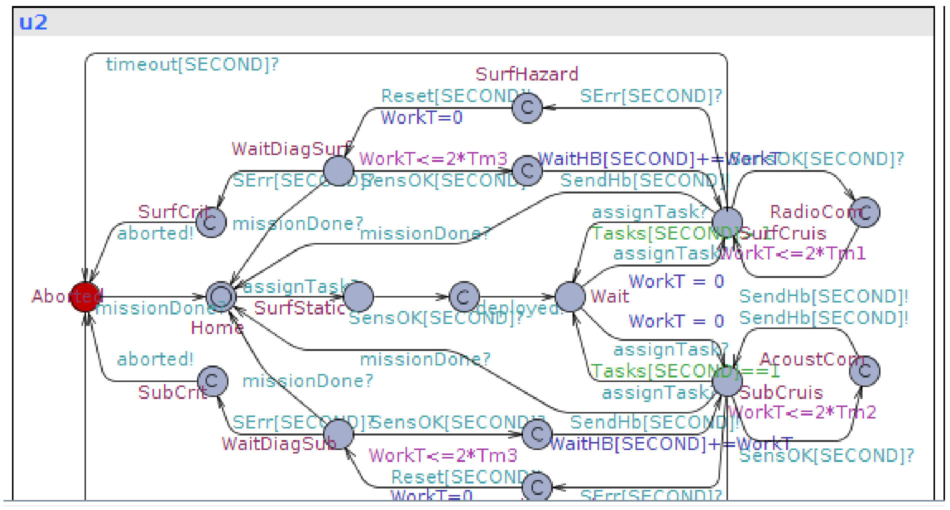


Figure 15. Diagnosis time period in UPPAAL model

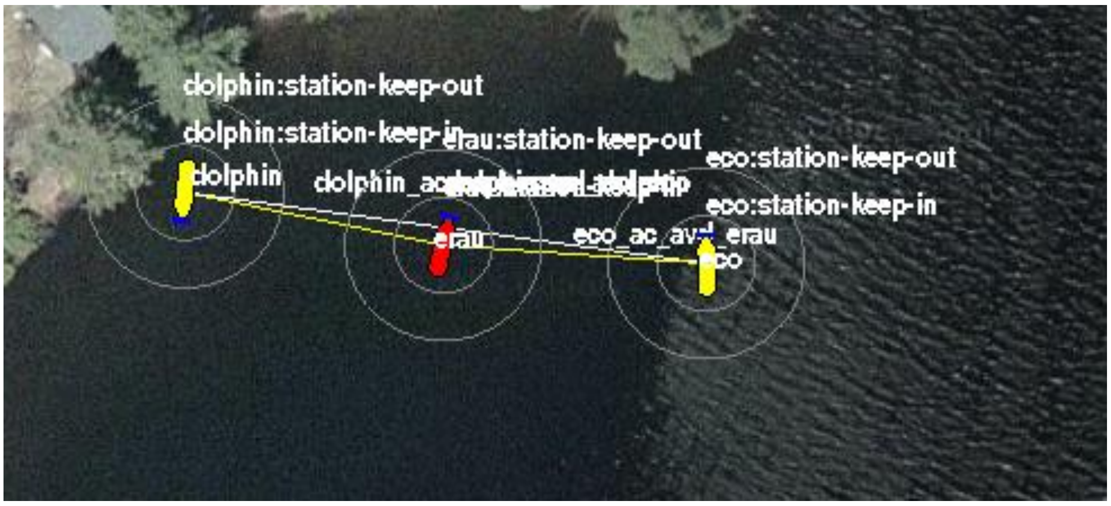


Figure 16. All 3 AUVs stop working

5. Formal Method Verification

5.1 Why using the formal method to verify?

There are endless cases to verify due to the infinite loop of different permutations and combinations. This kind of situation calls state explosion. Brain power could not satisfy the requirements of modeling and checking all the possible situations. The formal method is a useful mathematical verification method which helps build the abstract logic model for the collaborative system to run and check the logic state machine of the design. The tool will enhance the test and implementation efficiency. It could also automatically and randomly verify the behavior models in different situations so the checking processes become easier and quicker.

