Autonomous Underwater Vehicle: A Surveillance Protocol

S.V. Rajendran

February 1, 2025

Preface

This project is an attempt at combining all of my major skills into creating a truly sophisticated project real world project. The aim of this project is to come up with a perception and control pipeline for AUVs for maritime surveillance. As such, the work involves creating a number of sub-pipelines.

The first is the signal simulation and geometry pipeline. This pipeline takes care of creating the underwater profile and the signal simulation that is involved for the perception stack.

The perception stack for the AUV is one front-looking-SONAR and two side-scan SONARs. The parameters used for this project are obtaine from that of NOAA ships that are publically available. No proprietary parameters or specifications have been included as part of this project. The three SONARs help the AUV perceive the environment around it. The goal of the AUV is to essentially map the sea-floor and flag any new alien bodies in the "water"-space.

The control stack essentially assists in controlling the AUV in achieving the goal by controlling the AUV to spend minimal energy in achieving the goal of mapping. The terrains are randomly generated and thus, intelligent control is important to perceive the surrounding environment from the acoustic-images and control the AUV accordingly. The AUV is currently granted six degrees of freedom. The policy will be trained using a reinforcement learning approach (DQN is the plan). The aim is to learn a policy that will successfully learn how to achieve the goals of the AUV while also learning and adapting to the different kinds of terrains the first pipeline creates. To that end, this will be an online algorithm since the simulation cannot truly cover real terrains.

The project is currently written in C++. Despite the presence of significant deep learning aspects of the project, we choose C++ due to the real-time nature of the project and this is not merely a prototype. In addition, to enable the learning aspect, we use LibTorch (the C++ API to PyTorch).

Introduction

Contents

Chapter 1

Setup

1.1 Overview

- Clone the AUV repository: `https://github.com/vrsreeganesh/AUV.git`.

- This can be performed by entering the terminal, "cd"-ing to the directory you wish and then typing: `git clone https://github.com/vrsreeganesh/AUV.git` and press enter.

- Note that in case it has not been setup, ensure github setup in the terminal. If not familiar with the whole git work-routine, I suggest sticking to Github Desktop. Its a lot easier and the best to get started right away.

Chapter 2

Underwater Environment Setup

Overview

- The underwater environment is modelled using discrete scatterers.
- They contain two attributes: coordinates and reflectivity.

2.1 Seafloor Setup

- The sea-floor is the first set of scatterers we introduce.
- A simple flat or flat-ish mesh of scatterers.
- Further structures are simulated on top of this.
- The seafloor setup script is written in section 8.2.1;

2.2 Additional Structures

- We create additional scatters on the second layer.
- For now, we stick to simple spheres, boxes and so on;

Chapter 3

Hardware Setup

Overview

Chapter 4

Geometry

Overview

4.1   Ray Tracing

- There are multiple ways for ray-tracing.

- The method implemented during the FBLS and SS SONARs weren't super efficient as it involved pair-wise dot-products. Which becomes an issue when the number of points are increased, which is the case when the range is super high or the beamwidth is super high.

4.1.1   Pairwise Dot-Product

- In this method, given the coordinates of all points that are currently in the illumination cone, we find the cosines between every possible pairs of points.

- This is where the computational complexity arises as the number of dot products increase exponentially with increasing number of points.

- This method is a liability when it comes to situations where the range is super high or when the angle-beamwidth is non-narrow.

4.1.2   Range Histogram Method

- Given the angular beamwidths: azimuthal beamwidth and elevation beamwidth, we quantize square cone into a number of different values (note that the square cone is not an issue as the step before ensures conical subsetting).

- We split the points into different "range-cells".

- For each range-cell, we make a 2D histogram of azimuths and elevations. Then within each range-cell and for each azimuth-elevation pair, we find the closest point and add it to the check-box.

- In the next range-cell, we only work with those azimuth-elevation pairs whose checkbox has not been filled. Since, for the filled ones, the filled scatter will shadow the othersin the following range cells.

Algorithm 1 Range Histogram Method

ScatterCoordinates ←
ScatterReflectivity ←
AngleDensity ← Quantization of angles per degree.
AzimuthalBeamwidth ← Azimuthal Beamwidth
RangeCellWidth ← The range-cell width

Chapter 5

Signal Simulation

Overview

- Define LFM.
- Define shadowing.
- Simulate Signals (basic)
- Simulate Signals with additional effects (doppler)

5.1 Transmitted Signal

- We use a linear frequency modulated signal.
- The signal is defined in setup-script of the transmitter. Please refer to section: 8.1.2;

5.2 Signal Simulation

1. First we obtain the set of scatterers that reflect the transmitted signal.
2. The distance between all the sensors and the scatterer distances are calculated.
3. The time of flight from the transmitter to each scatterer and each sensor is calculated.
4. This time is then calculated into sample number by multiplying with the sampling-frequency of the uniform linear arrays.
5. We then build a signal matrix that has the dimensions corresponding to the number of samples that are recorded and the number of sensors that are present in the sensor-array.
6. We place impulses in the points corresponding to when the signals arrives from the scatterers. The result is a matrix that has x-dimension as the number of samples and the y-dimension as the number of sensors.

7. Each column is then convolved (linearly convolved) with the transmitted signal. The resulting matrix gives us the signal received by each sensor. Note that this method doesn't consider doppler effects. This will be added later.

Chapter 6

Imaging

Overview

- Present different imaging methods.

Decimation

1. The signals received by the sensors have a huge number of samples in it. Storing that kind of information, especially when it will be accumulated over a long time like in the case of synthetic aperture SONAR, is impractical.

2. Since the transmitted signal is LFM and non-baseband, this means that making the signal a complex baseband and decimating it will result in smaller data but same information.

3. So what we do is once we receive the signal at a stop-hop, we baseband the signal, low-pass filter it around the bandwidth and then decimate the signal. This reduces the sample number by a lot.

4. Since we're working with spotlight-SAS, this can be further reduced by beamforming the received signals in the direction of the patch and just storing the single beam. (This needs validation from Hareesh sir btw)

Match-Filtering

- A match-filter is any signal, that which when multiplied with another signal produces a signal that has a flag frequency-response = an impulse basically. (I might've butchered that definition but this will be updated later)

- This is created by time-reversing and calculating the complex conjugate of the signal.

- The resulting match-filter is then convolved with the received signal. This will result in a sincs being placed where impulse responses would've been if we used an infinite bandwidth signal.

Questions

- Do we match-filter before beamforming or after. I do realize that theoretically they're the same but practically, does one conserve resolution more than the other.

Chapter 7

Results

Chapter 8

Software

Overview

-

8.1 Class Definitions

8.1.1 Class: Scatter

The following is the class definition used to encapsulate attributes and methods of the scatterers.

```cpp
// header-files
#include <iostream>
#include <ostream>
#include <torch/torch.h>

#pragma once

// hash defines
#ifndef PRINTSPACE
#define PRINTSPACE    std::cout<<"\n\n\n\n\n\n\n"<<std::endl;
#endif
#ifndef PRINTSMALLLINE
#define PRINTSMALLLINE std::cout<<"---------------------------------------------"<<std::endl;
#endif
#ifndef PRINTLINE
#define PRINTLINE     std::cout<<"========================================="<<std::endl;
#endif
#ifndef DEVICE
    #define DEVICE        torch::kMPS
    // #define DEVICE        torch::kCPU
#endif


#define PI            3.14159265


// function to print tensor size
void print_tensor_size(const torch::Tensor& inputTensor) {
    // Printing size
    std::cout << "[";
```

```
31      for (const auto& size : inputTensor.sizes()) {
32          std::cout << size << ",";
33      }
34      std::cout << "\b]" <<std::endl;
35  }
36
37  // Scatterer Class = Scatterer Class
38  // Scatterer Class = Scatterer Class
39  // Scatterer Class = Scatterer Class
40  // Scatterer Class = Scatterer Class
41  // Scatterer Class = Scatterer Class
42  class ScattererClass{
43  public:
44
45      // public variables
46      torch::Tensor coordinates; // tensor holding coordinates [3, x]
47      torch::Tensor reflectivity; // tensor holding reflectivity [1, x]
48
49      // constructor = constructor
50      ScattererClass(torch::Tensor arg_coordinates = torch::zeros({3,1}),
51                     torch::Tensor arg_reflectivity = torch::zeros({3,1})):
52                     coordinates(arg_coordinates),
53                     reflectivity(arg_reflectivity) {}
54
55      // overloading output
56      friend std::ostream& operator<<(std::ostream& os, ScattererClass& scatterer){
57
58          // printing coordinate shape
59          os<<"\t> scatterer.coordinates.shape = ";
60          print_tensor_size(scatterer.coordinates);
61
62          // printing reflectivity shape
63          os<<"\t> scatterer.reflectivity.shape = ";
64          print_tensor_size(scatterer.reflectivity);
65
66          PRINTSMALLLINE
67
68          // returning os
69          return os;
70      }
71
72  };
```

