

Autonomous Underwater Vehicle: A Surveillance Protocol

S.V. Rajendran

February 2, 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 obtained from that of NOAA ships that are publicly 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

Preface	i
Introduction	ii
1 Setup	1
1.1 Overview	1
2 Underwater Environment Setup	2
2.1 Seafloor Setup	2
2.2 Additional Structures	2
3 Hardware Setup	3
3.1 Transmitter	3
3.2 Uniform Linear Array	3
3.3 Marine Vessel	3
4 Geometry	4
4.1 Ray Tracing	4
4.1.1 Pairwise Dot-Product	4
4.1.2 Range Histogram Method	4
5 Signal Simulation	6
5.1 Transmitted Signal	6
5.2 Signal Simulation	6
6 Imaging	8
7 Results	10
8 Software	11
8.1 Class Definitions	11
8.1.1 Class: Scatter	11
8.1.2 Class: Transmitter	13
8.1.3 Class: Uniform Linear Array	20
8.1.4 Class: Autonomous Underwater Vehicle	23
8.2 Setup Scripts	29
8.2.1 Seafloor Setup	29
8.2.2 Transmitter Setup	31

8.2.3	Uniform Linear Array	33
8.2.4	AUV Setup	34
8.3	Function Definitions	36
8.3.1	Cartesian Coordinates to Spherical Coordinates	36
9	Reading	37
9.1	Primary Books	37
9.2	Interesting Papers	37

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

3.1 Transmitter

3.2 Uniform Linear Array

3.3 Marine Vessel

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 others in the following range cells.

Algorithm 1 Range Histogram Method

ScatterCoordinates \leftarrow
ScatterReflectivity \leftarrow
AngleDensity \leftarrow Quantization of angles per degree.
AzimuthalBeamwidth \leftarrow Azimuthal Beamwidth
RangeCellWidth \leftarrow 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 flat 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.

```
1 // header-files
2 #include <iostream>
3 #include <ostream>
4 #include <torch/torch.h>
5
6 #pragma once
7
8 // hash defines
9 #ifndef PRINTSPACE
10 #define PRINTSPACE    std::cout<<"\n\n\n\n\n\n\n\n\n\n"<<std::endl;
11 #endif
12 #ifndef PRINTSMALLLINE
13 #define PRINTSMALLLINE std::cout<<"-----"<<std::endl;
14 #endif
15 #ifndef PRINTLINE
16 #define PRINTLINE    std::cout<<"===== "<<std::endl;
17 #endif
18 #ifndef DEVICE
19     #define DEVICE    torch::kMPS
20     // #define DEVICE    torch::kCPU
21 #endif
22
23
24 #define PI    3.14159265
25
26
27 // function to print tensor size
28 void print_tensor_size(const torch::Tensor& inputTensor) {
29     // Printing size
30     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.