5.2 UPPAAL Tool

The model checking tool UPPAAL proposes a model checking state machine. [20] The logic sequence results from running the UPPAAL verification model will be utilized as the fault tolerance behavior rules of the collaborative AUV fleet under the fault situation.[21] On the basis of the logic in the UPPAAL model as shown in figure, implemented the state machine of the UPPAAL model with different functions and behaviors in the MOOS-IvP middleware.

Model properties are expressed using a subset of computational tree logic (CTL) without nested temporal operators.

Clocks can be used to check bounded time responses. [22] The fault tolerance shall present the corresponding error handling action. When the counter examples against the right state machines proposed by the model checking tool appear, the proposed project shall effectively display an error message and prompt the user related information about the error. Therefore, the user could make appropriate further inspection or testing based on the prompt. This handling process will reliably verify the fault tolerant control protocol of the simulated AUVs' system on a virtual MOOS-IvP platform to save human and material resources. [23]

5.3 State Machine Models

5.3.1 AUV fault handling models

The AUV fault handling models describes the logic state machines of each AUV as shown in the figure 17 below. Every AUV will checked in this kind of model collaboratively. [24] The circle represents each state of the execution. When the AUV is running, the representative circle would be in red color to show which kind of status the AUV should be at. When the AUV is in the normal situation, the red circle goes to the next point in the sequences. Depending on the fault mission configuration of each AUV in the fleet, the AUV shall go into the *SurfCruis* or the *SubCruis* state. When there occurs any communication interruption such as timeout, the AUV shall go into the *wait* status for further diagnosis. Depending on the diagnosis result, the red circle might go to the hazard, critical, or back to normal work state. Once the AUV is in critical state, the red circle would go to the *aborted* state and wait for further instructions or being rescued by human being manually. Then the other AUV(s) would be reallocate the new tasks if the other two AUVs run normally. Once there are less than two AUVs in the normal situation, the mission would fail because there will lack AUVs to finish the mission.

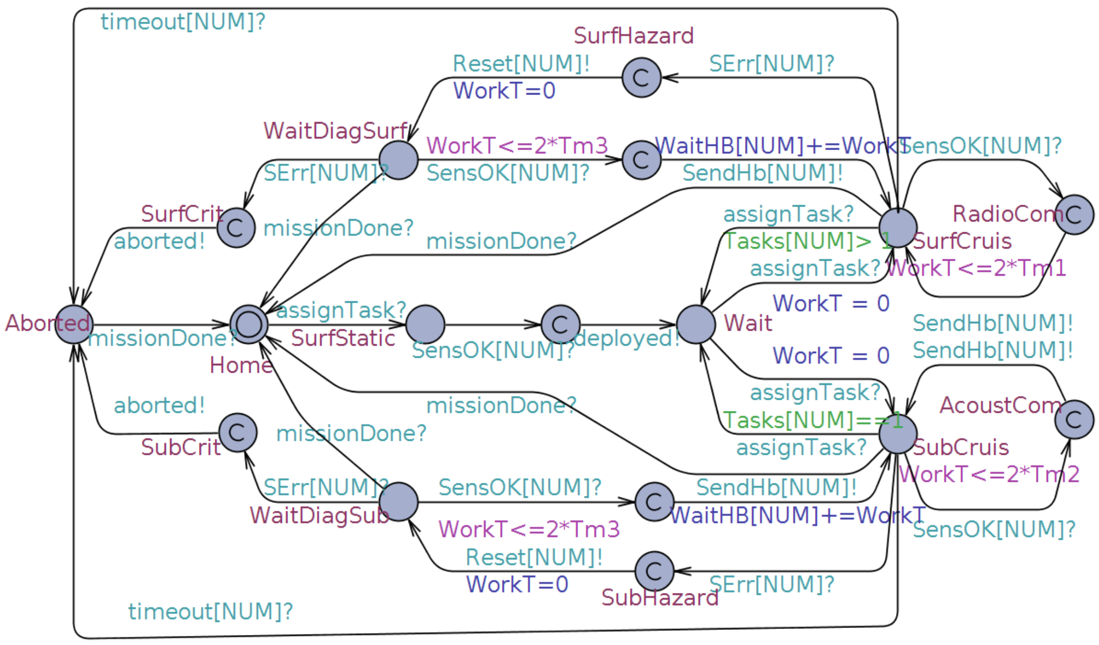


Figure 17. The AUV model in UPPAAL

5.3.2 Sensor fault tolerance models

There are mainly four types of states of the sensor for each AUV. They are normal, critical, diagnosis, and hazard as shown in figure 13 below. The sensor model would work with the previous AUV model in figure 18. For example, when an AUV is in SubHazard situation, the state of the sensor would go to the hazard point correspondingly.