8.1.2 Class: Transmitter

The following is the class definition used to encapsulate attributes and methods of the projectors used.

```cpp
// header-files
#include <iostream>
#include <ostream>
#include <cmath>

// Including classes
#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"

// Including functions
#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fCart2Sph.cpp"
#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"
#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fSph2Cart.cpp"

#pragma once

// hash defines
#ifndef PRINTSPACE
#   define PRINTSPACE   std::cout<<"\n\n\n\n\n\n\n"<<std::endl;
#endif
#ifndef PRINTSMALLLINE
#   define PRINTSMALLLINE std::cout<<"----------------------------------------------"<<std::endl;
#endif
#ifndef PRINTLINE
#   define PRINTLINE    std::cout<<"=============================================="<<std::endl;
#endif

#define PI          3.14159265
#define DEBUGMODE_TRANSMITTER   false

#ifndef DEVICE
    #define DEVICE      torch::kMPS
    // #define DEVICE     torch::kCPU
#endif


class TransmitterClass{
public:

    // physical/intrinsic properties
    torch::Tensor location;          // location tensor
    torch::Tensor pointing_direction; // pointing direction

    // basic parameters
    torch::Tensor Signal;    // transmitted signal (LFM)
    float azimuthal_angle;   // transmitter's azimuthal pointing direction
    float elevation_angle;   // transmitter's elevation pointing direction
    float azimuthal_beamwidth; // azimuthal beamwidth of transmitter
    float elevation_beamwidth; // elevation beamwidth of transmitter
    float range;             // a parameter used for spotlight mode.

    // transmitted signal attributes
    float f_low;             // lowest frequency of LFM
    float f_high;            // highest frequency of LFM
    float fc;                // center frequency of LFM
    float bandwidth;         // bandwidth of LFM

    // shadowing properties
    int azimuthQuantDensity;        // quantization of angles along the azimuth
    int elevationQuantDensity;      // quantization of angles along the elevation
    float rangeQuantSize;           // range-cell size when shadowing
    float azimuthShadowThreshold;   // azimuth thresholding
    float elevationShadowThreshold; // elevation thresholding

    // // shadowing related
    // torch::Tensor checkbox;          // box indicating whether a scatter for a range-angle pair has been
        found
```

```
66      // torch::Tensor finalScatterBox;  // a 3D tensor where the third dimension represnets the vector length
67      // torch::Tensor finalReflectivityBox; // to store the reflectivity
68
69
70
71      // Constructor
72      TransmitterClass(torch::Tensor location = torch::zeros({3,1}),
73                       torch::Tensor Signal     = torch::zeros({10,1}),
74                       float azimuthal_angle   = 0,
75                       float elevation_angle   = -30,
76                       float azimuthal_beamwidth = 30,
77                       float elevation_beamwidth = 30):
78                       location(location),
79                       Signal(Signal),
80                       azimuthal_angle(azimuthal_angle),
81                       elevation_angle(elevation_angle),
82                       azimuthal_beamwidth(azimuthal_beamwidth),
83                       elevation_beamwidth(elevation_beamwidth) {}
84
85      // overloading output
86      friend std::ostream& operator<<(std::ostream& os, TransmitterClass& transmitter){
87          os<<"\t> azimuth         : "<<transmitter.azimuthal_angle <<std::endl;
88          os<<"\t> elevation       : "<<transmitter.elevation_angle <<std::endl;
89          os<<"\t> azimuthal beamwidth: "<<transmitter.azimuthal_beamwidth<<std::endl;
90          os<<"\t> elevation beamwidth: "<<transmitter.elevation_beamwidth<<std::endl;
91          PRINTSMALLLINE
92          return os;
93      }
94
95      // overloading copyign operator
96      TransmitterClass& operator=(const TransmitterClass& other){
97
98          // checking self-assignment
99          if(this==&other){
100             return *this;
101         }
102
103         // allocating memory
104         this->location          = other.location;
105         this->Signal            = other.Signal;
106         this->azimuthal_angle   = other.azimuthal_angle;
107         this->elevation_angle   = other.elevation_angle;
108         this->azimuthal_beamwidth = other.azimuthal_beamwidth;
109         this->elevation_beamwidth = other.elevation_beamwidth;
110         this->range             = other.range;
111
112         // transmitted signal attributes
113         this->f_low             = other.f_low;
114         this->f_high            = other.f_high;
115         this->fc                = other.fc;
116         this->bandwidth         = other.bandwidth;
117
118         // shadowing properties
119         this->azimuthQuantDensity   = other.azimuthQuantDensity;
120         this->elevationQuantDensity = other.elevationQuantDensity;
121         this->rangeQuantSize        = other.rangeQuantSize;
122         this->azimuthShadowThreshold = other.azimuthShadowThreshold;
123         this->elevationShadowThreshold = other.elevationShadowThreshold;
124
125         // this->checkbox              = other.checkbox;
126         // this->finalScatterBox       = other.finalScatterBox;
127         // this->finalReflectivityBox  = other.finalReflectivityBox;
128
129         // returning
130         return *this;
131
132     };
133
134     /*=======================================================================
135     Aim: Update pointing angle
136     -----------------------------------------------------------------------
137     Note:
138         > This function updates pointing angle based on AUV's pointing angle
```

```
139        > for now, we're assuming no roll;
140    -----------------------------------------------------------------------*/
141    void updatePointingAngle(torch::Tensor AUV_pointing_vector){
142
143        // calculate yaw and pitch
144        if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 140 \n";
145        torch::Tensor AUV_pointing_vector_spherical = fCart2Sph(AUV_pointing_vector);
146        torch::Tensor yaw                      = AUV_pointing_vector_spherical[0];
147        torch::Tensor pitch                    = AUV_pointing_vector_spherical[1];
148        if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 144 \n";
149
150        // std::cout<<"\t TransmitterClass: AUV_pointing_vector = "<<torch::transpose(AUV_pointing_vector, 0,
151            1)<<std::endl;
151        // std::cout<<"\t TransmitterClass: AUV_pointing_vector_spherical =
                "<<torch::transpose(AUV_pointing_vector_spherical, 0, 1)<<std::endl;
152
153        // calculating azimuth and elevation of transmitter object
154        torch::Tensor absolute_azimuth_of_transmitter = yaw +
                torch::tensor({this->azimuthal_angle}).to(torch::kFloat).to(DEVICE);
155        torch::Tensor absolute_elevation_of_transmitter = pitch +
                torch::tensor({this->elevation_angle}).to(torch::kFloat).to(DEVICE);
156        if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 149 \n";
157
158        // std::cout<<"\t TransmitterClass: this->azimuthal_angle = "<<this->azimuthal_angle<<std::endl;
159        // std::cout<<"\t TransmitterClass: this->elevation_angle = "<<this->elevation_angle<<std::endl;
160        // std::cout<<"\t TransmitterClass: absolute_azimuth_of_transmitter =
                "<<absolute_azimuth_of_transmitter<<std::endl;
161        // std::cout<<"\t TransmitterClass: absolute_elevation_of_transmitter =
                "<<absolute_elevation_of_transmitter<<std::endl;
162
163        // converting back to Cartesian
164        torch::Tensor pointing_direction_spherical = torch::zeros({3,1}).to(torch::kFloat).to(DEVICE);
165        pointing_direction_spherical[0]        = absolute_azimuth_of_transmitter;
166        pointing_direction_spherical[1]        = absolute_elevation_of_transmitter;
167        pointing_direction_spherical[2]        = torch::tensor({1}).to(torch::kFloat).to(DEVICE);
168        if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 60 \n";
169
170        this->pointing_direction = fSph2Cart(pointing_direction_spherical);
171        if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 169 \n";
172
173    }
174
175    /*=======================================================================
176    Aim: Subsetting Scatterers inside the cone
177    ........................................................................
178    steps:
179        1. Find azimuth and range of all points.
180        2. Fint azimuth and range of current pointing vector.
181        3. Subtract azimuth and range of points from that of azimuth and range of current pointing vector
182        4. Use tilted ellipse equation to find points in the ellipse
183    -----------------------------------------------------------------------*/
184    void subsetScatterers(ScattererClass* scatterers,
185                      float tilt_angle){
186
187        // translationally change origin
188        scatterers->coordinates = scatterers->coordinates - this->location; if(DEBUGMODE_TRANSMITTER)
                std::cout<<"\t\t TransmitterClass: line 188 "<<std::endl;
189
190        // Finding spherical coordinates of scatterers and pointing direction
191        torch::Tensor scatterers_spherical    = fCart2Sph(scatterers->coordinates);
                if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 191 "<<std::endl;
192        torch::Tensor pointing_direction_spherical = fCart2Sph(this->pointing_direction);
                if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 192 "<<std::endl;
193
194        // Calculating relative azimuths and radians
195        torch::Tensor relative_spherical = scatterers_spherical - pointing_direction_spherical;
                if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 199 "<<std::endl;
196
197        // clearing some stuff up
198        scatterers_spherical.reset(); if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 202
                "<<std::endl;
199        pointing_direction_spherical.reset(); if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass:
                line 203 "<<std::endl;
```

```cpp
200
201        // tensor corresponding to switch.
202        torch::Tensor tilt_angle_Tensor = torch::tensor({tilt_angle}).to(torch::kFloat).to(DEVICE);
               if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 206 "<<std::endl;
203
204        torch::Tensor axis_a = torch::tensor({this->azimuthal_beamwidth / 2}).to(torch::kFloat).to(DEVICE);
               if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 208 "<<std::endl;
205        torch::Tensor axis_b = torch::tensor({this->elevation_beamwidth / 2}).to(torch::kFloat).to(DEVICE);
               if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 209 "<<std::endl;
206
207        torch::Tensor xcosa  = relative_spherical[0] * torch::cos(tilt_angle_Tensor * PI/180);
208        torch::Tensor ysina  = relative_spherical[1] * torch::sin(tilt_angle_Tensor * PI/180);
209        torch::Tensor xsina  = relative_spherical[0] * torch::sin(tilt_angle_Tensor * PI/180);
210        torch::Tensor ycosa  = relative_spherical[1] * torch::cos(tilt_angle_Tensor * PI/180);
211        relative_spherical.reset(); if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 215
               "<<std::endl;
212
213        // findings points inside the cone
214        torch::Tensor scatter_boolean = torch::div(torch::square(xcosa + ysina), \
215                                     torch::square(axis_a)) + \
216                            torch::div(torch::square(xsina - ycosa), \
217                                     torch::square(axis_b))    <= 1; if(DEBUGMODE_TRANSMITTER)
                                         std::cout<<"\t\t TransmitterClass: line 221 "<<std::endl;
218
219        // clearing
220        xcosa.reset(); ysina.reset(); xsina.reset(); ycosa.reset(); if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t
               TransmitterClass: line 224 "<<std::endl;
221
222        // subsetting points within the elliptical beam
223        auto mask              = (scatter_boolean == 1); // creating a mask
224        scatterers->coordinates   = scatterers->coordinates.index({torch::indexing::Slice(), mask});
225        scatterers->reflectivity = scatterers->reflectivity.index({torch::indexing::Slice(), mask});
               if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 229 "<<std::endl;
226
227        // // this is where histogram shadowing comes in (later)
