Project Report on

# Design of 3D mobile underwater network

Prepared for and under the guidance of

## Dr. Sarang C. Dhongri

(Department of Electrical & Electronics Engineering)


**BITS PILANI, K. K. BIRLA GOA CAMPUS**

***Submitted in partial fulfillment of the course:***

**ECE F376: Design Oriented Project**

**Prepared By:**

Shubham Ajay Agrawal 2017AAPS0363G

# Table of Content

# 1. UnetStack3.0

## 1.1. Introduction

UnetStack3.0 is a new technology introduced to replace UnetSim which was an Underwater Network Simulator. Unetsim was a technology developed as part of the Unet project, to build communication networks that extend underwater.

Quick recap of features that **Unetsim** has offered for simulation of underwater networks:

- **Datagram Service**: This service defines messages and capabilities for transfer of packets of data over the network.
- **Physical Service**: The physical service is typically provided by physical layer agents such as modem drivers and simulated modems.
- **Ranging Service**: Agents offering the ranging service provide time synchronization and ranging functionalities between pairs of nodes.
- **Link Service**: Agents offering the link service provide single-hop communication. Single-hop here refers to a logical single hop in the UnetStack network.
- **Medium Access Control Service**: Medium access control (MAC) service provide request, waiting and cancelling of the corresponding Messages.
- **Routing and Route Maintenance Services**: Provide multi-hop communication for datagram messages. These agents accept datagram messages and route them to their destination based on supported underlying routing algorithms
- **Transport Service**: Provide end-to-end reliability and fragmentation/reassembly for large datagrams.
- **Remote Access Service**: Provide control over remote nodes. This includes querying/setting parameters, delivering text messages, transferring files and running scripts remotely
- **Node information Service**: Manages and maintains a node's attributes such as address, location, speed etc.
- **Address Resolution Service**: Defines the messages required for address allocation and name-to-address resolution.
- **Baseband Service**: Enables researchers to access low-level signal transmission and reception capability of a modem.
- **Network Simulation**: Simulates an underwater network on a single computer (or a cluster of computers) in real-time, or as a discrete-event simulation. Simulation codes are written in GROOVY DLS. Uses all above Service to make simulation more effective and readable to user.
- **Log Generation**: Provides every single process performed under simulation to the user with details for better understanding of code and the simulation Process.

## 1.2. What's new

Extending to the vast features already provided by Unetsim, changes has been made to make the IDE more user friendly and some extra features to make the algorithm to be more user readable and help in visualizing the nodes in real time.

Some new feature added to UnetStack3.0 are:

1. **Command Shell**
   Earlier in Unetsim it used the application of Unet which allowed to create node using code and work on those node only by coding to work on the corresponding node which had to be predefined.
   Command Shell which is recently introduced in UnetStack3.0 is the simplest way to interact with the created node on the console via web browser in real-time. This is similar to working on a modem in real-time. It can set up n-node network and access the command shell for each of the nodes. The default UnetStack shell accepts any Groovy code.
2. **Map**
   One of the very impressive feature that has been added to UnetStack3.0 is the Map. It create a 2D view of the created node based on the coordinates provided. It can also use real location using the corresponding longitude and latitude.
   It also work in real time showing the trajectory of a node and also give the trace of node that has been followed.
3. **UnetAudio**
   If you have a UnetStack-compatible acoustic modem that supports BASEBAND service, you can use them to transit and record arbitrary acoustic signals. Even without access to modems you can try this out using the Unet Audio SDOAM that stands for software defined open architecture modem, which is fully functional modem that uses your computer soundcard for transmission and reception. To run UnetAudio it require PortAudio to be installed and then it can be worked on using Command Shell similar to the working with nodes.

# 2. Circular Mobility

## 2.1. Theory

Most common simulation that can help in understanding different and variety of Communication feature is moving around few networks, here nodes, to study the underwater communication.