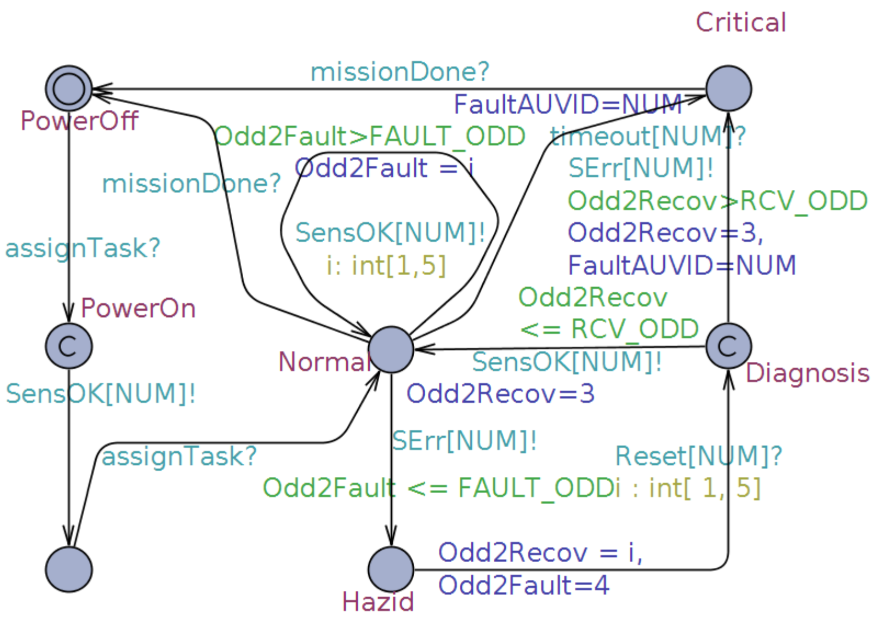


Figure 18. The sensor model in UPPAAL

5.4 Verification Results

With the help of the UPPAAL state machine models, there still need explosive time and space to calculate the real execution cases. Mostly, the verifier of the UPPAAL tool could not figure out how much exact scenarios of the fleets' communication because the

6. Conclusion and Future Work

6.1 Conclusion

In the research, we verified and simulated the fault-tolerant control protocols in UPPAAL and MOOS-IvP middleware. The two tools simulated the execution environment of the AUV fleet and the corresponding environment in the real world. The tools indeed enhance the efficiency of the test and the implementation from the design strategy.

6.2 Future Work

In the future, the MOOS-IvP model could implemented on the Eco-Dolphin fleet using the micro-processors and sensors embedded on the AUV. The MOOS-IvP has the wrapper layer which could wrap those C++ code into the basic programming language which could manipulate the hardware work of the real AUV fleet objects. The next step is to collect the data and statistics from the real execution environment scenarios. Because the real environment is much more complex than the simulation environment. There are much more boundary cases that need to be collected and analyzed. We also need help from other professional vehicle researchers to instruct the fleet activities and the communication. The professional researchers could also help us know the reasons which might cause hazard or critical fault. Then analyze the sorts of different types hazard and critical reasons and their corresponding boundary cases. Once we get more details of the boundary conditions, the MOOS-IvP platform could help us implement the simulation environment on the real AUV fleet using the interfaces between the software layer and hardware layer as shown in figure 19 based on the communication structures. [25]