```

1 // header-files
2 #include <iostream>
3 #include <ostream>
4 #include <cmath>
5
6 // Including classes
7 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"
8
9 // Including functions
10 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fCart2Sph.cpp"
11 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"
12 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fSph2Cart.cpp"
13
14 #pragma once
15
16 // hash defines
17 #ifndef PRINTSPACE
18 # define PRINTSPACE      std::cout<<"\n\n\n\n\n\n\n\n\n\n"<<std::endl;
19 #endif
20 #ifndef PRINTSMALLLINE
21 # define PRINTSMALLLINE std::cout<<"-----"<<std::endl;
22 #endif
23 #ifndef PRINTLINE
24 # define PRINTLINE      std::cout<<"===== "<<std::endl;
25 #endif
26
27 #define PI              3.14159265
28 #define DEBUGMODE_TRANSMITTER  false
29
30 #ifndef DEVICE
31 #define DEVICE          torch::kMPS
32 // #define DEVICE        torch::kCPU
33 #endif
34
35
36 class TransmitterClass{
37 public:
38
39     // physical/intrinsic properties
40     torch::Tensor location;          // location tensor
41     torch::Tensor pointing_direction; // pointing direction
42
43     // basic parameters
44     torch::Tensor Signal;           // transmitted signal (LFM)
45     float azimuthal_angle;          // transmitter's azimuthal pointing direction
46     float elevation_angle;          // transmitter's elevation pointing direction
47     float azimuthal_beamwidth;       // azimuthal beamwidth of transmitter
48     float elevation_beamwidth;       // elevation beamwidth of transmitter
49     float range;                    // a parameter used for spotlight mode.
50
51     // transmitted signal attributes
52     float f_low;                    // lowest frequency of LFM
53     float f_high;                   // highest frequency of LFM
54     float fc;                       // center frequency of LFM
55     float bandwidth;                // bandwidth of LFM
56
57     // shadowing properties
58     int azimuthQuantDensity;         // quantization of angles along the azimuth
59     int elevationQuantDensity;       // quantization of angles along the elevation
60     float rangeQuantSize;            // range-cell size when shadowing
61     float azimuthShadowThreshold;    // azimuth thresholding
62     float elevationShadowThreshold;  // elevation thresholding
63
64     // // shadowing related
65     // 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 azimuthal_angle : "<<transmitter.azimuthal_angle <<std::endl;
88     os<<"\t elevation_angle : "<<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     // std::cout<<"\t TransmitterClass: AUV_pointing_vector_spherical =
152     // std::cout<<"\t TransmitterClass: AUV_pointing_vector_spherical =
153     // calculating azimuth and elevation of transmitter object
154     torch::Tensor absolute_azimuth_of_transmitter = yaw +
155     torch::tensor({this->azimuthal_angle}).to(torch::kFloat).to(DEVICE);
156     torch::Tensor absolute_elevation_of_transmitter = pitch +
157     torch::tensor({this->elevation_angle}).to(torch::kFloat).to(DEVICE);
158     if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 149 \n";
159
160     // std::cout<<"\t TransmitterClass: this->azimuthal_angle = "<<this->azimuthal_angle<<std::endl;
161     // std::cout<<"\t TransmitterClass: this->elevation_angle = "<<this->elevation_angle<<std::endl;
162     // std::cout<<"\t TransmitterClass: absolute_azimuth_of_transmitter =
163     // std::cout<<"\t TransmitterClass: absolute_elevation_of_transmitter =
164     // converting back to Cartesian
165     torch::Tensor pointing_direction_spherical = torch::zeros({3,1}).to(torch::kFloat).to(DEVICE);
166     pointing_direction_spherical[0] = absolute_azimuth_of_transmitter;
167     pointing_direction_spherical[1] = absolute_elevation_of_transmitter;
168     pointing_direction_spherical[2] = torch::tensor({1}).to(torch::kFloat).to(DEVICE);
169     if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 60 \n";
170
171     this->pointing_direction = fSph2Cart(pointing_direction_spherical);
172     if(DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass: page 169 \n";
173 }
174
175 /*=====
176 Aim: Subsetting Scatterers inside the cone
177 .....
178 steps:
179 1. Find azimuth and range of all points.
180 2. Find 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)
189     std::cout<<"\t\t TransmitterClass: line 188 "<<std::endl;
190
191     // Finding spherical coordinates of scatterers and pointing direction
192     torch::Tensor scatterers_spherical = fCart2Sph(scatterers->coordinates);
193     if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 191 "<<std::endl;
194     torch::Tensor pointing_direction_spherical = fCart2Sph(this->pointing_direction);
195     if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 192 "<<std::endl;
196
197     // Calculating relative azimuths and radians
198     torch::Tensor relative_spherical = scatterers_spherical - pointing_direction_spherical;
199     if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 199 "<<std::endl;
200
201     // clearing some stuff up
202     scatterers_spherical.reset(); if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass: line 202
203     "<<std::endl;
204     pointing_direction_spherical.reset(); if(DEBUGMODE_TRANSMITTER) std::cout<<"\t\t TransmitterClass:
205     line 203 "<<std::endl;

```