Here we are using 4 static nodes and 1 mobile node that tries to communicate with other static nodes while doing a circular trajectory around the 4 static nodes. Provided a particular power limit to the AUV for sending message and receiving the corresponding response. This will ensure only one node is able to communicate at a time for the simplicity of the simulation. When a static node is in the range of communication both nodes will exchange messages to ensure secure connection.

Given a fixed location to each static node and a starting point of the moving node we easily calculated the relation between the speed and the turning rate of the AUV.

## 2.2. Simulation

**Code Snippet:**

```
node 'C', address: 31, location: [80.m, 80.m, -50.m]

node 'D', address: 32, location: [80.m, -80.m, -50.m]

node 'E', address: 33, location: [520.m, 80.m, -50.m]

node 'F', address: 34, location: [520.m, -80.m, -50.m]
```

Initializes the static nodes, gives them a fixed address and a static location.

```
Def AUV= node 'B', address: 21, location: [0, 0, -25.m], mobility: true

AUV.motionModel = [ speed: 47.mps, turnRate: 9.dps]
```

Initializes the mobile node with a fixed starting location and given a fixed speed and turnRate that are can easily be acquired from some basic calculations.

```
phy[1].powerLevel = -35.dB;
```

Given a particular power level to the node for sending messages, ensuring that only one node receives the signal at a time for the simplicity of the task.

```
phy << new DatagramReq(to: {AUV_Addr} ,data : {DATA} , protocol:cluster_protocol)

println "data sent by {Sender_Addr} to {AUV_addr}"
```

Each Static node tries to send a fixed message to AUV which if comes under the radius of power limit acknowledge the message, hence the static node print the following message “data sent by 31 to 21”. Where 31 is the address of sender and 21 is the address of the AUV.

## 2.3. Result

On simulation the AUV or the mobile node starts rotating on fixed path around the statics node.

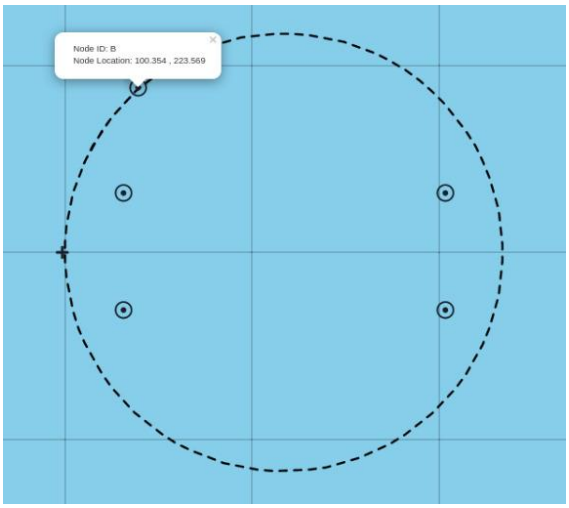


*Figure 1. AUV following the Circular Trajectory around the Static Nodes.*

For more clear understanding we generate AUV location every 10 seconds to check the location where the AUV and one of the static node exchanges messages.

Here are the 4 instances when AUV and each of the 4 static node communicated:

1. Two nearest location of the AUV when it communicated with Node C with addr 31

   Current location of 21 [16.208077391785697, 97.14204967240575, -25.0]

   data sent by 31 to 21

   Current location of 21 [34.888681166406286, 140.21752648328163, -25.0]

2. Two nearest location of the AUV when it communicated with Node F with addr 34

   Current location of 21 [568.0145879376737, 131.42369080147336, -25.0]

   data sent by 34 to 21

   Current location of 21 [585.2643680877947, 87.75554743996591, -25.0]

3. Two nearest location of the AUV when it communicated with Node E with addr 33

   Current location of 21 [585.2643680877947, 87.75554743996591, -25.0]

   data sent by 33 to 21

   Current location of 21 [582.2145086337408, -97.14204967240575, -25.0]

4. Two nearest location of the AUV when it communicated with Node D with addr 32

   Current location of 21 [30.40799808785286, -131.42369080147355, -25.0]

   data sent by 32 to 21

   Current location of 21 [13.158217937731887, -87.75554743996611, -25.0]

# 3. Dynamic Trajectory

## 3.1. Theory

Dynamic Trajectory means defining the trajectory of any node dynamically based on the information provided corresponding to the current location. It uses the current location of the node to do the according calculation for turning and dive rate to reach the newly provided location provided to the node by nearby node on any undefined time.