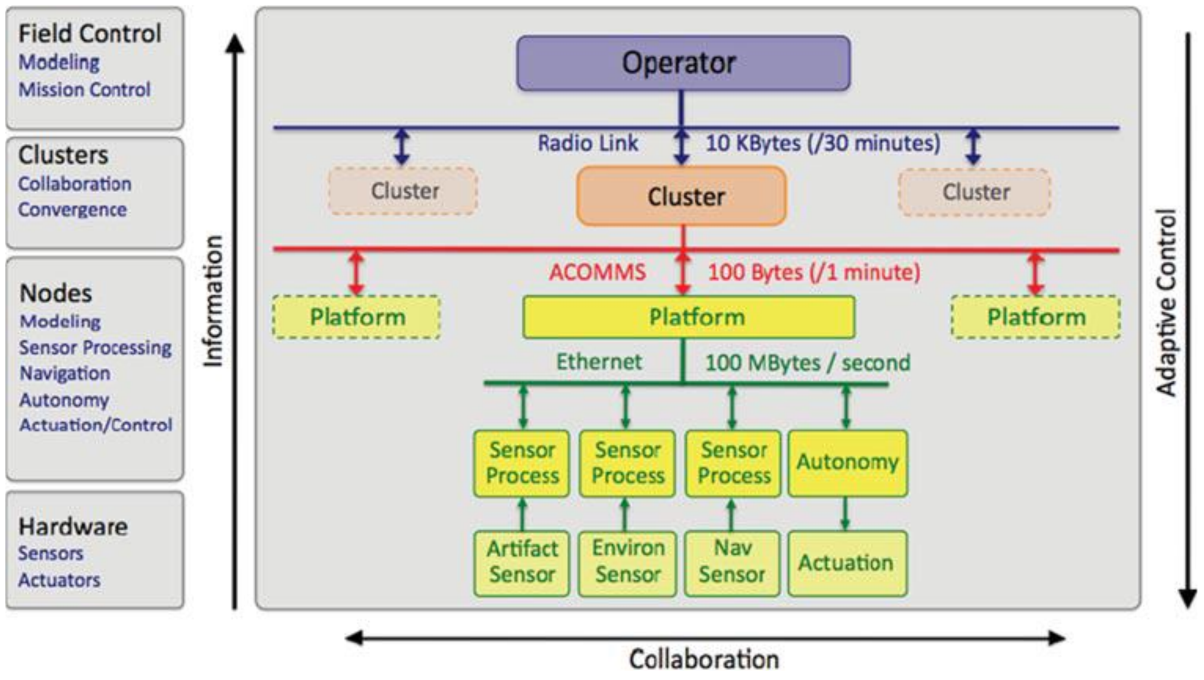
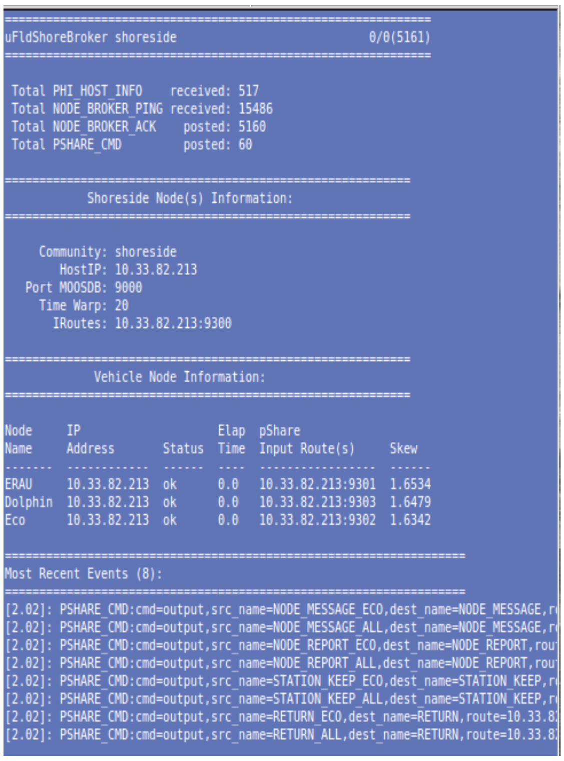