```

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);
265         std::cout<<"\t\t TransmitterClass: line 267 "<<std::endl;
266
267     // finding the deltas
268     float delta_azimuth = this->azimuthal_beamwidth / numazimuthcells;
269     float delta_elevation = this->elevation_beamwidth / numelevationcells; std::cout<<"\t\t
270         TransmitterClass: line 271"<<std::endl;
271
272     // creating the centers (verified)
273     torch::Tensor azimuth_centers = torch::linspace(delta_azimuth/2, \
274         numazimuthcells * delta_azimuth - delta_azimuth/2, \
275         numazimuthcells);
276     torch::Tensor elevation_centers = torch::linspace(delta_elevation/2, \
277         numelevationcells * delta_elevation - delta_elevation/2, \
278         numelevationcells); std::cout<<"\t\t TransmitterClass:
279         line 279"<<std::endl;
280
281     // centering (verified)
282     azimuth_centers = azimuth_centers + torch::tensor({this->azimuthal_angle - \
283         (this->azimuthal_beamwidth/2)}).to(torch::kFloat);
284     elevation_centers = elevation_centers + torch::tensor({this->elevation_angle - \
285         (this->elevation_beamwidth/2)}).to(torch::kFloat);
286         std::cout<<"\t\t TransmitterClass: line
287         285"<<std::endl;
288
289     // building checkboxes
290     torch::Tensor checkbox = torch::zeros({numelevationcells, numazimuthcells}, torch::kBool);
291     torch::Tensor finalScatterBox = torch::zeros({numelevationcells, numazimuthcells,
292         3}).to(torch::kFloat);
293     torch::Tensor finalReflectivityBox = torch::zeros({numelevationcells,
294         numazimuthcells}).to(torch::kFloat); std::cout<<"\t\t TransmitterClass: line 290"<<std::endl;
295
296     // going through each-range-cell
297     for(int i = 0; i<(int)rangeBoundaries.numel(); ++i){
298         this->internal_subsetCurrentRangeCell(rangeBoundaries[i], \
299             scatterers, \
300             checkbox, \
301             finalScatterBox, \
302             finalReflectivityBox, \
303             azimuth_centers, \
304             elevation_centers, \
305             spherical_coordinates); std::cout<<"\t\t TransmitterClass: line
306             301"<<std::endl;
307
308         // after each-range-cell
309         torch::Tensor checkboxfilled = torch::sum(checkbox);
310         std::cout<<"\t\t\t\t checkbox-filled = "<<checkboxfilled.item<int>()<<"/"<<checkbox.numel()<<" |
311             percent = "<<100 * checkboxfilled.item<float>()/((float)checkbox.numel())<<std::endl;
312
313     }
314
315     // converting from box structure to [3, num-points] structure
316     torch::Tensor final_coords_spherical = \
317         torch::permute(finalScatterBox, {2, 0, 1}).reshape({3, (int)(finalScatterBox.numel()/3)});
318     torch::Tensor final_coords_cart = fSph2Cart(final_coords_spherical); std::cout<<"\t\t
319         TransmitterClass: line 308"<<std::endl;
320     std::cout<<"\t\t finalReflectivityBox.shape = "; fPrintTensorSize(finalReflectivityBox);
321     torch::Tensor final_reflectivity = finalReflectivityBox.reshape({finalReflectivityBox.numel()});
322         std::cout<<"\t\t TransmitterClass: line 310"<<std::endl;
323     torch::Tensor test_checkbox = checkbox.reshape({checkbox.numel()}); std::cout<<"\t\t TransmitterClass:
324         line 311"<<std::endl;
325
326     // just taking the points corresponding to the filled. Else, there's gonna be a lot of zero zero zero
327     tensors
328     auto mask = (test_checkbox == 1); std::cout<<"\t\t TransmitterClass: line 319"<<std::endl;
329     final_coords_cart = final_coords_cart.index({torch::indexing::Slice(), mask}); std::cout<<"\t\t
330         TransmitterClass: line 320"<<std::endl;
331     final_reflectivity = final_reflectivity.index({mask}); std::cout<<"\t\t TransmitterClass: line
332         321"<<std::endl;
333
334     // overwriting the scatterers
335     scatterers->coordinates = final_coords_cart;
336     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 =
351         scatterers->reflectivity.index({torch::indexing::Slice(), mask});
352     pointsincurrentrangeCell = spherical_coordinates.index({torch::indexing::Slice(),
353                                                            mask});
354
355     // finding number of azimuth sizes and what not
356     int numazimuthcells = azimuth_centers.numel();
357     int numelevationcells = elevation_centers.numel();
358
359     // go through all the combinations
360     for(int azi_index = 0; azi_index < numazimuthcells; ++azi_index){
361         for(int ele_index = 0; ele_index < numelevationcells; ++ele_index){
362
363             // check if this particular azimuth-elevation direction has been taken-care of.
364             if (checkbox[ele_index][azi_index].item<bool>()) break;
365
366             // init (verified)
367             torch::Tensor current_azimuth = azimuth_centers.index({azi_index});
368             torch::Tensor current_elevation = elevation_centers.index({ele_index});
369
370             // // finding azimuth boolean
371             // torch::Tensor azi_neighbours = torch::abs(pointsincurrentrangeCell[0] - current_azimuth);
372             // azi_neighbours = azi_neighbours <= delta_azimuth; // tinker with this.
373
374             // // finding elevation boolean
375             // torch::Tensor ele_neighbours = torch::abs(pointsincurrentrangeCell[1] - current_elevation);
376             // ele_neighbours = ele_neighbours <= delta_elevation;
377
378             // finding azimuth boolean
379             torch::Tensor azi_neighbours = torch::abs(pointsincurrentrangeCell[0] - current_azimuth);
380             azi_neighbours = azi_neighbours <= this->azimuthShadowThreshold; // tinker with
381             this.
382
383             // finding elevation boolean
384             torch::Tensor ele_neighbours = torch::abs(pointsincurrentrangeCell[1] - current_elevation);
385             ele_neighbours = ele_neighbours <= this->elevationShadowThreshold;
386
387             // combining booleans: means find all points that are within the limits of both the azimuth and
388             // boolean.
389             torch::Tensor neighbours_boolean = torch::mul(azi_neighbours, ele_neighbours);
390
391             // checking if there are any points along this direction
392             int num347 = torch::sum(neighbours_boolean).item<int>();
393             if (num347 == 0) continue;

```