228        // rangeHistogramShadowing(scatterers); std::cout<<"\t\t TransmitterClass: line 232 "<<std::endl;
229
230
231        // translating back to the points
232        scatterers->coordinates = scatterers->coordinates + this->location;
233
234    }
235
236    /*========================================================================
237    Aim: Shadowing method (range-histogram shadowing)
238    .........................................................................
239    Note:
240        > cut down the number of threads into range-cells
241        > for each range cell, calculate histogram
242        >
243        std::cout<<"\t TransmitterClass: "
244    ------------------------------------------------------------------------*/
245    void rangeHistogramShadowing(ScattererClass* scatterers){
246
247        // converting points to spherical coordinates
248        torch::Tensor spherical_coordinates = fCart2Sph(scatterers->coordinates); std::cout<<"\t\t
               TransmitterClass: line 252 "<<std::endl;
249
250        // finding maximum range
251        torch::Tensor maxdistanceofpoints = torch::max(spherical_coordinates[2]); std::cout<<"\t\t
               TransmitterClass: line 256 "<<std::endl;
252
253        // calculating number of range-cells (verified)
254        int numrangecells = std::ceil(maxdistanceofpoints.item<int>()/this->rangeQuantSize);
255
256        // finding range-cell boundaries (verified)
257        torch::Tensor rangeBoundaries = \
258            torch::linspace(this->rangeQuantSize, \
259                        numrangecells * this->rangeQuantSize,\
260                        numrangecells); std::cout<<"\t\t TransmitterClass: line 263 "<<std::endl;
261
262        // creating the checkbox (verified)
263        int numazimuthcells  = std::ceil(this->azimuthal_beamwidth * this->azimuthQuantDensity);
```

```
264        int numelevationcells = std::ceil(this->elevation_beamwidth * this->elevationQuantDensity);
               std::cout<<"\t\t TransmitterClass: line 267 "<<std::endl;
265
266        // finding the deltas
267        float delta_azimuth  = this->azimuthal_beamwidth / numazimuthcells;
268        float delta_elevation = this->elevation_beamwidth / numelevationcells; std::cout<<"\t\t
               TransmitterClass: line 271"<<std::endl;
269
270        // creating the centers (verified)
271        torch::Tensor azimuth_centers = torch::linspace(delta_azimuth/2, \
272                                             numazimuthcells * delta_azimuth - delta_azimuth/2, \
273                                             numazimuthcells);
274        torch::Tensor elevation_centers = torch::linspace(delta_elevation/2, \
275                                             numelevationcells * delta_elevation - delta_elevation/2, \
276                                             numelevationcells); std::cout<<"\t\t TransmitterClass:
                                                     line 279"<<std::endl;
277
278        // centering (verified)
279        azimuth_centers   = azimuth_centers + torch::tensor({this->azimuthal_angle - \
280                                                 (this->azimuthal_beamwidth/2)}).to(torch::kFloat);
281        elevation_centers = elevation_centers + torch::tensor({this->elevation_angle - \
282                                                     (this->elevation_beamwidth/2)}).to(torch::kFloat);
                                                         std::cout<<"\t\t TransmitterClass: line
                                                         285"<<std::endl;
283
284        // building checkboxes
285        torch::Tensor checkbox           = torch::zeros({numelevationcells, numazimuthcells}, torch::kBool);
286        torch::Tensor finalScatterBox    = torch::zeros({numelevationcells, numazimuthcells,
               3}).to(torch::kFloat);
287        torch::Tensor finalReflectivityBox = torch::zeros({numelevationcells,
               numazimuthcells}).to(torch::kFloat); std::cout<<"\t\t TransmitterClass: line 290"<<std::endl;
288
289        // going through each-range-cell
290        for(int i = 0; i<(int)rangeBoundaries.numel(); ++i){
291            this->internal_subsetCurrentRangeCell(rangeBoundaries[i], \
292                                             scatterers,           \
293                                             checkbox,             \
294                                             finalScatterBox,      \
295                                             finalReflectivityBox, \
296                                             azimuth_centers,      \
297                                             elevation_centers,    \
298                                             spherical_coordinates); std::cout<<"\t\t TransmitterClass: line
                                                     301"<<std::endl;
299
300            // after each-range-cell
301            torch::Tensor checkboxfilled = torch::sum(checkbox);
302            std::cout<<"\t\t\t\t checkbox-filled = "<<checkboxfilled.item<int>()<<"/"<<checkbox.numel()<<" |
                   percent = "<<100 * checkboxfilled.item<float>()/(float)checkbox.numel()<<std::endl;
303
304        }
305
306        // converting from box structure to [3, num-points] structure
307        torch::Tensor final_coords_spherical = \
308            torch::permute(finalScatterBox, {2, 0, 1}).reshape({3, (int)(finalScatterBox.numel()/3)});
309        torch::Tensor final_coords_cart = fSph2Cart(final_coords_spherical); std::cout<<"\t\t
               TransmitterClass: line 308"<<std::endl;
310        std::cout<<"\t\t finalReflectivityBox.shape = "; fPrintTensorSize(finalReflectivityBox);
311        torch::Tensor final_reflectivity = finalReflectivityBox.reshape({finalReflectivityBox.numel()});
               std::cout<<"\t\t TransmitterClass: line 310"<<std::endl;
312        torch::Tensor test_checkbox = checkbox.reshape({checkbox.numel()}); std::cout<<"\t\t TransmitterClass:
               line 311"<<std::endl;
313
314        // just taking the points corresponding to the filled. Else, there's gonna be a lot of zero zero zero
                   tensors
315        auto mask = (test_checkbox == 1); std::cout<<"\t\t TransmitterClass: line 319"<<std::endl;
316        final_coords_cart = final_coords_cart.index({torch::indexing::Slice(), mask}); std::cout<<"\t\t
               TransmitterClass: line 320"<<std::endl;
317        final_reflectivity = final_reflectivity.index({mask}); std::cout<<"\t\t TransmitterClass: line
               321"<<std::endl;
318
319        // overwriting the scatterers
320        scatterers->coordinates  = final_coords_cart;
321        scatterers->reflectivity = final_reflectivity; std::cout<<"\t\t TransmitterClass: line 324"<<std::endl;
```

```
322
323        }
324
325
326        void internal_subsetCurrentRangeCell(torch::Tensor rangeupperlimit, \
327                                    ScattererClass* scatterers,          \
328                                    torch::Tensor& checkbox,             \
329                                    torch::Tensor& finalScatterBox,      \
330                                    torch::Tensor& finalReflectivityBox, \
331                                    torch::Tensor& azimuth_centers,      \
332                                    torch::Tensor& elevation_centers,    \
333                                    torch::Tensor& spherical_coordinates){
334
335            // finding indices for points in the current range-cell
336            torch::Tensor pointsincurrentrangecell = \
337                torch::mul((spherical_coordinates[2] <= rangeupperlimit) , \
338                        (spherical_coordinates[2] > rangeupperlimit - this->rangeQuantSize));
339
340            // checking out if there are no points in this range-cell
341            int num311 = torch::sum(pointsincurrentrangecell).item<int>();
342            if(num311 == 0) return;
343
344            // calculating delta values
345            float delta_azimuth  = azimuth_centers[1].item<float>() - azimuth_centers[0].item<float>();
346            float delta_elevation = elevation_centers[1].item<float>() - elevation_centers[0].item<float>();
347
348            // subsetting points in the current range-cell
349            auto mask                              = (pointsincurrentrangecell == 1); // creating a mask
350            torch::Tensor reflectivityincurrentrangecell =
                    scatterers->reflectivity.index({torch::indexing::Slice(), mask});
351            pointsincurrentrangecell               = spherical_coordinates.index({torch::indexing::Slice(),
                    mask});
352
353            // finding number of azimuth sizes and what not
354            int numazimuthcells   = azimuth_centers.numel();
355            int numelevationcells = elevation_centers.numel();
356
357            // go through all the combinations
358            for(int azi_index = 0; azi_index < numazimuthcells; ++azi_index){
359                for(int ele_index = 0; ele_index < numelevationcells; ++ele_index){
360
361                    // check if this particular azimuth-elevation direction has been taken-care of.
362                    if (checkbox[ele_index][azi_index].item<bool>()) break;
363
364                    // init (verified)
365                    torch::Tensor current_azimuth = azimuth_centers.index({azi_index});
366                    torch::Tensor current_elevation = elevation_centers.index({ele_index});
367
368                    // // finding azimuth boolean
369                    // torch::Tensor azi_neighbours = torch::abs(pointsincurrentrangecell[0] - current_azimuth);
370                    // azi_neighbours               = azi_neighbours <= delta_azimuth; // tinker with this.
371
372                    // // finding elevation boolean
373                    // torch::Tensor ele_neighbours = torch::abs(pointsincurrentrangecell[1] - current_elevation);
374                    // ele_neighbours               = ele_neighbours <= delta_elevation;
375
376                    // finding azimuth boolean
377                    torch::Tensor azi_neighbours = torch::abs(pointsincurrentrangecell[0] - current_azimuth);
378                    azi_neighbours               = azi_neighbours <= this->azimuthShadowThreshold; // tinker with
                        this.
379
380                    // finding elevation boolean
381                    torch::Tensor ele_neighbours = torch::abs(pointsincurrentrangecell[1] - current_elevation);
382                    ele_neighbours               = ele_neighbours <= this->elevationShadowThreshold;
383
384
385                    // combining booleans: means find all points that are within the limits of both the azimuth and
                        boolean.
386                    torch::Tensor neighbours_boolean = torch::mul(azi_neighbours, ele_neighbours);
387
388                    // checking if there are any points along this direction
389                    int num347 = torch::sum(neighbours_boolean).item<int>();
390                    if (num347 == 0) continue;
```

```
391
392              // findings point along this direction
393              mask                                    = (neighbours_boolean == 1);
394              torch::Tensor coords_along_aziele_spherical =
                     pointsincurrentrangecell.index({torch::indexing::Slice(), mask});
395              torch::Tensor reflectivity_along_aziele =
                     reflectivityincurrentrangecell.index({torch::indexing::Slice(), mask});
396
397              // finding the index where the points are at the maximum distance
398              int index_where_min_range_is = torch::argmin(coords_along_aziele_spherical[2]).item<int>();
399              torch::Tensor closest_coord = coords_along_aziele_spherical.index({torch::indexing::Slice(), \
400                                                                      index_where_min_range_is});
401              torch::Tensor closest_reflectivity = reflectivity_along_aziele.index({torch::indexing::Slice(),
                     \
402                                                                      index_where_min_range_is});
403
404              // filling the matrices up
405              finalScatterBox.index_put_({ele_index, azi_index, torch::indexing::Slice()}, \
406                                  closest_coord.reshape({1,1,3}));
407              finalReflectivityBox.index_put_({ele_index, azi_index}, \
408                                      closest_reflectivity);
409              checkbox.index_put_({ele_index, azi_index}, \
410                          true);
411
412          }
413        }
414    }
415
416
417
418
419 };
```