Data has been provided as a form of an array with predefined meaning of each element.

We used an array of length 7 of Data Type: INT

1st element represent the Address of next Node

2nd and 3rd combine to form x coordinate of the new location.

4nd and 5rd combine to form y coordinate of the new location.

6nd and 7rd combine to form z coordinate of the new location.

For example let any given time Message received by the Mobile node is an array of following type

INT [A, B, C, D, E, F, G]

INT represent the data type of each element of the array which is always fixed and predefined for our example.

A will give the address of the next node

**For location we restrict each element to be restricted to 0 to (n-1) limit.**

X coordinate will then be represented as B*n+ C (meters)

Y coordinate will then be represented as D*n+ E (meters)

Z coordinate will then be represented as F*n+ G (meters)

(In our example we restricted n to 100 for convenience and recommend **n = $10^x$** Format, x be any positive Integer**)**

Parameters involved in Mobility

- speed — speed in m/s, if mobile node
- heading — heading in degrees, 0 is North, measured clockwise, if mobile node
- turnRate — turn rate in degrees/s, measured clockwise, if mobile node
- diveRate — dive rate in m/s, if mobile node

For Defining a trajectory we only need **heading** and **diveRate**.

**Speed** and **turnRate** is **not important** because speed of a node is already defined and we keep it fixed. While turnRate is also not necessary because we opt to make the change in direction instantaneously.

We calculated the heading of the node using current location and the location provided. For dive rate we calculated to keep it a constant minimal value because we want the node to follow a straight path of heading and diving simultaneously. Clear understanding can be done in Figure 2.

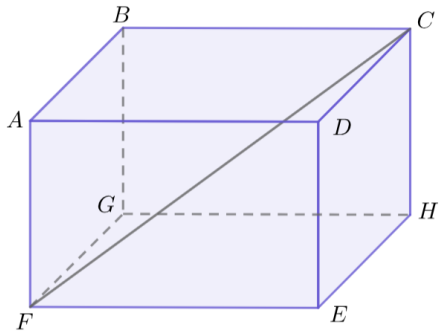


*Figure 2. Illustration of 3D least distance Path*

Instead of First going C to H (doing a dive only) then H to F (by setting heading)

Here we prefer to save moving distance by directly doing from C to F by combing the two.

**Calculations:**

Heading = $\tan^{-1}\left(\frac{y2-y1}{x2-x1}\right)$

DiveRate = $\frac{(z1-z2)}{\sqrt{((x2-x1)^2+(y2-y1)^2)}*speed}$

(x1, y1, z1) is the current location of the node.

(x2, y2, z2) is the location to be reached, provided in the array.

## 3.2. Simulation

**Code Snippet:**

```
def auv = node('AUV', address: 100, location: [-1.km, 0.km, -1.km], mobility: true}
```

This Defines an AUV node with an Address 100 and location of (-1000, 0, -1000).meters with mobility

```
def n = node('Node', address: 6, location: [0, 0, -1.km], mobility: true}
```

This defines a mobile node with Address 6 and location of (0, 0, -1000).meters.

```
auv.motionModel = [[speed: 5.mps, heading: 270.deg]]
n.motionModel = [[speed: 10.mps, heading: 90.deg ]]
```

Then given AUV a speed of 5 m/s and a heading in direction of 270° from current location and mobile node a speed of 10 m/s and a heading in direction of 90°. This define a trajectory of both mobile node to move towards each other.

```
phy << new DatagramReq(to: 100, data: [2, 19, 00, 23, 00, 00, 00], protocol:PING_PROTOCOL)
```