```

391
392 // findings point along this direction
393 mask = (neighbours_boolean == 1);
394 torch::Tensor coords_along_aziele_spherical =
395     pointsincurrentrange.cell.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     \
403     index_where_min_range_is});
404
405 // filling the matrices up
406 finalScatterBox.index_put_({ele_index, azi_index, torch::indexing::Slice()}, \
407     closest_coord.reshape({1,1,3}));
408 finalReflectivityBox.index_put_({ele_index, azi_index}, \
409     closest_reflectivity);
410 checkbox.index_put_({ele_index, azi_index}, \
411     true);
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.

```

1  #include <cstdint>
2  #include <iostream>
3  #include <torch/torch.h>
4
5  #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"
6  #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fConvolveColumns.cpp"
7  #include "ScattererClass.h"
8  #include "TransmitterClass.h"
9
10 #pragma once
11
12 // hash defines
13 #ifndef PRINTSPACE
14 #define PRINTSPACE      std::cout<<"\n\n\n\n\n\n\n\n\n\n"<<std::endl;
15 #endif
16 #ifndef PRINTSMALLLINE
17 #define PRINTSMALLLINE  std::cout<<"-----"<<std::endl;
18 #endif
19 #ifndef PRINTLINE
20 #define PRINTLINE        std::cout<<"===== "<<std::endl;
21 #endif
22
23 #ifndef DEVICE
24     #define DEVICE        torch::kMPS
25     // #define DEVICE      torch::kCPU
26 #endif
27
28 #define PI                3.14159265
29
30 // #define DEBUG_ULA true
31 #define DEBUG_ULA false
32
33
34 class ULAClass{
35 public:
36     // intrinsic parameters
37     int num_sensors;           // number of sensors
38     float inter_element_spacing; // space between sensors
39     torch::Tensor coordinates;  // coordinates of each sensor
40     float sampling_frequency;   // sampling frequency of the sensors
41     float recording_period;     // recording period of the ULA
42     torch::Tensor location;     // location of first coordinate
43
44     // derived stuff
45     torch::Tensor sensorDirection;
46     torch::Tensor signalMatrix;
47
48     // constructor
49     ULAClass(int numsensors      = 32,
50             float inter_element_spacing = 1e-3,
51             torch::Tensor coordinates = torch::zeros({3, 2}),
52             float sampling_frequency = 48e3,
53             float recording_period   = 1):
54         num_sensors(numsensors),
55         inter_element_spacing(inter_element_spacing),
56         coordinates(coordinates),
57         sampling_frequency(sampling_frequency),
58         recording_period(recording_period) {
59         // calculating ULA direction
60         torch::Tensor sensorDirection = coordinates.slice(1, 0, 1) - coordinates.slice(1, 1, 2);
61
62         // normalizing
63         float normvalue = torch::linalg_norm(sensorDirection, 2, 0, true, torch::kFloat).item<float>();
64         if (normvalue != 0){
65             sensorDirection = sensorDirection / normvalue;
66         }