8.1.3   Class: Uniform Linear Array

The following is the class definition used to encapsulate attributes and methods for the uniform linear array.

```cpp
#include <cstdint>
#include <iostream>
#include <torch/torch.h>

#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"
#include "ScattererClass.h"
#include "TransmitterClass.h"

#pragma once

// hash defines
#ifndef PRINTSPACE
#define PRINTSPACE     std::cout<<"\n\n\n\n\n\n\n"<<std::endl;
#endif
#ifndef PRINTSMALLLINE
#define PRINTSMALLLINE std::cout<<"-----------------------------------------------"<<std::endl;
#endif
#ifndef PRINTLINE
#define PRINTLINE      std::cout<<"==============================================="<<std::endl;
#endif

#ifndef DEVICE
    #define DEVICE        torch::kMPS
    // #define DEVICE        torch::kCPU
#endif

#define PI           3.14159265

// #define DEBUG_ULA true
#define DEBUG_ULA false


class ULAClass{
public:
    // intrinsic parameters
    int num_sensors;                // number of sensors
    float inter_element_spacing;    // space between sensors
    torch::Tensor coordinates;      // coordinates of each sensor
    float sampling_frequency;       // sampling frequency of the sensors
    float recording_period;         // recording period of the ULA
    torch::Tensor location;         // location of first coordinate

    // derived stuff
    torch::Tensor sensorDirection;
    torch::Tensor signalMatrix;

    // constructor
    ULAClass(int numsensors         = 32,
            float inter_element_spacing = 1e-3,
            torch::Tensor coordinates = torch::zeros({3, 2}),
            float sampling_frequency = 48e3,
            float recording_period   = 1):
            num_sensors(numsensors),
            inter_element_spacing(inter_element_spacing),
            coordinates(coordinates),
            sampling_frequency(sampling_frequency),
            recording_period(recording_period) {
                // calculating ULA direction
                torch::Tensor sensorDirection = coordinates.slice(1, 0, 1) - coordinates.slice(1, 1, 2);

                // normalizing
                float normvalue = torch::linalg_norm(sensorDirection, 2, 0, true, torch::kFloat).item<float>();
                if (normvalue != 0){
                    sensorDirection = sensorDirection / normvalue;
                }
```

```
67                    // copying direction
68                    this->sensorDirection = sensorDirection;
69            }
70
71        // overrinding printing
72        friend std::ostream& operator<<(std::ostream& os, ULAClass& ula){
73            os<<"\t number of sensors : "<<ula.num_sensors          <<std::endl;
74            os<<"\t inter-element spacing: "<<ula.inter_element_spacing <<std::endl;
75            os<<"\t sensor-direction "   <<torch::transpose(ula.sensorDirection, 0, 1)<<std::endl;
76            PRINTSMALLLINE
77            return os;
78        }
79
80        // overloading the "=" operator
81        ULAClass& operator=(const ULAClass& other){
82            // checking if copying to the same object
83            if(this == &other){
84                return *this;
85            }
86
87            // copying everything
88            this->num_sensors         = other.num_sensors;
89            this->inter_element_spacing = other.inter_element_spacing;
90            this->coordinates          = other.coordinates.clone();
91            this->sampling_frequency = other.sampling_frequency;
92            this->recording_period   = other.recording_period;
93            this->sensorDirection    = other.sensorDirection.clone();
94
95            // returning
96            return *this;
97        }
98
99        /* ======================================================================
100       Aim: Build coordinates on top of location.
101       ....................................................................
102       Note:
103           > This function builds the location of the coordinates based on the location and direction member.
104       --------------------------------------------------------------------*/
105       void buildCoordinatesBasedOnLocation(){
106
107           // length-normalize the sensor-direction
108           this->sensorDirection = torch::div(this->sensorDirection, torch::linalg_norm(this->sensorDirection, \
109                                                            2, 0, true, \
110                                                            torch::kFloat));
111           if(DEBUG_ULA) std::cout<<"\t ULAClass: line 105 \n";
112
113           // multiply with inter-element distance
114           this->sensorDirection = this->sensorDirection * this->inter_element_spacing;
115           this->sensorDirection = this->sensorDirection.reshape({this->sensorDirection.numel(), 1});
116           if(DEBUG_ULA) std::cout<<"\t ULAClass: line 110 \n";
117
118           // create integer-array
119           // torch::Tensor integer_array = torch::linspace(0, \
120           //                                     this->num_sensors-1, \
121           //                                     this->num_sensors).reshape({1,
122               this->num_sensors}).to(torch::kFloat);
122           torch::Tensor integer_array = torch::linspace(0, \
123                                                this->num_sensors-1, \
124                                                this->num_sensors).reshape({1, this->num_sensors});
125           std::cout<<"integer_array = "; fPrintTensorSize(integer_array);
126           if(DEBUG_ULA) std::cout<<"\t ULAClass: line 116 \n";
127
128
129           // this->coordinates = torch::mul(torch::tile(integer_array, {3, 1}).to(torch::kFloat), \
130           //                       torch::tile(this->sensorDirection, {1,
131               this->num_sensors}).to(torch::kFloat));
131           torch::Tensor test = torch::mul(torch::tile(integer_array, {3, 1}).to(torch::kFloat), \
132                                   torch::tile(this->sensorDirection, {1,
132                                       this->num_sensors}).to(torch::kFloat));
133           this->coordinates = this->location + test;
134           if(DEBUG_ULA) std::cout<<"\t ULAClass: line 120 \n";
135
136       }
```

```
137
138        // Signal Simulation
139        void nfdc_simulateSignal(ScattererClass* scatterers,
140                                 TransmitterClass* transmitterObj){
141
142            // creating signal matrix
143            int numsamples    = std::ceil((this->sampling_frequency * this->recording_period));
144            this->signalMatrix = torch::zeros({numsamples, this->num_sensors}).to(torch::kFloat);
145
146            // getting shape of coordinates
147            std::vector<int64_t> scatterers_coordinates_shape = scatterers->coordinates.sizes().vec();
148
149            // making a slot out of the coordinates
150            torch::Tensor slottedCoordinates = \
151                torch::permute(scatterers->coordinates.reshape({scatterers_coordinates_shape[0], \
152                                                    scatterers_coordinates_shape[1], \
153                                                    1                              }), \
154                        {2, 1, 0}).reshape({1, (int)(scatterers->coordinates.numel()/3), 3});
155
156            // repeating along the y-direction number of sensor times.
157            slottedCoordinates = torch::tile(slottedCoordinates, {this->num_sensors, 1, 1});
158            std::vector<int64_t> slottedCoordinates_shape = slottedCoordinates.sizes().vec();
159
160            // finding the shape of the sensor-coordinates
161            std::vector<int64_t> sensor_coordinates_shape = this->coordinates.sizes().vec();
162
163            // creating a slot tensor out of the sensor-coordinates
164            torch::Tensor slottedSensors = \
165                torch::permute(this->coordinates.reshape({sensor_coordinates_shape[0], \
166                                                    sensor_coordinates_shape[1], \
167                                                    1}), {2, 1, 0}).reshape({(int)(this->coordinates.numel()/3),
                                                                                \
168                                                                  1, \
169                                                                  3});
170
171            // repeating slices along the y-coordinates
172            slottedSensors = torch::tile(slottedSensors, {1, slottedCoordinates_shape[1], 1});
173
174            // slotting the coordinate of the transmitter
175            torch::Tensor slotted_location = torch::permute(this->location.reshape({3, 1, 1}), \
176                                                    {2, 1, 0}).reshape({1,1,3});
177            slotted_location = torch::tile(slotted_location, \
178                                    {slottedCoordinates_shape[0], slottedCoordinates_shape[1], 1});
179
180            // subtracting to find the relative distances
181            torch::Tensor distBetweenScatterersAndSensors = \
182                torch::linalg_norm(slottedCoordinates - slottedSensors, 2, 2, true, torch::kFloat);
183
184            // substracting distance between relative fields
185            torch::Tensor distBetweenScatterersAndTransmitter = \
186                torch::linalg_norm(slottedCoordinates - slotted_location, 2, 2, true, torch::kFloat);
187
188            // adding up the distances
189            torch::Tensor distOfFlight   = distBetweenScatterersAndSensors + distBetweenScatterersAndTransmitter;
190            torch::Tensor timeOfFlight   = distOfFlight/1500;
191            torch::Tensor samplesOfFlight = torch::floor(timeOfFlight.squeeze() * this->sampling_frequency);
192
193
194
195            // // Adding impulses to the signal matrix
196            // for(int sensor_index = 0; sensor_index<this->num_sensors; ++sensor_index){
197            //     for(int scatter_index = 0; scatter_index < scatterers->coordinates[0].numel(); ++scatter_index){
198
199            //          // sample to be added at
200            //          int where_to_place = samplesOfFlight.index({sensor_index, scatter_index}).item<int>();
201            //          // std::cout<<"\t\t\t > where_to_place = "<<where_to_place<<std::endl;
202
203            //          // in case the points are out of bounds
204            //          if (where_to_place >= numsamples) continue;
205
206            //          // assigning impulses
207            //          // this->signalMatrix.index_put_({where_to_place, sensor_index}, \
208            //          //                       this->signalMatrix.index({where_to_place, sensor_index}) + \
```

```
209         //          //                          torch::tensor({1}).to(torch::kFloat));
210
211         //          // // assigning impulses
212         //          // this->signalMatrix.index_put_({where_to_place, sensor_index}, \
213         //          //                  torch::tensor({sensor_index}).to(torch::kFloat));
214
215         //          torch::Tensor bruh = torch::tensor({sensor_index}).to(torch::kFloat);
216         //          this->signalMatrix.index_put_({where_to_place, sensor_index}, bruh);
217         //          // std::cout<<"\t\t\t\t bruh = "<<bruh.item<float>()<<std::endl;
218
219         //     }
220         // }
221
222         // // testing something
223         // std::vector<int64_t> shape1 = samplesOfFlight.sizes().vec();
224         // std::vector<int64_t> shape2 = scatterers->reflectivity.sizes().vec();
225         // PRINTSMALLLINE; std::cout<<"samplesOfFlight.shape = "<<shape1<<std::endl;
226         // std::cout<<"scatters->reflectivity.shape = "<<shape2<<std::e1ndl; PRINTSMALLLINE
227
228         // second writing
229         for(int sensor_index = 0; sensor_index < this->num_sensors; ++sensor_index){
230             for(int scatter_index = 0; scatter_index < samplesOfFlight[0].numel(); ++scatter_index){
231
232                 // getting the sample where the current scatter's contribution must be placed.
233                 int where_to_place = samplesOfFlight.index({sensor_index, scatter_index}).item<int>();
234
235                 // checking whether that point is out of bounds
236                 if(where_to_place >= numsamples) continue;
237
238                 // placing a point in there
239                 this->signalMatrix.index_put_({where_to_place, sensor_index}, \
240                                     this->signalMatrix.index({where_to_place, sensor_index}) + \
241                                     torch::tensor({1}).to(torch::kFloat));
242
243                 this->signalMatrix.index_put_({where_to_place, sensor_index}, \
244                                     this->signalMatrix.index({where_to_place, sensor_index}) + \
245                                       scatterers->reflectivity.index({0, scatter_index}) );
246             }
247         }
248
249         std::cout<<"\t\t\t> this->signalMatrix.shape (after signal sim) = ";
250             fPrintTensorSize(this->signalMatrix);
251         return;
252     }
253 };
```