Figure 19. The communication structures

Glossary

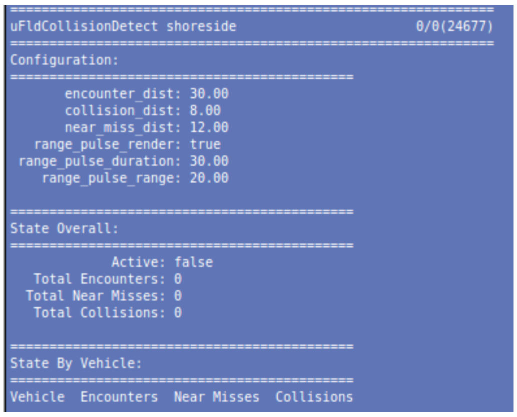
Table 3. 13 Default Scenarios

No.	Type	State	Event	Action
1	*Normal*	Three AUVs work normally	------------------------------	
2	*Hazard*	*ERAU* occurs hazard status while other two AUVs work normally	Diagnosis recovers	All three AUVs continue working
3		*Eco* occurs hazard status while other two AUVs work normally	Diagnosis recovers	All three AUVs continue working
4		*Dolphin* occurs hazard status while other 2 AUVs work normally	Diagnosis recovers	All three AUVs continue working
5		*ERAU* occurs hazard status while other 1 AUV works normally	Diagnosis recovers	All two AUVs continue working
6		*Eco* occurs hazard status while other 1 AUV works normally	Diagnosis recovers	All two AUVs continue working
7		*Dolphin* occurs hazard status while other 1 AUV works normally	Diagnosis recovers	All two AUVs continue working
8	*Critical*	*ERAU* occurs hazard status while other 2 AUVs work normally	Diagnosis turns to Critical	*Eco* stops to substitute for the *ERAU*
9		*Eco* occurs hazard status while other 2 AUVs work normally	Diagnosis turns to Critical	*ERAU* and Dolphin continue working
10		*Dolphin* occurs hazard status while other 2 AUVs work normally	Diagnosis turns to Critical	*Eco* stops to substitute for the *Dolphin*
11		*ERAU* occurs hazard status while other 1 AUV work normally	Diagnosis turns to Critical	All two AUVs stop to return home
12		*Eco* occurs hazard status while other 1 AUV works normally	Diagnosis turns to Critical	All two AUVs stop to return home
13		*Dolphin* occurs hazard status while other 1 AUV works normally	Diagnosis turns to Critical	All two AUVs stop to return home



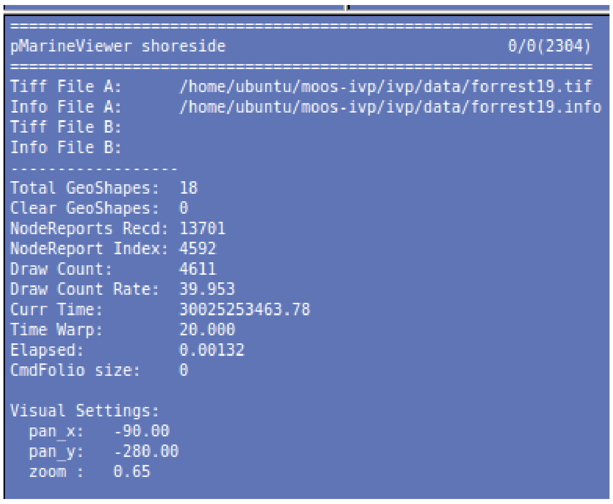
```
==============================================================
uFldShoreBroker shoreside                             0/0(5161)
==============================================================

 Total PHI_HOST_INFO    received: 517
 Total NODE_BROKER_PING received: 15486
 Total NODE_BROKER_ACK    posted: 5160
 Total PSHARE_CMD         posted: 60

===========================================================
            Shoreside Node(s) Information:
===========================================================

     Community: shoreside
        HostIP: 10.33.82.213
   Port MOOSDB: 9000
     Time Warp: 20
       IRoutes: 10.33.82.213:9300

===========================================================
             Vehicle Node Information:
===========================================================

Node     IP                   Elap  pShare
Name     Address       Status Time  Input Route(s)     Skew
-------  ------------  ------ ----  -----------------  ------
ERAU     10.33.82.213  ok     0.0   10.33.82.213:9301  1.6534
Dolphin  10.33.82.213  ok     0.0   10.33.82.213:9303  1.6479
Eco      10.33.82.213  ok     0.0   10.33.82.213:9302  1.6342

===================================================================
Most Recent Events (8):
===================================================================
[2.02]: PSHARE_CMD:cmd=output,src_name=NODE_MESSAGE_ECO,dest_name=NODE_MESSAGE,r
[2.02]: PSHARE_CMD:cmd=output,src_name=NODE_MESSAGE_ALL,dest_name=NODE_MESSAGE,r
[2.02]: PSHARE_CMD:cmd=output,src_name=NODE_REPORT_ECO,dest_name=NODE_REPORT,rou
[2.02]: PSHARE_CMD:cmd=output,src_name=NODE_REPORT_ALL,dest_name=NODE_REPORT,rou
[2.02]: PSHARE_CMD:cmd=output,src_name=STATION_KEEP_ECO,dest_name=STATION_KEEP,r
[2.02]: PSHARE_CMD:cmd=output,src_name=STATION_KEEP_ALL,dest_name=STATION_KEEP,r
[2.02]: PSHARE_CMD:cmd=output,src_name=RETURN_ECO,dest_name=RETURN,route=10.33.8
[2.02]: PSHARE_CMD:cmd=output,src_name=RETURN_ALL,dest_name=RETURN,route=10.33.8
```