This code sends an array of [2, 19, 00, 23, 00, 00, 00] to node with address 100 i.e. AUV

```
ArrayList<Integer> collectB=new ArrayList<Integer>()
collectB.addAll(msg.data)
int d =msg.data[0]
int x2=collectB[1]*100+collectB[2]
int y2=collectB[3]*100+collectB[4]
int z2=collectB[5]*100+collectB[6]
int x1=node.location[0]
int y1=node.location[1]
int z1=node.location[2]
double heading= Math.toDegrees(Math.atan((y2-y1)/(x2-x1)))
double diveRate=(z1-z2)/(Math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1))/node.speed)
```

This creates an arraylist and stores the incoming message for further processing.

Stores the address of new node in a variable d and coordinates in (x2, y2, z2)

Then retrieve the current location of mobile node in (x1, y1, z1)

Calculate and stores the heading and diveRate in corresponding variables.

## 3.3. Results

On simulation both the AUV and the mobile node start moving towards each other.

- AUV keeps sending a message to mobile node trying to communicate and acquire the message that will be used to decode the location of next location. This instance shows the location of both the node when there are far apart and unable to exchange message.

  Sending data to: 6

  n [-276.095, -3.381188580105885E-14, -1000.0]

  auv [-448.95000000000005, 0.0, -1000.0]

  Sending data to: 6

  n [-301.095, -3.6873502798927236E-14, -1000.0]

  auv [-398.95000000000005, 0.0, -1000.0]

- Once the node come in the range so that they can exchange messages, AUV accept the array that will be used to decode the next location for AUV. Following log show the exchange of message between the nodes.

  Sending data to : 6

  n [-326.09499999999997, -3.993511979679561E-14, -1000.0]

  auv [-358.9420373833367, 23.965518851061, -989.5771227642459]

  Data received as from 100: If near send data

  sent msg to 100 with data: [2, 19, 00, 23, 00, 00, 00]

  Received data from 6: [2, 19, 0, 23, 0, 0, 0]

  Next node address: 2

  Nodes 2 coordinates: (1900.m, 2300.m)