8.1.4   Class: Autonomous Underwater Vehicle

The following is the class definition used to encapsulate attributes and methods of the marine vessel.

```
1 #include "ScattererClass.h"
2 #include "TransmitterClass.h"
3 #include "ULAClass.h"
4 #include <iostream>
5 #include <ostream>
6 #include <torch/torch.h>
7 #include <cmath>
8
9
10 // // including functions
11 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fGetCurrentTimeFormatted.cpp"
12
13 #pragma once
14
15 // function to plot the thing
16 void fPlotTensors(){
17     system("python /Users/vrsreeganesh/Documents/GitHub/AUV/Code/Python/TestingSaved_tensors.py");
```

```
18  }
19
20
21  void fSaveSeafloorScatteres(ScattererClass scatterer, \
22                              ScattererClass scatterer_fls, \
23                              ScattererClass scatterer_port, \
24                              ScattererClass scatterer_starboard){
25      // saving the tensors
26      if (true) {
27
28          // getting time ID
29          auto timeID = fGetCurrentTimeFormatted();
30
31          std::cout<<"\t\t\t\t\t\t Saving Tensors (timeID: "<<timeID<<")"<<std::endl;
32
33          // saving the ground-truth
34          ScattererClass SeafloorScatter_gt = scatterer;
35          torch::save(SeafloorScatter_gt.coordinates, \
36                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_gt.pt");
37          torch::save(SeafloorScatter_gt.reflectivity, \
38                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_gt_reflectivity.pt");
39
40
41          // saving coordinates
42          torch::save(scatterer_fls.coordinates, \
43                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates.pt");
44          torch::save(scatterer_port.coordinates, \
45                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates.pt");
46          torch::save(scatterer_starboard.coordinates, \
47                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates.pt");
48
49          // saving reflectivities
50          torch::save(scatterer_fls.reflectivity, \
51                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates_reflectivity.pt");
52          torch::save(scatterer_port.reflectivity, \
53                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates_reflectivity.pt");
54          torch::save(scatterer_starboard.reflectivity, \
55                  "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates_reflectivity.pt")
56
57          // plotting tensors
58          fPlotTensors();
59
60          // indicating end of thread
61          std::cout<<"\t\t\t\t\t\t Ended (timeID: "<<timeID<<")"<<std::endl;
62      }
63  }
64
65  // including class-definitions
66  #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"
67
68  // hash defines
69  #ifndef PRINTSPACE
70  #define PRINTSPACE    std::cout<<"\n\n\n\n\n\n\n"<<std::endl;
71  #endif
72  #ifndef PRINTSMALLLINE
73  #define PRINTSMALLLINE std::cout<<"---------------------------------------------"<<std::endl;
74  #endif
75  #ifndef PRINTLINE
76  #define PRINTLINE     std::cout<<"============================================="<<std::endl;
77  #endif
78
79  #ifndef DEVICE
80  #define DEVICE        torch::kMPS
81  // #define DEVICE        torch::kCPU
82  #endif
83
84  #define PI           3.14159265
85  // #define DEBUGMODE_AUV true
86  #define DEBUGMODE_AUV false
87
88
89  class AUVClass{
90  public:
```

```
 91      // Intrinsic attributes
 92      torch::Tensor location;          // location of vessel
 93      torch::Tensor velocity;          // current speed of the vessel [a vector]
 94      torch::Tensor acceleration;      // current acceleration of vessel [a vector]
 95      torch::Tensor pointing_direction; // direction to which the AUV is pointed
 96
 97      // uniform linear-arrays
 98      ULAClass ULA_fls;                // front-looking SONAR ULA
 99      ULAClass ULA_port;               // mounted ULA [object of class, ULAClass]
100      ULAClass ULA_starboard;          // mounted ULA [object of class, ULAClass]
101
102      // transmitters
103      TransmitterClass transmitter_fls;   // transmitter for front-looking SONAR
104      TransmitterClass transmitter_port;  // mounted transmitter [obj of class, TransmitterClass]
105      TransmitterClass transmitter_starboard; // mounted transmitter [obj of class, TransmitterClass]
106
107      // derived or dependent attributes
108      torch::Tensor signalMatrix_1;          // matrix containing the signals obtained from ULA_1
109      torch::Tensor largeSignalMatrix_1;     // matrix holding signal of synthetic aperture
110      torch::Tensor beamformedLargeSignalMatrix;// each column is the beamformed signal at each stop-hop
111
112      // plotting mode
113      bool plottingmode;  // to suppress plotting associated with classes
114
115      // spotlight mode related
116      torch::Tensor absolute_coords_patch_cart; // cartesian coordinates of patch
117
118      // Synthetic Aperture Related
119      torch::Tensor ApertureSensorLocations; // sensor locations of aperture
120
121
122      /*========================================================================
123      Aim: stepping motion
124      ------------------------------------------------------------------------*/
125      void step(float timestep){
126
127          // updating location
128          this->location = this->location + this->velocity * timestep;
129          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 81 \n";
130
131          // updating attributes of members
132          this->syncComponentAttributes();
133          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 85 \n";
134      }
135
136      /*========================================================================
137      Aim: updateAttributes
138      ------------------------------------------------------------------------*/
139      void syncComponentAttributes(){
140
141          // updating ULA attributes
142          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 97 \n";
143
144          // updating locations
145          this->ULA_fls.location       = this->location;
146          this->ULA_port.location      = this->location;
147          this->ULA_starboard.location = this->location;
148
149          // updating the pointing direction of the ULAs
150          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line 99 \n";
151          torch::Tensor ula_fls_sensor_direction_spherical = fCart2Sph(this->pointing_direction);      //
                spherical coords
152          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line 101 \n";
153          ula_fls_sensor_direction_spherical[0]            = ula_fls_sensor_direction_spherical[0] - 90;
154          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line 98 \n";
155          torch::Tensor ula_fls_sensor_direction_cart     = fSph2Cart(ula_fls_sensor_direction_spherical);
156          if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line 100 \n";
157
158          this->ULA_fls.sensorDirection        = ula_fls_sensor_direction_cart; // assigning sensor directionf or
                ULA-FLS
159          this->ULA_port.sensorDirection       = -this->pointing_direction;   // assigning sensor direction for
                ULA-Port
160          this->ULA_starboard.sensorDirection = -this->pointing_direction;    // assigning sensor direction for
```

```
                ULA-Starboard
161         if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line 105 \n";
162
163         // // calling the function to update the arguments
164         // this->ULA_fls.buildCoordinatesBasedOnLocation();  if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line
                109 \n";
165         // this->ULA_port.buildCoordinatesBasedOnLocation(); if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: line
                111 \n";
166         // this->ULA_starboard.buildCoordinatesBasedOnLocation(); if(DEBUGMODE_AUV) std::cout<<"\t AUVClass:
                line 113 \n";
167
168         // updating transmitter locations
169         this->transmitter_fls.location      = this->location;
170         this->transmitter_port.location     = this->location;
171         this->transmitter_starboard.location = this->location;
172         if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 102 \n";
173
174         // updating transmitter pointing directions
175         this->transmitter_fls.updatePointingAngle(     this->pointing_direction);
176         this->transmitter_port.updatePointingAngle(    this->pointing_direction);
177         this->transmitter_starboard.updatePointingAngle( this->pointing_direction);
178         if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 108 \n";
179     }
180
181     /*=======================================================================
182     Aim: operator overriding for printing
183     -----------------------------------------------------------------------*/
184     friend std::ostream& operator<<(std::ostream& os, AUVClass &auv){
185         os<<"\t location = "<<torch::transpose(auv.location, 0, 1)<<std::endl;
186         os<<"\t velocity = "<<torch::transpose(auv.velocity, 0, 1)<<std::endl;
187         return os;
188     }
189
190
191     /*=======================================================================
192     Aim: Subsetting Scatterers
193     -----------------------------------------------------------------------*/
194     void subsetScatterers(ScattererClass* scatterers,\
195                           TransmitterClass* transmitterObj,\
196                           float tilt_angle){
197
198         // ensuring components are synced
199         this->syncComponentAttributes();
200         if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 120 \n";
201
202         // calling the method associated with the transmitter
203         if(DEBUGMODE_AUV) {std::cout<<"\t\t scatterers.shape = "; fPrintTensorSize(scatterers->coordinates);}
204         if(DEBUGMODE_AUV) std::cout<<"\t\t tilt_angle = "<<tilt_angle<<std::endl;
205         transmitterObj->subsetScatterers(scatterers, tilt_angle);
206         if(DEBUGMODE_AUV) std::cout<<"\t AUVClass: page 124 \n";
207     }
208
209
210     // pitch-correction matrix
211     torch::Tensor createYawCorrectionMatrix(torch::Tensor pointing_direction_spherical, \
212                               float target_azimuth_deg){
213
214         // building parameters
215         torch::Tensor azimuth_correction        =
                torch::tensor({target_azimuth_deg}).to(torch::kFloat).to(DEVICE) - \
216                                         pointing_direction_spherical[0];
217         torch::Tensor azimuth_correction_radians = azimuth_correction * PI / 180;
218
219         torch::Tensor yawCorrectionMatrix = \
220             torch::tensor({torch::cos(azimuth_correction_radians).item<float>(), \
221                         torch::cos(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +
                            azimuth_correction_radians).item<float>(), \
222                         (float)0,                                                  \
223                         torch::sin(azimuth_correction_radians).item<float>(), \
224                         torch::sin(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +
                            azimuth_correction_radians).item<float>(), \
225                         (float)0,                                                  \
226                         (float)0,                                                  \
```

```
227                              (float)0,                                                    \
228                              (float)1}).reshape({3,3}).to(torch::kFloat).to(DEVICE);
229
230          // returning the matrix
231          return yawCorrectionMatrix;
232      }
233
234      // pitch-correction matrix
235      torch::Tensor createPitchCorrectionMatrix(torch::Tensor pointing_direction_spherical, \
236                                      float target_elevation_deg){
237
238          // building parameters
239          torch::Tensor elevation_correction        =
240              torch::tensor({target_elevation_deg}).to(torch::kFloat).to(DEVICE) - \
                                            pointing_direction_spherical[1];
241          torch::Tensor elevation_correction_radians = elevation_correction * PI / 180;
242
243          // creating the matrix
244          torch::Tensor pitchCorrectionMatrix = \
245              torch::tensor({(float)1,                                                 \
246                          (float)0,                                                    \
247                          (float)0,                                                    \
248                          (float)0,                                                    \
249                          torch::cos(elevation_correction_radians).item<float>(), \
250                          torch::cos(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +
                                  elevation_correction_radians).item<float>(),\
251                          (float)0,                                                    \
252                          torch::sin(elevation_correction_radians).item<float>(), \
253                          torch::sin(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +
                                  elevation_correction_radians).item<float>()}).reshape({3,3}).to(torch::kFloat);
254
255          // returning the matrix
256          return pitchCorrectionMatrix;
257      }
258
259      // Signal Simulation
260      void simulateSignal(ScattererClass& scatterer){
261
262          // making three copies
263          ScattererClass scatterer_fls      = scatterer;
264          ScattererClass scatterer_port     = scatterer;
265          ScattererClass scatterer_starboard = scatterer;
266
267          // printing the size of these points before subsetting
268          std::cout<< "> AUVClass: Beginning Scatterer Subsetting"<<std::endl;
269          std::cout<<"\t AUVClass: scatterer_fls.coordinates.shape (before) = ";
                  fPrintTensorSize(scatterer_fls.coordinates);
270          std::cout<<"\t AUVClass: scatterer_port.coordinates.shape (before) = ";
                  fPrintTensorSize(scatterer_port.coordinates);
271          std::cout<<"\t AUVClass: scatterer_starboard.coordinates.shape (before) = ";
                  fPrintTensorSize(scatterer_starboard.coordinates);
272
273          // finding the pointing direction in spherical
274          torch::Tensor auv_pointing_direction_spherical = fCart2Sph(this->pointing_direction);
275
276          // asking the transmitters to subset the scatterers by multithreading
277          std::thread transmitterFLSSubset_t(&AUVClass::subsetScatterers, this, \
278                                  &scatterer_fls,\
279                                  &this->transmitter_fls, \
280                                  (float)0);
281          std::thread transmitterPortSubset_t(&AUVClass::subsetScatterers, this, \
282                                  &scatterer_port,\
283                                  &this->transmitter_port, \
284                                  - auv_pointing_direction_spherical[1].item<float>());
285          std::thread transmitterStarboardSubset_t(&AUVClass::subsetScatterers, this, \
286                                      &scatterer_starboard, \
287                                      &this->transmitter_starboard, \
288                                      auv_pointing_direction_spherical[1].item<float>());
289
290          // joining the subset threads back
291          transmitterFLSSubset_t.join(); transmitterPortSubset_t.join(); transmitterStarboardSubset_t.join();
292
293          // printing the size of these points before subsetting
```

```
294        PRINTDOTS
295        std::cout<<"\t AUVClass: scatterer_fls.coordinates.shape (after) = ";
               fPrintTensorSize(scatterer_fls.coordinates);
296        std::cout<<"\t AUVClass: scatterer_port.coordinates.shape (after) = ";
               fPrintTensorSize(scatterer_port.coordinates);
297        std::cout<<"\t AUVClass: scatterer_starboard.coordinates.shape (after) = ";
               fPrintTensorSize(scatterer_starboard.coordinates);
298
299        // // multithreading the saving tensors part.
300        // std::thread savetensor_t(fSaveSeafloorScatteres, \
301        //                    scatterer,                 \
302        //                    scatterer_fls,             \
303        //                    scatterer_port,            \
304        //                    scatterer_starboard);
305        // savetensor_t.detach();
306
307        // saving the tensors
308        if (true) {
309            // saving the ground-truth
310            ScattererClass SeafloorScatter_gt = scatterer;
311            torch::save(SeafloorScatter_gt.coordinates,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_gt.pt");
312            torch::save(SeafloorScatter_gt.reflectivity,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_gt_reflectivity.pt");
313
314            // saving coordinates
315            torch::save(scatterer_fls.coordinates,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates.pt");
316            torch::save(scatterer_port.coordinates,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates.pt");
317            torch::save(scatterer_starboard.coordinates,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates.pt");
318
319            // saving reflectivities
320            torch::save(scatterer_fls.reflectivity,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates_reflectivity.pt");
321            torch::save(scatterer_port.reflectivity,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates_reflectivity.pt");
322            torch::save(scatterer_starboard.reflectivity,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates_reflectivity.pt");
323
324            // plotting tensors
325            fPlotTensors();
326        }
327
328        // asking ULAs to simulate signal through multithreading
329        std::thread ulafls_signalsim_t(&ULAClass::nfdc_simulateSignal,   \
330                            &this->ULA_fls,                  \
331                            &scatterer_fls,                  \
332                            &this->transmitter_fls);
333        std::thread ulaport_signalsim_t(&ULAClass::nfdc_simulateSignal,  \
334                            &this->ULA_port,                 \
335                            &scatterer_port,                 \
336                            &this->transmitter_port);
337        std::thread ulastarboard_signalsim_t(&ULAClass::nfdc_simulateSignal, \
338                                &this->ULA_starboard,            \
339                                &scatterer_starboard,            \
340                                &this->transmitter_starboard);
341
342        // joining them back
343        ulafls_signalsim_t.join();
344        ulaport_signalsim_t.join();
345        ulastarboard_signalsim_t.join();
346
347        // saving the tensors
348        if (true) {
349            // saving the ground-truth
350            torch::save(this->ULA_fls.signalMatrix,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/signalMatrix_fls.pt");
351            torch::save(this->ULA_port.signalMatrix,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/signalMatrix_port.pt");
352            torch::save(this->ULA_starboard.signalMatrix,
                   "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/signalMatrix_starboard.pt");
```

```
353           }
354
355
356       }
357
358
359  };
```