```

```

67
68         // copying direction
69         this->sensorDirection = sensorDirection;
70     }
71
72     // overriding printing
73     friend std::ostream& operator<<(std::ostream& os, ULAClass& ula){
74         os<<"\t number of sensors : "<<ula.num_sensors <<std::endl;
75         os<<"\t inter-element spacing: "<<ula.inter_element_spacing <<std::endl;
76         os<<"\t sensor-direction " <<torch::transpose(ula.sensorDirection, 0, 1)<<std::endl;
77         PRINTSMALLLINE
78         return os;
79     }
80
81     // overloading the "=" operator
82     ULAClass& operator=(const ULAClass& other){
83         // checking if copying to the same object
84         if(this == &other){
85             return *this;
86         }
87
88         // copying everything
89         this->num_sensors = other.num_sensors;
90         this->inter_element_spacing = other.inter_element_spacing;
91         this->coordinates = other.coordinates.clone();
92         this->sampling_frequency = other.sampling_frequency;
93         this->recording_period = other.recording_period;
94         this->sensorDirection = other.sensorDirection.clone();
95
96         // returning
97         return *this;
98     }
99
100     /* =====
101     Aim: Build coordinates on top of location.
102     .....
103     Note:
104         > This function builds the location of the coordinates based on the location and direction member.
105     -----*/
106     void buildCoordinatesBasedOnLocation(){
107
108         // length-normalize the sensor-direction
109         this->sensorDirection = torch::div(this->sensorDirection, torch::linalg_norm(this->sensorDirection, \
110                                                                                     2, 0, true, \
111                                                                                     torch::kFloat));
112
113         if(DEBUG_ULA) std::cout<<"\t ULAClass: line 105 \n";
114
115         // multiply with inter-element distance
116         this->sensorDirection = this->sensorDirection * this->inter_element_spacing;
117         this->sensorDirection = this->sensorDirection.reshape({this->sensorDirection.numel(), 1});
118         if(DEBUG_ULA) std::cout<<"\t ULAClass: line 110 \n";
119
120         // create integer-array
121         // torch::Tensor integer_array = torch::linspace(0, \
122         //                                                                                     this->num_sensors-1, \
123         //                                                                                     this->num_sensors).reshape({1,
124         //                                                                                     this->num_sensors}).to(torch::kFloat);
125         torch::Tensor integer_array = torch::linspace(0, \
126                                                                                     this->num_sensors-1, \
127                                                                                     this->num_sensors).reshape({1, this->num_sensors});
128
129         std::cout<<"integer_array = "; fPrintTensorSize(integer_array);
130         if(DEBUG_ULA) std::cout<<"\t ULAClass: line 116 \n";
131
132         // this->coordinates = torch::mul(torch::tile(integer_array, {3, 1}).to(torch::kFloat), \
133         //                                                                                     torch::tile(this->sensorDirection, {1,
134         //                                                                                     this->num_sensors}).to(torch::kFloat));
135         torch::Tensor test = torch::mul(torch::tile(integer_array, {3, 1}).to(torch::kFloat), \
136                                                                                     torch::tile(this->sensorDirection, {1,
137                                                                                     this->num_sensors}).to(torch::kFloat));
138
139         this->coordinates = this->location + test;
140         if(DEBUG_ULA) std::cout<<"\t ULAClass: line 120 \n";
141     }

```

```

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

```

```

209         this->signalMatrix.index_put_({where_to_place, sensor_index}, \
210                                     this->signalMatrix.index({where_to_place, sensor_index}) + \
211                                     scatterers->reflectivity.index({0, scatter_index}) );
212     }
213 }
214
215 // Convolving signals with transmitted signal
216 torch::Tensor signalTensorAsArgument = \
217     transmitterObj->Signal.reshape({transmitterObj->Signal.numel(),1});
218 signalTensorAsArgument = torch::tile(signalTensorAsArgument, \
219                                     {1, this->signalMatrix.size(1)});
220
221 // convolving the pulse-matrix with the signal matrix
222 fConvolveColumns(this->signalMatrix, \
223                 signalTensorAsArgument);
224
225 // trimming the convolved signal since the signal matrix length remains the same
226 this->signalMatrix = this->signalMatrix.index({torch::indexing::Slice(0, numsamples), \
227                                               torch::indexing::Slice()});
228
229 // printing the shape
230 std::cout<<"\t\t\t this->signalMatrix.shape (after signal sim) = ";
231     fPrintTensorSize(this->signalMatrix);
232
233 return;
234 }
235 };