Figure 20. uFldShoreBroker parameters



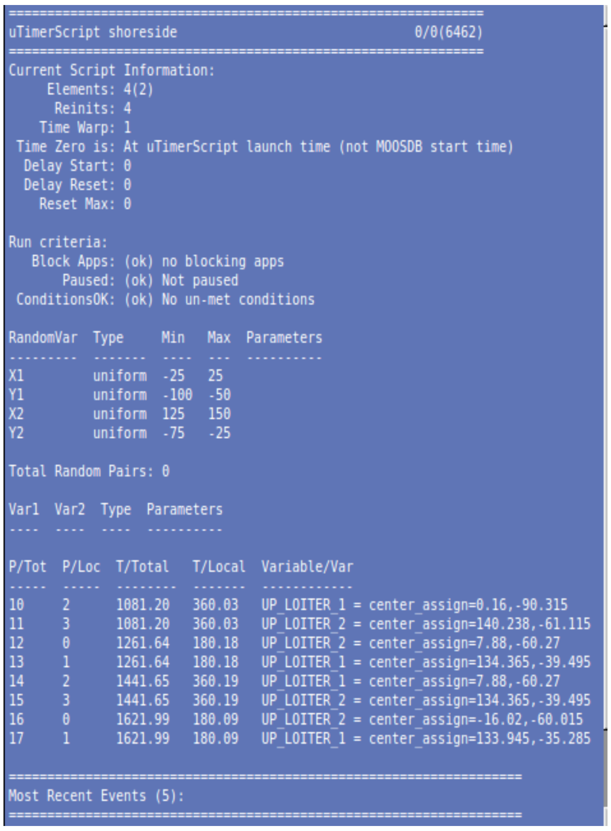
```
==============================================================
uFldCollisionDetect shoreside                       0/0(24677)
==============================================================
Configuration:
============================================
       encounter_dist: 30.00
       collision_dist: 8.00
       near_miss_dist: 12.00
   range_pulse_render: true
 range_pulse_duration: 30.00
    range_pulse_range: 20.00

============================================
State Overall:
============================================
             Active: false
   Total Encounters: 0
  Total Near Misses: 0
   Total Collisions: 0

============================================
State By Vehicle:
============================================
Vehicle  Encounters  Near Misses  Collisions
```

Figure 21. uFldCollisionDetect parameters



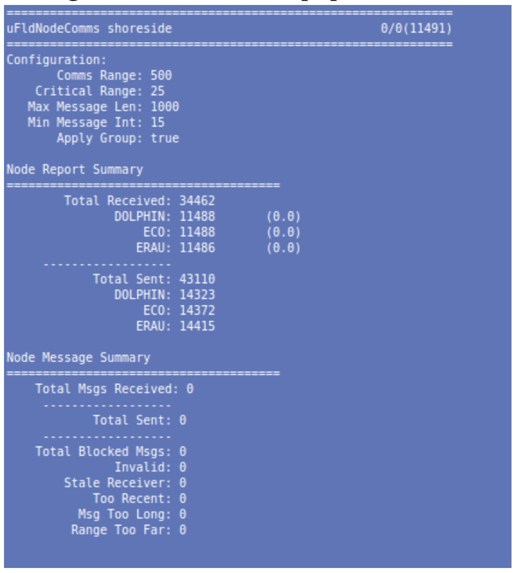
```
==============================================================
pMarineViewer shoreside                               0/0(2304)
==============================================================
Tiff File A:       /home/ubuntu/moos-ivp/ivp/data/forrest19.tif
Info File A:       /home/ubuntu/moos-ivp/ivp/data/forrest19.info
Tiff File B:
Info File B:
------------------
Total GeoShapes:   18
Clear GeoShapes:   0
NodeReports Recd:  13701
NodeReport Index:  4592
Draw Count:        4611
Draw Count Rate:   39.953
Curr Time:         30025253463.78
Time Warp:         20.000
Elapsed:           0.00132
CmdFolio size:     0

Visual Settings:
  pan_x:   -90.00
  pan_y:   -280.00
  zoom :   0.65
```