8.2   Setup Scripts

8.2.1   Seafloor Setup

Following is the script to be run to setup the seafloor.

```cpp
/* =====================================
Aim: Setup sea floor
=====================================*/
#include <torch/torch.h>
#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"

#ifndef DEVICE
    // #define DEVICE        torch::kMPS
    #define DEVICE        torch::kCPU
#endif


// adding terrrain features
#define BOXES       true
#define TERRAIN     false
#define DEBUG_SEAFLOOR false



// Adding boxes
void fCreateBoxes(float across_track_length, \
                  float along_track_length, \
                  torch::Tensor& box_coordinates,\
                  torch::Tensor& box_reflectivity){

    // converting arguments to torch tensos

    // setting up parameters
    float min_width       = 2;     // minimum across-track dimension of the boxes in the sea-floor
    float max_width       = 5;     // maximum across-track dimension of the boxes in the sea-floor

    float min_length      = 2;     // minimum along-track dimension of the boxes in the sea-floor
    float max_length      = 5;     // maximum along-track dimension of the boxes in the sea-floor

    float min_height      = 3;     // minimum height of the boxes in the sea-floor
    float max_height      = 20;    // maximum height of the boxes in the sea-floor

    int meshdensity       = 10;     // number of points per meter.
    float meshreflectivity = 2;     // average reflectivity of the mesh

    int num_boxes         = 10;    // number of boxes in the sea-floor
    if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 41\n";

    // finding center point
    torch::Tensor midxypoints = torch::rand({3, num_boxes}).to(torch::kFloat).to(DEVICE);
    midxypoints[0]            = midxypoints[0] * across_track_length;
    midxypoints[1]            = midxypoints[1] * along_track_length;
    midxypoints[2]            = 0;
    if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 48\n";

    // assigning dimensions to boxes
    torch::Tensor boxwidths = torch::rand({num_boxes})*(max_width - min_width) + min_width; // assigning
            widths to each boxes
    torch::Tensor boxlengths = torch::rand({num_boxes})*(max_length - min_length) + min_length; // assigning
            lengths to each boxes
    torch::Tensor boxheights = torch::rand({num_boxes})*(max_height - min_height) + min_height; // assigning
            heights to each boxes
    if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 54\n";

    // creating mesh for each box
    for(int i = 0; i<num_boxes; ++i){

        // finding x-points
        torch::Tensor xpoints = torch::linspace(-boxwidths[i].item<float>()/2, \
                                                 boxwidths[i].item<float>()/2, \
```

```
63                                              (int)(boxwidths[i].item<float>() * meshdensity));
64          torch::Tensor ypoints = torch::linspace(-boxlengths[i].item<float>()/2, \
65                                      boxlengths[i].item<float>()/2, \
66                                      (int)(boxlengths[i].item<float>() * meshdensity));
67          torch::Tensor zpoints = torch::linspace(0, \
68                                      boxheights[i].item<float>(),\
69                                      (int)(boxheights[i].item<float>() * meshdensity));
70          if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 69\n";
71
72          // meshgridding
73          auto mesh_grid = torch::meshgrid({xpoints, ypoints, zpoints}, "xy");
74          auto X          = mesh_grid[0];
75          auto Y          = mesh_grid[1];
76          auto Z          = mesh_grid[2];
77          X               = torch::reshape(X, {1, X.numel()});
78          Y               = torch::reshape(Y, {1, Y.numel()});
79          Z               = torch::reshape(Z, {1, Z.numel()});
80          if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 79\n";
81
82          // coordinates
83          torch::Tensor boxcoordinates = torch::cat({X, Y, Z}, 0).to(DEVICE);
84          boxcoordinates[0] = boxcoordinates[0] + midxypoints[0][i];
85          boxcoordinates[1] = boxcoordinates[1] + midxypoints[1][i];
86          boxcoordinates[2] = boxcoordinates[2] + midxypoints[2][i];
87          if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 86\n";
88
89          // creating some reflectivity points too.
90          torch::Tensor boxreflectivity = meshreflectivity + torch::rand({1, boxcoordinates[0].numel()}) - 0.5;
91          boxreflectivity = boxreflectivity.to(DEVICE);
92          if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 90\n";
93
94          // adding to larger matrices
95          if(DEBUG_SEAFLOOR) {std::cout<<"box_coordinates.shape = "; fPrintTensorSize(box_coordinates);}
96          if(DEBUG_SEAFLOOR) {std::cout<<"box_coordinates.shape = "; fPrintTensorSize(boxcoordinates);}
97
98          if(DEBUG_SEAFLOOR) {std::cout<<"box_reflectivity.shape = "; fPrintTensorSize(box_reflectivity);}
99          if(DEBUG_SEAFLOOR) {std::cout<<"boxreflectivity.shape = "; fPrintTensorSize(boxreflectivity);}
100
101         box_coordinates   = torch::cat({box_coordinates.to(DEVICE), boxcoordinates}, 1);
102         if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 95\n";
103         box_reflectivity  = torch::cat({box_reflectivity.to(DEVICE), boxreflectivity}, 1);
104         if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 97\n";
105     }
106 }
107
108
109
110 // functin that setups the sea-floor
111 void SeafloorSetup(ScattererClass* scatterers) {
112
113     // sea-floor bounds
114     int bed_width = 100; // width of the bed (x-dimension)
115     int bed_length = 100; // length of the bed (y-dimension)
116
117     // multithreading the box creation
118
119     // creating some tensors to pass. This is put outside to maintain scope
120     bool add_boxes_flag = BOXES;
121     torch::Tensor box_coordinates = torch::zeros({3,1}).to(torch::kFloat).to(DEVICE);
122     torch::Tensor box_reflectivity = torch::zeros({1,1}).to(torch::kFloat).to(DEVICE);
123     // std::thread boxes_t(fCreateBoxes, \
124     //                  bed_width, bed_length, \
125     //                  &box_coordinates, &box_reflectivity);
126     fCreateBoxes(bed_width, \
127              bed_length, \
128              box_coordinates, \
129              box_reflectivity);
130
131     // scatter-intensity
132     // int bed_width_density   = 100; // density of points along x-dimension
133     // int bed_length_density  = 100; // density of points along y-dimension
134     int bed_width_density    = 10; // density of points along x-dimension
135     int bed_length_density   = 10; // density of points along y-dimension
```

```
136
137        // setting up coordinates
138        auto xpoints = torch::linspace(0, \
139                                  bed_width, \
140                                  bed_width * bed_width_density).to(DEVICE);
141        auto ypoints = torch::linspace(0, \
142                                  bed_length, \
143                                  bed_length * bed_length_density).to(DEVICE);
144
145        // creating mesh
146        auto mesh_grid = torch::meshgrid({xpoints, ypoints}, "ij");
147        auto X         = mesh_grid[0];
148        auto Y         = mesh_grid[1];
149        X              = torch::reshape(X, {1, X.numel()});
150        Y              = torch::reshape(Y, {1, Y.numel()});
151
152        // creating heights of scattereres
153        torch::Tensor Z = torch::zeros({1, Y.numel()}).to(DEVICE);
154
155        // setting up floor coordinates
156        torch::Tensor floorScatter_coordinates = torch::cat({X, Y, Z}, 0);
157        torch::Tensor floorScatter_reflectivity = torch::ones({1, Y.numel()}).to(DEVICE);
158
159        // populating the values of the incoming argument.
160        scatterers->coordinates  = floorScatter_coordinates; // assigning coordinates
161        scatterers->reflectivity = floorScatter_reflectivity;// assigning reflectivity
162
163        // // rejoining if multithreading
164        // boxes_t.join();// bringing thread back
165
166        // combining the values
167        if(DEBUG_SEAFLOOR) std::cout<<"\t SeafloorSetup: line 166 \n";
168        if(DEBUG_SEAFLOOR) {std::cout<<"\t scatterers->coordinates.shape = ";
                fPrintTensorSize(scatterers->coordinates);}
169        if(DEBUG_SEAFLOOR) {std::cout<<"\t box_coordinates.shape = "; fPrintTensorSize(box_coordinates);}
170        if(DEBUG_SEAFLOOR) {std::cout<<"\t scatterers->reflectivity.shape = ";
                fPrintTensorSize(scatterers->reflectivity);}