  changed diveRate to: -3.08416460983117 mps

  changed heading to: 44.77142762431264 deg

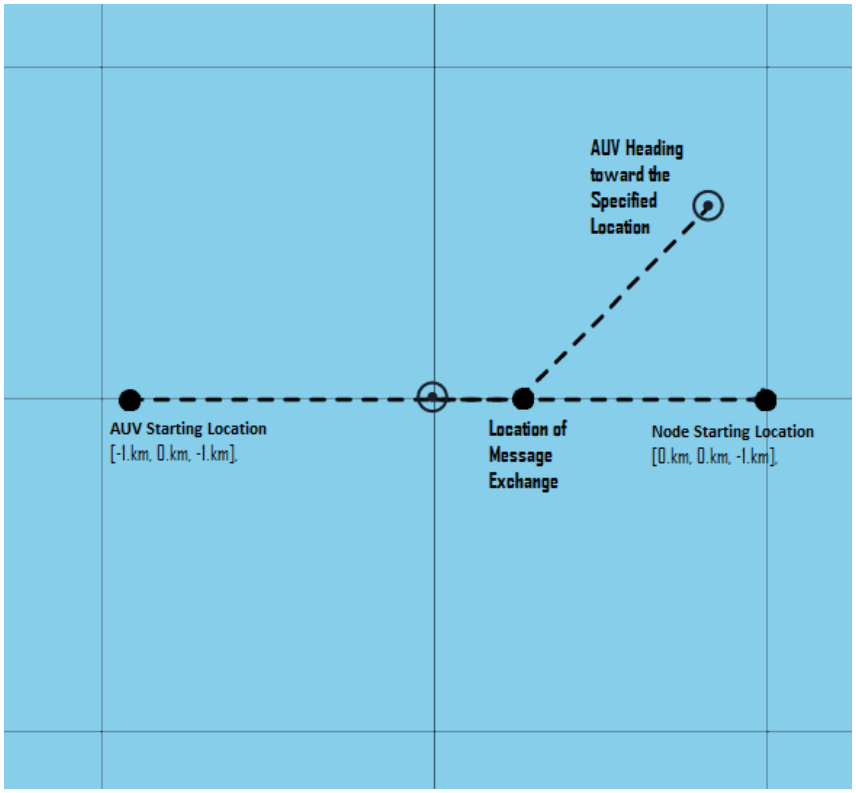


*Figure 3. Dynamic Trajectory of AUV*

Figure x show how AUV and Node started from the specified location. Once the reach in range they exchanged messages. AUV then calculated the heading and the diveRate and start moving in that direction, while the node keeps following its predefined trajectory. Following log shows the location of node and AUV at some later time.

Sending data to : 6

n [-476.1, -5.830543410740549E-14, -1000.0]

auv [-147.64119338164443, 236.99592864855973, -897.0308506655031]

Sending data to : 6

n [-501.09499999999997, -6.136643878187429E-14, -1000.0]

auv [-112.45994893237096, 272.4681944205747, -881.6228883678178]

# 4. Plume Detection

Our objective is to mark the boundary of the plume using a heterogeneous deployment of sparsely deployed static nodes and AUVs. Here we have sensors on the nodes can sense the concentration of the plume. The entire mechanism of the AUVs is dynamic and no part of the trajectory is predefined.

A network of static node has been deployed which continuously senses the concentration of the plume.

## 4.1. Static Nodes

- Once the concentration of the plume crosses a certain threshold value T, the static node has to communicate its location to the base station through hop by hop communication.
- A mechanism of packet history buffer is added so that the same data is not sent again and again
- The Data is sent to the static node above it, which forwards the data to the static node above it and so on till it reaches the base station at sea level.