Figure 22. pMarineViewer parameters



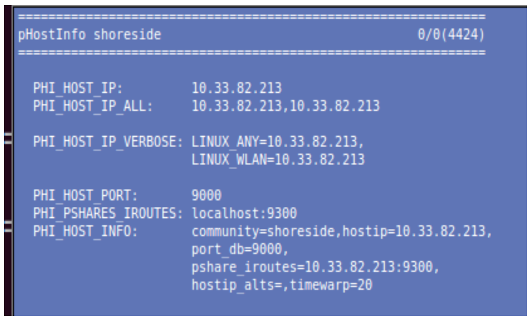
```
==============================================================
uTimerScript shoreside                                0/0(6462)
==============================================================
Current Script Information:
     Elements: 4(2)
      Reinits: 4
    Time Warp: 1
 Time Zero is: At uTimerScript launch time (not MOOSDB start time)
  Delay Start: 0
  Delay Reset: 0
    Reset Max: 0

Run criteria:
   Block Apps: (ok) no blocking apps
       Paused: (ok) Not paused
 ConditionsOK: (ok) No un-met conditions

RandomVar  Type     Min   Max  Parameters
---------  -------  ----  ---  ----------
X1         uniform  -25   25
Y1         uniform  -100  -50
X2         uniform  125   150
Y2         uniform  -75   -25

Total Random Pairs: 0

Var1  Var2  Type  Parameters
----  ----  ----  ----------

P/Tot  P/Loc  T/Total   T/Local  Variable/Var
-----  -----  --------  -------  ------------
10     2      1081.20   360.03   UP_LOITER_1 = center_assign=0.16,-90.315
11     3      1081.20   360.03   UP_LOITER_2 = center_assign=140.238,-61.115
12     0      1261.64   180.18   UP_LOITER_2 = center_assign=7.88,-60.27
13     1      1261.64   180.18   UP_LOITER_1 = center_assign=134.365,-39.495
14     2      1441.65   360.19   UP_LOITER_1 = center_assign=7.88,-60.27
15     3      1441.65   360.19   UP_LOITER_2 = center_assign=134.365,-39.495
16     0      1621.99   180.09   UP_LOITER_2 = center_assign=-16.02,-60.015
17     1      1621.99   180.09   UP_LOITER_1 = center_assign=133.945,-35.285

===================================================================
Most Recent Events (5):
===================================================================
```

Figure 23. uTimerScript parameters



```
==============================================================
uFldNodeComms shoreside                              0/0(11491)
==============================================================
Configuration:
       Comms Range: 500
    Critical Range: 25
   Max Message Len: 1000
   Min Message Int: 15
       Apply Group: true

Node Report Summary
======================================
        Total Received: 34462
               DOLPHIN: 11488       (0.0)
                   ECO: 11488       (0.0)
                  ERAU: 11486       (0.0)
     ------------------
            Total Sent: 43110
               DOLPHIN: 14323
                   ECO: 14372
                  ERAU: 14415

Node Message Summary
======================================
    Total Msgs Received: 0
     ------------------
            Total Sent: 0
     ------------------
    Total Blocked Msgs: 0
               Invalid: 0
        Stale Receiver: 0
            Too Recent: 0
          Msg Too Long: 0
         Range Too Far: 0
```

Figure 24. uFldNodeComms parameters



```
==============================================================
pHostInfo shoreside                                   0/0(4424)
==============================================================

  PHI_HOST_IP:         10.33.82.213
  PHI_HOST_IP_ALL:     10.33.82.213,10.33.82.213

  PHI_HOST_IP_VERBOSE: LINUX_ANY=10.33.82.213,
                       LINUX_WLAN=10.33.82.213

  PHI_HOST_PORT:       9000
  PHI_PSHARES_IROUTES: localhost:9300
  PHI_HOST_INFO:       community=shoreside,hostip=10.33.82.213,
                       port_db=9000,
                       pshare_iroutes=10.33.82.213:9300,
                       hostip_alts=,timewarp=20
```

Figure 25. pHostInfo parameters

Reference

[1] Liu, Hong, et al. "The mechatronic system of Eco-Dolphin—A fleet of autonomous underwater vehicles." Advanced Mechatronic Systems (ICAMechS), 2015 International Conference on. IEEE, 2015.