171        if(DEBUG_SEAFLOOR) {std::cout<<"\t box_reflectivity = "; fPrintTensorSize(box_reflectivity);}
172
173
174        scatterers->coordinates  = torch::cat({scatterers->coordinates, box_coordinates}, 1);
175        PRINTLINE
176        scatterers->reflectivity = torch::cat({scatterers->reflectivity, box_reflectivity}, 1);
177        PRINTSMALLLINE
178
179
180    }
```

8.2.2 Transmitter Setup

Following is the script to be run to setup the transmitter.

```
1    /* ====================================
2    Aim: Setup sea floor
3    ====================================*/
4    #include <torch/torch.h>
5    #include <cmath>
6
7    #ifndef DEVICE
8        // #define DEVICE        torch::kMPS
9        #define DEVICE        torch::kCPU
10   #endif
11
12
13
14   // function to calibrate the transmitters
15   void TransmitterSetup(TransmitterClass* transmitter_fls,
16                    TransmitterClass* transmitter_port,
17                    TransmitterClass* transmitter_starboard) {
```

```cpp
18
19      // Setting up transmitter
20      float sampling_frequency = 160e3;                   // sampling frequency
21      float f1              = 50e3;                        // first frequency of LFM
22      float f2              = 70e3;                        // second frequency of LFM
23      float fc              = (f1 + f2)/2;                 // finding center-frequency
24      float bandwidth       = std::abs(f2 - f1); // bandwidth
25      float pulselength     = 0.2;                         // time of recording
26
27      // building LFM
28      torch::Tensor timearray = torch::linspace(0, \
29                                  pulselength, \
30                                  floor(pulselength * sampling_frequency)).to(DEVICE);
31      float K               = (f2 - f1)/pulselength;       // calculating frequency-slope
32      torch::Tensor Signal = K * timearray;               // frequency at each time-step, with f1 = 0
33      Signal                = torch::mul(2*PI*(f1 + Signal), \
34                                  timearray);              // creating
35      Signal                = cos(Signal);                 // calculating signal
36
37
38      // Setting up transmitter
39      torch::Tensor location             = torch::zeros({3,1}).to(DEVICE); // location of transmitter
40      float azimuthal_angle_fls          = 0;                 // initial pointing direction
41      float azimuthal_angle_port         = 90;                // initial pointing direction
42      float azimuthal_angle_starboard    = -90;               // initial pointing direction
43
44      float elevation_angle              = -60;               // initial pointing direction
45
46      float azimuthal_beamwidth_fls      = 90;                // azimuthal beamwidth of the signal cone
47      float azimuthal_beamwidth_port     = 20;                // azimuthal beamwidth of the signal cone
48      float azimuthal_beamwidth_starboard = 20;               // azimuthal beamwidth of the signal cone
49
50      float elevation_beamwidth_fls      = 40;                // elevation beamwidth of the signal cone
51      float elevation_beamwidth_port     = 40;                // elevation beamwidth of the signal cone
52      float elevation_beamwidth_starboard = 40;               // elevation beamwidth of the signal cone
53
54      int azimuthQuantDensity      = 10;   // number of points, a degree is split into quantization density
            along azimuth (used for shadowing)
55      int elevationQuantDensity    = 10;   // number of points, a degree is split into quantization density
            along elevation (used for shadowing)
56      float rangeQuantSize         = 10;   // the length of a cell (used for shadowing)
57
58      float azimuthShadowThreshold = 1;    // azimuth threshold   (in degrees)
59      float elevationShadowThreshold = 1;  // elevation threshold  (in degrees)
60
61
62
63      // transmitter-fls
64      transmitter_fls->location            = location;            // Assigning location
65      transmitter_fls->Signal              = Signal;              // Assigning signal
66      transmitter_fls->azimuthal_angle     = azimuthal_angle_fls; // assigning azimuth angle
67      transmitter_fls->elevation_angle     = elevation_angle;     // assigning elevation angle
68      transmitter_fls->azimuthal_beamwidth = azimuthal_beamwidth_fls; // assigning azimuth-beamwidth
69      transmitter_fls->elevation_beamwidth = elevation_beamwidth_fls; // assigning elevation-beamwidth
70      // updating quantization densities
71      transmitter_fls->azimuthQuantDensity   = azimuthQuantDensity;    // assigning azimuth quant density
72      transmitter_fls->elevationQuantDensity = elevationQuantDensity;  // assigning elevation quant density
73      transmitter_fls->rangeQuantSize        = rangeQuantSize;         // assigning range-quantization
74      transmitter_fls->azimuthShadowThreshold = azimuthShadowThreshold;  // azimuth-threshold in shadowing
75      transmitter_fls->elevationShadowThreshold = elevationShadowThreshold; // elevation-threshold in shadowing
76      // signal related
77      transmitter_fls->f_low               = f1;         // assigning lower frequency
78      transmitter_fls->f_high              = f2;         // assigning higher frequency
79      transmitter_fls->fc                  = fc;         // assigning center frequency
80      transmitter_fls->bandwidth           = bandwidth;  // assigning bandwidth
81
82
83
84      // transmitter-portside
85      transmitter_port->location           = location;              // Assigning location
86      transmitter_port->Signal             = Signal;                // Assigning signal
87      transmitter_port->azimuthal_angle    = azimuthal_angle_port;  // assigning azimuth angle
88      transmitter_port->elevation_angle    = elevation_angle;       // assigning elevation angle
```

```
 89        transmitter_port->azimuthal_beamwidth  = azimuthal_beamwidth_port;  // assigning azimuth-beamwidth
 90        transmitter_port->elevation_beamwidth  = elevation_beamwidth_port;  // assigning elevation-beamwidth
 91        // updating quantization densities
 92        transmitter_port->azimuthQuantDensity  = azimuthQuantDensity;       // assigning azimuth quant density
 93        transmitter_port->elevationQuantDensity = elevationQuantDensity;    // assigning elevation quant density
 94        transmitter_port->rangeQuantSize       = rangeQuantSize;            // assigning range-quantization
 95        transmitter_port->azimuthShadowThreshold = azimuthShadowThreshold;  // azimuth-threshold in shadowing
 96        transmitter_port->elevationShadowThreshold = elevationShadowThreshold; // elevation-threshold in shadowing
 97        // signal related
 98        transmitter_port->f_low                = f1;                        // assigning lower frequency
 99        transmitter_port->f_high               = f2;                        // assigning higher frequency
100        transmitter_port->fc                   = fc;                        // assigning center frequency
101        transmitter_port->bandwidth            = bandwidth;                 // assigning bandwidth
102
103
104
105        // transmitter-starboard
106        transmitter_starboard->location               = location;                // assigning location
107        transmitter_starboard->Signal                 = Signal;                  // assigning signal
108        transmitter_starboard->azimuthal_angle        = azimuthal_angle_starboard; // assigning azimuthal signal
109        transmitter_starboard->elevation_angle        = elevation_angle;
110        transmitter_starboard->azimuthal_beamwidth    = azimuthal_beamwidth_starboard;
111        transmitter_starboard->elevation_beamwidth    = elevation_beamwidth_starboard;
112        // updating quantization densities
113        transmitter_starboard->azimuthQuantDensity    = azimuthQuantDensity;
114        transmitter_starboard->elevationQuantDensity  = elevationQuantDensity;
115        transmitter_starboard->rangeQuantSize         = rangeQuantSize;
116        transmitter_starboard->azimuthShadowThreshold = azimuthShadowThreshold;
117        transmitter_starboard->elevationShadowThreshold = elevationShadowThreshold;
118        // signal related
119        transmitter_starboard->f_low                  = f1;          // assigning lower frequency
120        transmitter_starboard->f_high                 = f2;          // assigning higher frequency
121        transmitter_starboard->fc                     = fc;          // assigning center frequency
122        transmitter_starboard->bandwidth              = bandwidth;   // assigning bandwidth
123
124 }
```