```

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\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 // saving coordinates
41 torch::save(scatterer_fls.coordinates, \
42     "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates.pt");
43 torch::save(scatterer_port.coordinates, \
44     "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates.pt");
45 torch::save(scatterer_starboard.coordinates, \
46     "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates.pt");
47 
48 // saving reflectivities
49 torch::save(scatterer_fls.reflectivity, \
50     "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates_reflectivity.pt");
51 torch::save(scatterer_port.reflectivity, \
52     "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates_reflectivity.pt");
53 torch::save(scatterer_starboard.reflectivity, \
54     "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates_reflectivity.pt")
55 }
56 
57 // plotting tensors
58 fPlotTensors();
59 
60 // indicating end of thread
61 std::cout<<"\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 direction for
        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 =
216         torch::tensor({target_azimuth_deg}).to(torch::kFloat).to(DEVICE) - \
217         pointing_direction_spherical[0];
218     torch::Tensor azimuth_correction_radians = azimuth_correction * PI / 180;
219
220     torch::Tensor yawCorrectionMatrix = \
221         torch::tensor({torch::cos(azimuth_correction_radians).item<float>(), \
222                       torch::cos(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +
223                               azimuth_correction_radians).item<float>(), \
224                       (float)0, \
225                       torch::sin(azimuth_correction_radians).item<float>(), \
226                       torch::sin(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +
227                               azimuth_correction_radians).item<float>(), \
228                       (float)0, \
229                       (float)0, \
230                       (float)0, \
231                       (float)1}).reshape({3,3}).to(torch::kFloat).to(DEVICE);
232
233     // returning the matrix
234     return yawCorrectionMatrix;
235 }
236
237 // pitch-correction matrix
238 torch::Tensor createPitchCorrectionMatrix(torch::Tensor pointing_direction_spherical, \
239                                         float target_elevation_deg){
240
241     // building parameters
242     torch::Tensor elevation_correction =
243         torch::tensor({target_elevation_deg}).to(torch::kFloat).to(DEVICE) - \
244         pointing_direction_spherical[1];
245     torch::Tensor elevation_correction_radians = elevation_correction * PI / 180;
246
247     // creating the matrix
248     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) = ";
270     fPrintTensorSize(scatterer_fls.coordinates);
271     std::cout<<"\t AUVClass: scatterer_port.coordinates.shape (before) = ";
272     fPrintTensorSize(scatterer_port.coordinates);
273     std::cout<<"\t AUVClass: scatterer_starboard.coordinates.shape (before) = ";
274     fPrintTensorSize(scatterer_starboard.coordinates);
275
276     // finding the pointing direction in spherical
277     torch::Tensor auv_pointing_direction_spherical = fCart2Sph(this->pointing_direction);
278
279     // asking the transmitters to subset the scatterers by multithreading
280     std::thread transmitterFLSSubset_t(&AUVClass::subsetScatterers, this, \
281                                     &scatterer_fls, \
282                                     &this->transmitter_fls, \
283                                     (float)0);
284     std::thread transmitterPortSubset_t(&AUVClass::subsetScatterers, this, \
285                                     &scatterer_port, \
286                                     &this->transmitter_port, \
287                                     - auv_pointing_direction_spherical[1].item<float>());
288     std::thread transmitterStarboardSubset_t(&AUVClass::subsetScatterers, this, \
289                                     &scatterer_starboard, \
290                                     &this->transmitter_starboard, \
291                                     auv_pointing_direction_spherical[1].item<float>());
292
293     // joining the subset threads back
294     transmitterFLSSubset_t.join(); transmitterPortSubset_t.join(); transmitterStarboardSubset_t.join();
295
296     // printing the size of these points before subsetting
297     PRINTDOTS
298     std::cout<<"\t AUVClass: scatterer_fls.coordinates.shape (after) = ";
299     fPrintTensorSize(scatterer_fls.coordinates);
300     std::cout<<"\t AUVClass: scatterer_port.coordinates.shape (after) = ";
301     fPrintTensorSize(scatterer_port.coordinates);
302     std::cout<<"\t AUVClass: scatterer_starboard.coordinates.shape (after) = ";
303     fPrintTensorSize(scatterer_starboard.coordinates);
304
305     // // multithreading the saving tensors part.
306     // std::thread savetensor_t(fSaveSeafloorScatterers, \
307     //                          scatterer, \
308     //                          scatterer_fls, \
309     //                          scatterer_port, \
310     //                          scatterer_starboard);
311     // savetensor_t.detach();
312
313     // saving the tensors
314     if (true) {
315         // saving the ground-truth

```



```

310     ScattererClass SeafloorScatter_gt = scatterer;
311     torch::save(SeafloorScatter_gt.coordinates,
312                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_gt.pt");
313
314     // saving coordinates
315     torch::save(scatterer_fls.coordinates,
316                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates.pt");
317     torch::save(scatterer_port.coordinates,
318                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates.pt");
319     torch::save(scatterer_starboard.coordinates,
320                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates.pt");
321
322     // saving reflectivities
323     torch::save(scatterer_fls.reflectivity,
324                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_fls_coordinates_reflectivity.pt");
325     torch::save(scatterer_port.reflectivity,
326                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_port_coordinates_reflectivity.pt");
327     torch::save(scatterer_starboard.reflectivity,
328                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/SeafloorScatter_starboard.coordinates_reflectivity.pt");
329
330     // plotting tensors
331     fPlotTensors();
332 }
333
334 // asking ULAs to simulate signal through multithreading
335 std::thread ulafns_signalsim_t(&ULAClass::nfdc_simulateSignal, \
336                                &this->ULA_fls, \
337                                &scatterer_fls, \
338                                &this->transmitter_fls);
339 std::thread ulaport_signalsim_t(&ULAClass::nfdc_simulateSignal, \
340                                &this->ULA_port, \
341                                &scatterer_port, \
342                                &this->transmitter_port);
343 std::thread ulastarboard_signalsim_t(&ULAClass::nfdc_simulateSignal, \
344                                      &this->ULA_starboard, \
345                                      &scatterer_starboard, \
346                                      &this->transmitter_starboard);
347
348 // joining them back
349 ulafns_signalsim_t.join();
350 ulaport_signalsim_t.join();
351 ulastarboard_signalsim_t.join();
352
353 // saving the tensors
354 if (true) {
355     // saving the ground-truth
356     torch::save(this->ULA_fls.signalMatrix,
357                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/signalMatrix_fls.pt");
358     torch::save(this->ULA_port.signalMatrix,
359                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/signalMatrix_port.pt");
360     torch::save(this->ULA_starboard.signalMatrix,
361                 "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Assets/signalMatrix_starboard.pt");
362 }
363 }
364 };

```

8.2 Setup Scripts

8.2.1 Seafloor Setup

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