[2] Hong Liu, T. Yang and J. Wang, "Model Checking for the Fault Tolerance of Collaborative AUVs," 2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE), Orlando, FL, 2016, pp. 244-245

[3] Hong Liu, Modeling and Verifying the Communication and Control of a Fleet of Collaborative Autonomous Underwater Vehicles, 2017.Modeling and Verifying the Communication and Control of a Fleet of Collaborative Autonomous Underwater Vehicles, Hong Liu.

[4] Gao, Pu, Hong P. Liua, and David P. Gluch. "On modelling, simulating and verifying a decentralized mission control algorithm for a fleet of collaborative UAVs." Procedia Computer Science 9 (2012): 792-801.

[5] Eco-Dolphin Project Introduction: http://www.eco-dolphin.org/about-us/our-project/

[6] Heimerdinger, Walter L., and Charles B. Weinstock. *A conceptual framework for system fault tolerance*. No. CMU/SEI-92-TR-33. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1992.

[7] Middleware. https://en.wikipedia.org/wiki/Middleware

[8] Randell, Brian. "System structure for software fault tolerance." IEEE Transactions on Software Engineering 2 (1975): 220-232.

[9] Lee, Peter A., and Thomas Anderson. Fault tolerance: principles and practice. Vol. 3. Springer Science & Business Media, 2012. https://books.google.com/books?hl=zh-CN&lr=&id=RsqqCAAAQBAJ&oi=fnd&pg=PA1&dq=fault+tolerance&ots=J5a7PXSYd8&sig=QznritFgQP7NjR83TjI5JBP1y_M#v=onepage&q=fault%20tolerance&f=false

[10] Moos-Ivp http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php

[11] Benjamin, Michael R., et al. "Nested autonomy for unmanned marine vehicles with MOOS-IvP." Journal of Field Robotics 27.6 (2010): 834-875.

[12] Benjamin, Michael R., et al. "A tour of moos-ivp autonomy software modules." (2009).

[13] Benjamin, Michael R., et al. "An overview of moos-ivp and a brief users guide to the ivp helm autonomy software." (2009).

[14] Benjamin, Michael R., et al. "Autonomous control of an autonomous underwater vehicle towing a vector sensor array." Robotics and Automation, 2007 IEEE International Conference on. IEEE, 2007.

[15] Eickstedt, Donald P., Michael R. Benjamin, and J. Curcio. "Behavior based adaptive control for autonomous oceanographic sampling." Robotics and Automation, 2007 IEEE International Conference on. IEEE, 2007.

[16] Wolf, Michael T., et al. "360-degree visual detection and target tracking on an autonomous surface vehicle." Journal of Field Robotics 27.6 (2010): 819-833.

[17] Benjamin, Michael R., et al. "Extending a MOOS-IvP autonomy system and users guide to the IvPBuild toolbox." (2009).

[18] Shell Script. https://en.wikipedia.org/wiki/Shell_script

[19] Eickstedt, Donald P., and Michael R. Benjamin. "Cooperative target tracking in a distributed autonomous sensor network." OCEANS 2006. IEEE, 2006.

[20] UPPAAL. http://www.uppaal.org/

[21] Hessel, Anders, et al. "Testing real-time systems using UPPAAL." Formal methods and testing. Springer Berlin Heidelberg, 2008. 77-117.

[22] Larsen, Kim G., Paul Pettersson, and Wang Yi. "UPPAAL in a nutshell." International Journal on Software Tools for Technology Transfer (STTT) 1.1 (1997): 134-152.

[23] Larsen, Kim G., et al. "Testing real-time embedded software using UPPAAL-TRON: an industrial case study." Proceedings of the 5th ACM international conference on Embedded software. ACM, 2005.

[24] David, Alexandre, et al. "Model-based framework for schedulability analysis using UPPAAL 4.1." Model-based design for embedded systems 1.1 (2009): 93-119.

[25] Eickstedt, Donald P., and Scott R. Sideleau. "The backseat control architecture for autonomous robotic vehicles: A case study with the Iver2 AUV." Marine technology society journal 44.4 (2010): 42-54.