8.2.3   Uniform Linear Array

Following is the script to be run to setup the uniform linear array.

```
 1  /* ====================================
 2  Aim: Setup sea floor
 3  NOAA: 50 to 100 KHz is the transmission frequency
 4  we'll create our LFM with 50 to 70KHz
 5  ====================================*/
 6
 7
 8  // Choosing device
 9  #ifndef DEVICE
10      // #define DEVICE        torch::kMPS
11      #define DEVICE         torch::kCPU
12  #endif
13
14
15
16
17  void ULASetup(ULAClass* ula_fls,
18              ULAClass* ula_port,
19              ULAClass* ula_starboard) {
20
21      // setting up ula
22      int num_sensors        = 64;                        // number of sensors
23      float sampling_frequency = 160e3;                   // sampling frequency
24      float inter_element_spacing = 1500/(2*sampling_frequency); // space between samples
25      float recording_period   = 1;                       // sampling-period
26
27      // building the direction for the sensors
28      torch::Tensor ULA_direction = torch::tensor({-1,0,0}).reshape({3,1}).to(torch::kFloat).to(DEVICE);
```

```
29      ULA_direction           = ULA_direction/torch::linalg_norm(ULA_direction, 2, 0, true,
            torch::kFloat).to(DEVICE);
30      ULA_direction           = ULA_direction * inter_element_spacing;
31
32      // building the coordinates for the sensors
33      torch::Tensor ULA_coordinates = torch::mul(torch::linspace(0, num_sensors-1, num_sensors).to(DEVICE), \
34                                  ULA_direction);
35
36      // assigning values
37      ula_fls->num_sensors        = num_sensors;              // assigning number of sensors
38      ula_fls->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
39      ula_fls->coordinates        = ULA_coordinates;          // assigning ULA coordinates
40      ula_fls->sampling_frequency = sampling_frequency;       // assigning sampling frequencys
41      ula_fls->recording_period   = recording_period;         // assigning recording period
42      ula_fls->sensorDirection    = ULA_direction;            // ULA direction
43
44      ula_fls->num_sensors        = num_sensors;              // assigning number of sensors
45      ula_fls->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
46      ula_fls->coordinates        = ULA_coordinates;          // assigning ULA coordinates
47      ula_fls->sampling_frequency = sampling_frequency;       // assigning sampling frequencys
48      ula_fls->recording_period   = recording_period;         // assigning recording period
49      ula_fls->sensorDirection    = ULA_direction;            // ULA direction
50
51      // assigning values
52      ula_port->num_sensors       = num_sensors;              // assigning number of sensors
53      ula_port->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
54      ula_port->coordinates       = ULA_coordinates;          // assigning ULA coordinates
55      ula_port->sampling_frequency = sampling_frequency;      // assigning sampling frequencys
56      ula_port->recording_period  = recording_period;         // assigning recording period
57      ula_port->sensorDirection   = ULA_direction;            // ULA direction
58
59      ula_port->num_sensors       = num_sensors;              // assigning number of sensors
60      ula_port->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
61      ula_port->coordinates       = ULA_coordinates;          // assigning ULA coordinates
62      ula_port->sampling_frequency = sampling_frequency;      // assigning sampling frequencys
63      ula_port->recording_period  = recording_period;         // assigning recording period
64      ula_port->sensorDirection   = ULA_direction;            // ULA direction
65
66
67      // assigning values
68      ula_starboard->num_sensors      = num_sensors;              // assigning number of sensors
69      ula_starboard->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
70      ula_starboard->coordinates      = ULA_coordinates;          // assigning ULA coordinates
71      ula_starboard->sampling_frequency = sampling_frequency;     // assigning sampling frequencys
72      ula_starboard->recording_period = recording_period;         // assigning recording period
73      ula_starboard->sensorDirection  = ULA_direction;            // ULA direction
74
75      ula_starboard->num_sensors      = num_sensors;              // assigning number of sensors
76      ula_starboard->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
77      ula_starboard->coordinates      = ULA_coordinates;          // assigning ULA coordinates
78      ula_starboard->sampling_frequency = sampling_frequency;     // assigning sampling frequencys
79      ula_starboard->recording_period = recording_period;         // assigning recording period
80      ula_starboard->sensorDirection  = ULA_direction;            // ULA direction
81
82  }
```

8.2.4   AUV Setup

Following is the script to be run to setup the vessel.

```
1   /* =====================================
2   Aim: Setup sea floor
3   NOAA: 50 to 100 KHz is the transmission frequency
4   we'll create our LFM with 50 to 70KHz
5   =====================================*/
6
7   #ifndef DEVICE
8       #define DEVICE          torch::kMPS
9       // #define DEVICE          torch::kCPU
```

```cpp
10   #endif
11
12   // ==========================================================
13   void AUVSetup(AUVClass* auv) {
14
15       // building properties for the auv
16       torch::Tensor location          = torch::tensor({0,50,30}).reshape({3,1}).to(torch::kFloat).to(DEVICE); //
                starting location of AUV
17       torch::Tensor velocity          = torch::tensor({5,0, 0}).reshape({3,1}).to(torch::kFloat).to(DEVICE); //
                starting velocity of AUV
18       torch::Tensor pointing_direction = torch::tensor({1,0, 0}).reshape({3,1}).to(torch::kFloat).to(DEVICE);
                // pointing direction of AUV
19
20       // assigning
21       auv->location          = location;              // assigning location of auv
22       auv->velocity          = velocity;              // assigning vector representing velocity
23       auv->pointing_direction = pointing_direction; // assigning pointing direction of auv
24   }
```

8.3   Function Definitions

8.3.1   Cartesian Coordinates to Spherical Coordinates

```cpp
/* ====================================
Aim: Setup sea floor
====================================*/
#include <torch/torch.h>
#include <iostream>

// hash-defines
#define PI          3.14159265
#define DEBUG_Cart2Sph false

#ifndef DEVICE
    #define DEVICE        torch::kMPS
    // #define DEVICE        torch::kCPU
#endif


// bringing in functions
#include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"

#pragma once

torch::Tensor fCart2Sph(torch::Tensor cartesian_vector){

    // sending argument to the device
    cartesian_vector = cartesian_vector.to(DEVICE);
    if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 26 \n";

    // splatting the point onto xy plane
    torch::Tensor xysplat = cartesian_vector.clone().to(DEVICE);
    xysplat[2] = 0;
    if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 31 \n";

    // finding splat lengths
    torch::Tensor xysplat_lengths = torch::linalg_norm(xysplat, 2, 0, true, torch::kFloat).to(DEVICE);
    if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 35 \n";

    // finding azimuthal and elevation angles
    torch::Tensor azimuthal_angles = torch::atan2(xysplat[1],     xysplat[0]).to(DEVICE)   * 180/PI;
    azimuthal_angles              = azimuthal_angles.reshape({1, azimuthal_angles.numel()});
    torch::Tensor elevation_angles = torch::atan2(cartesian_vector[2], xysplat_lengths).to(DEVICE) * 180/PI;
    torch::Tensor rho_values     = torch::linalg_norm(cartesian_vector, 2, 0, true, torch::kFloat).to(DEVICE);
    if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 42 \n";


    // printing values for debugging
    if (DEBUG_Cart2Sph){
        std::cout<<"azimuthal_angles.shape = "; fPrintTensorSize(azimuthal_angles);
        std::cout<<"elevation_angles.shape = "; fPrintTensorSize(elevation_angles);
        std::cout<<"rho_values.shape     = "; fPrintTensorSize(rho_values);
    }
    if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 51 \n";

    // creating tensor to send back
    torch::Tensor spherical_vector = torch::cat({azimuthal_angles, \
                                        elevation_angles, \
                                        rho_values}, 0).to(DEVICE);
    if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 57 \n";

    // returning the value
    return spherical_vector;
}
```

Chapter 9

Reading

9.1 Primary Books

1.

9.2 Interesting Papers