```

1  /* =====
2  Aim: Setup sea floor
3  =====*/
4  #include <torch/torch.h>
5  #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"
6
7  #ifndef DEVICE
8      // #define DEVICE      torch::kMPS
9      #define DEVICE      torch::kCPU
10 #endif
11
12
13 // adding terrain features
14 #define BOXES      true
15 #define TERRAIN      false
16 #define DEBUG_SEAFLOOR false
17
18
19
20 // Adding boxes
21 void fCreateBoxes(float across_track_length, \
22                  float along_track_length, \
23                  torch::Tensor& box_coordinates, \
24                  torch::Tensor& box_reflectivity){
25
26     // converting arguments to torch tensors
27
28     // setting up parameters
29     float min_width      = 2;      // minimum across-track dimension of the boxes in the sea-floor
30     float max_width      = 5;      // maximum across-track dimension of the boxes in the sea-floor
31
32     float min_length     = 2;      // minimum along-track dimension of the boxes in the sea-floor
33     float max_length     = 5;      // maximum along-track dimension of the boxes in the sea-floor
34
35     float min_height     = 3;      // minimum height of the boxes in the sea-floor
36     float max_height     = 20;     // maximum height of the boxes in the sea-floor
37
38     int meshdensity      = 10;     // number of points per meter.
39     float meshreflectivity = 2;     // average reflectivity of the mesh
40
41     int num_boxes        = 10;     // number of boxes in the sea-floor
42     if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 41\n";
43
44     // finding center point
45     torch::Tensor midxypoints = torch::rand({3, num_boxes}).to(torch::kFloat).to(DEVICE);
46     midxypoints[0]            = midxypoints[0] * across_track_length;
47     midxypoints[1]            = midxypoints[1] * along_track_length;
48     midxypoints[2]            = 0;
49     if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 48\n";
50
51     // assigning dimensions to boxes
52     torch::Tensor boxwidths = torch::rand({num_boxes})*(max_width - min_width) + min_width; // assigning
53     // widths to each boxes
54     torch::Tensor boxlengths = torch::rand({num_boxes})*(max_length - min_length) + min_length; // assigning
55     // lengths to each boxes
56     torch::Tensor boxheights = torch::rand({num_boxes})*(max_height - min_height) + min_height; // assigning
57     // heights to each boxes
58     if(DEBUG_SEAFLOOR) std::cout<<"\t fCreateBoxes: line 54\n";
59
60     // creating mesh for each box
61     for(int i = 0; i<num_boxes; ++i){
62
63         // finding x-points
64         torch::Tensor xpoints = torch::linspace(-boxwidths[i].item<float>()/2, \
65         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 scatterers
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 = ";
169                     fPrintTensorSize(scatterers->coordinates);}
170 if(DEBUG_SEAFLOOR) {std::cout<<"\t box_coordinates.shape = "; fPrintTensorSize(box_coordinates);}
171 if(DEBUG_SEAFLOOR) {std::cout<<"\t scatterers->reflectivity.shape = ";
172                     fPrintTensorSize(scatterers->reflectivity);}
173 if(DEBUG_SEAFLOOR) {std::cout<<"\t box_reflectivity = "; fPrintTensorSize(box_reflectivity);}
174
175 scatterers->coordinates = torch::cat({scatterers->coordinates, box_coordinates}, 1);
176 PRINTLINE
177 scatterers->reflectivity = torch::cat({scatterers->reflectivity, box_reflectivity}, 1);
178 PRINTSMALLLINE
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) {

```