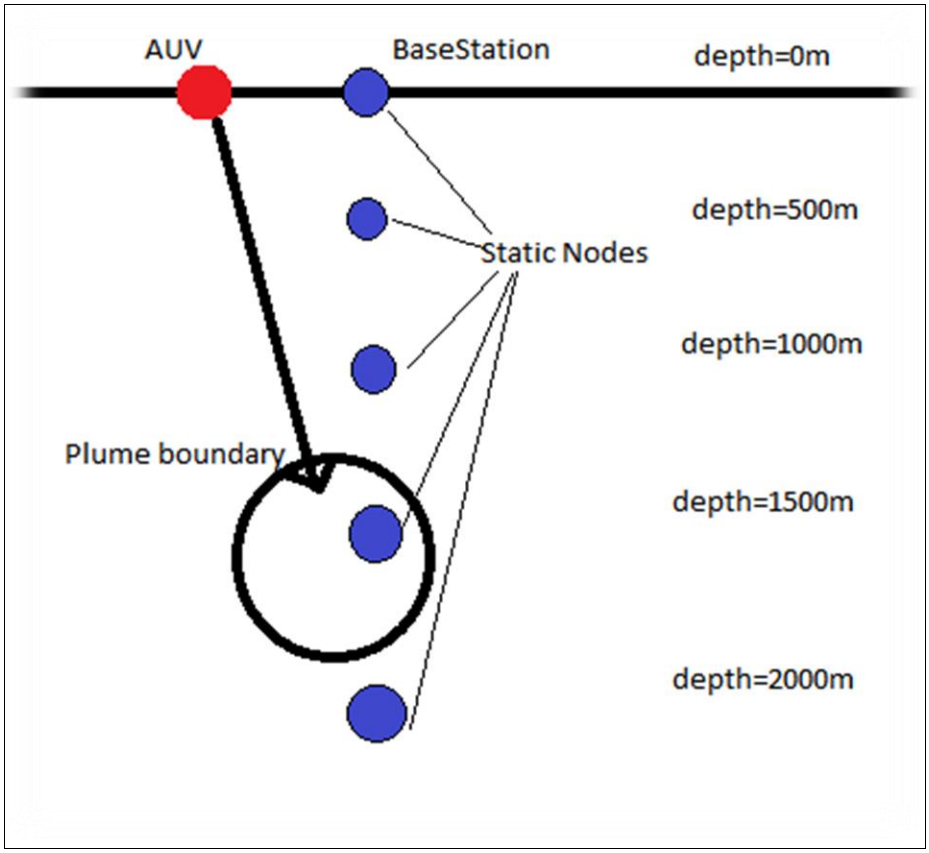


*Figure 4 Static Nodes and AUV detecting Plume*

## 4.2. Working of AUV

1. Once the base station receives the data, it deploys the AUVs and forward the data to the AUVs.

2. The AUVs calculate its dive rate and heading dynamically to move in the direction of the static node which had detected the plume.

3. The AUV keeps on sensing the concentration every 100ms. Once it crosses the threshold value of the plume, the AUV stops.

4. Now it will only move in the X-Y plane to mark the boundary of the plume at that depth.

5. It senses the area around it, and moves in the direction where concentration is closest to T.

6. Once it reaches a point where concentration = T, it can move in 2 directions with concentration = T too. Conditions are mentioned so that the AUV does not go back and forth between 2 points infinitely.

7. The AUV record the coordinates of the boundary.

8. The AUV eventually completes the boundary and reaches back to its initial point.

9. AUV stops when reaches back to its initial point.

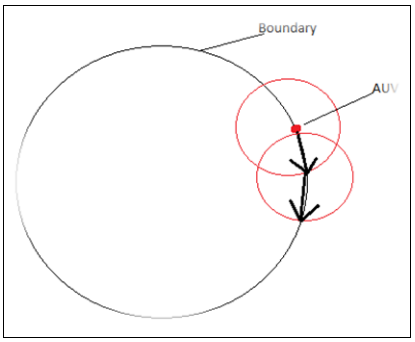


*Figure 5. Movement of AUV*

10. Move the AUV to the location with maximum temp by continuously changing its heading in the direction of maximum concentration around it.

11. Move the AUV in the Z plane 30m downwards and stop.

12. If concentration at center is less than T go to step 15.

13. Now on the plume can only move in the X, Y plane. Move the AUV in heading 0 degrees will it reaches a point with concentration=T.

14. Repeat step 6

15. Go back to station

# 5. Bibliography

List of Reference Book(s), Journal(s), Other Authentic Document(s)/Source(s):


- Deng, Yueyue et al. "Task Allocation And Path Planning For Collaborative Autonomous Underwater Vehicles Operating Through An Underwater Acoustic Network". *Journal Of Robotics*, vol 2013, 2013, pp. 1-15. *Hindawi Limited*, doi:10.1155/2013/483095.
- Petillo, Stephanie et al. "Constructing A Distributed AUV Network For Underwater Plume-Tracking Operations". *International Journal Of Distributed Sensor Networks*, vol 8, no. 1, 2012, p. 191235. *SAGE Publications*, doi:10.1155/2012/191235
- Petillo, Stephanie M., and Henrik Schmidt. "Autonomous And Adaptive Underwater Plume Detection And Tracking With Auvs: Concepts, Methods, And Available Technology". *IFAC Proceedings Volumes*, vol 45, no. 27, 2012, pp. 232-237. *Elsevier BV*, doi:10.3182/20120919-3-it-2046.00040.
- Zhuo, Xiaoxiao et al. "AUV-Aided Energy-Efficient Data Collection In Underwater Acoustic Sensor Networks". *IEEE Internet Of Things Journal*, 2020, pp. 1-1. *Institute Of Electrical And Electronics Engineers (IEEE)*, doi:10.1109/jiot.2020.2988697.
- Unet (www.unetstack.net)
- fjåge 1.6.2 documentation(https://fjage.readthedocs.io/)
- http://docs.groovy-lang.org/
- https://ns2blogger.blogspot.com/p/the-file-written-by-application-or-by.html
- https://fjage.readthedocs.io/en/latest/behaviors.html