```

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
55 // along azimuth (used for shadowing)
56 int elevationQuantDensity = 10; // number of points, a degree is split into quantization density
57 // along elevation (used for shadowing)
58 float rangeQuantSize = 10; // the length of a cell (used for shadowing)
59
60 float azimuthShadowThreshold = 1; // azimuth threshold (in degrees)
61 float elevationShadowThreshold = 1; // elevation threshold (in degrees)
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 = 0.1; // 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  // assigning values
67  ula_starboard->num_sensors      = num_sensors;          // assigning number of sensors
68  ula_starboard->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
69  ula_starboard->coordinates      = ULA_coordinates;      // assigning ULA coordinates
70  ula_starboard->sampling_frequency = sampling_frequency;  // assigning sampling frequencys
71  ula_starboard->recording_period  = recording_period;    // assigning recording period
72  ula_starboard->sensorDirection   = ULA_direction;       // ULA direction
73
74  ula_starboard->num_sensors      = num_sensors;          // assigning number of sensors
75  ula_starboard->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
76  ula_starboard->coordinates      = ULA_coordinates;      // assigning ULA coordinates
77  ula_starboard->sampling_frequency = sampling_frequency;  // assigning sampling frequencys
78  ula_starboard->recording_period  = recording_period;    // assigning recording period
79  ula_starboard->sensorDirection   = ULA_direction;       // ULA direction
80
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

```



```
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); //
17     // starting location of AUV
18     torch::Tensor velocity      = torch::tensor({5,0, 0}).reshape({3,1}).to(torch::kFloat).to(DEVICE); //
19     // starting velocity of AUV
20     torch::Tensor pointing_direction = torch::tensor({1,0, 0}).reshape({3,1}).to(torch::kFloat).to(DEVICE);
21     // pointing direction of AUV
22
23     // assigning
24     auv->location      = location;          // assigning location of auv
25     auv->velocity      = velocity;          // assigning vector representing velocity
26     auv->pointing_direction = pointing_direction; // assigning pointing direction of auv
27 }
```

8.3 Function Definitions

8.3.1 Cartesian Coordinates to Spherical Coordinates

```

1  /* =====
2  Aim: Setup sea floor
3  =====*/
4  #include <torch/torch.h>
5  #include <iostream>
6
7  // hash-defines
8  #define PI          3.14159265
9  #define DEBUG_Cart2Sph false
10
11 #ifndef DEVICE
12     #define DEVICE      torch::kMPS
13     // #define DEVICE    torch::kCPU
14 #endif
15
16
17 // bringing in functions
18 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"
19
20 #pragma once
21
22 torch::Tensor fCart2Sph(torch::Tensor cartesian_vector){
23
24     // sending argument to the device
25     cartesian_vector = cartesian_vector.to(DEVICE);
26     if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 26 \n";
27
28     // splatting the point onto xy plane
29     torch::Tensor xysplat = cartesian_vector.clone().to(DEVICE);
30     xysplat[2] = 0;
31     if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 31 \n";
32
33     // finding splat lengths
34     torch::Tensor xysplat_lengths = torch::linalg_norm(xysplat, 2, 0, true, torch::kFloat).to(DEVICE);
35     if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 35 \n";
36
37     // finding azimuthal and elevation angles
38     torch::Tensor azimuthal_angles = torch::atan2(xysplat[1], xysplat[0]).to(DEVICE) * 180/PI;
39     azimuthal_angles = azimuthal_angles.reshape({1, azimuthal_angles.numel()});
40     torch::Tensor elevation_angles = torch::atan2(cartesian_vector[2], xysplat_lengths).to(DEVICE) * 180/PI;
41     torch::Tensor rho_values = torch::linalg_norm(cartesian_vector, 2, 0, true, torch::kFloat).to(DEVICE);
42     if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 42 \n";
43
44
45     // printing values for debugging
46     if (DEBUG_Cart2Sph){
47         std::cout<<"azimuthal_angles.shape = "; fPrintTensorSize(azimuthal_angles);
48         std::cout<<"elevation_angles.shape = "; fPrintTensorSize(elevation_angles);
49         std::cout<<"rho_values.shape = "; fPrintTensorSize(rho_values);
50     }
51     if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 51 \n";
52
53     // creating tensor to send back
54     torch::Tensor spherical_vector = torch::cat({azimuthal_angles, \
55                                                  elevation_angles, \
56                                                  rho_values}, 0).to(DEVICE);
57     if (DEBUG_Cart2Sph) std::cout<<"\t fCart2Sph: line 57 \n";
58
59     // returning the value
60     return spherical_vector;
61 }

```

Chapter 9

Reading

9.1 Primary Books

- 1.

9.2 Interesting Papers