

Autonomous Underwater Vehicle: A Surveillance Protocol

S.V. Rajendran

January 28, 2025

Contents

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	16
8.1.4	Class: Autonomous Underwater Vehicle	18
8.2	Setup Scripts	22
8.2.1	Seafloor Setup	22
8.2.2	Transmitter Setup	23
8.2.3	Uniform Linear Array	25
8.2.4	AUV Setup	26
8.3	Function Definitions	27

9	Reading	28
9.1	Primary Books	28
9.2	Interesting Papers	28

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
5 // Including classes
6 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"
7
8 // Including functions
9 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fCart2Sph.cpp"
10 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/Functions/fPrintTensorSize.cpp"
11
12 #pragma once
13
14 // hash defines
15 #ifndef PRINTSPACE
16 # define PRINTSPACE      std::cout<<"\n\n\n\n\n\n\n\n"<<std::endl;
17 #endif
18 #ifndef PRINTSMALLLINE
19 # define PRINTSMALLLINE std::cout<<"-----"<<std::endl;
20 #endif
21 #ifndef PRINTLINE
22 # define PRINTLINE      std::cout<<"===== "<<std::endl;
23 #endif
24
25 #define PI              3.14159265
26 #define DEBUGMODE_TRANSMITTER  false
27
28 #ifndef DEVICE
29 #define DEVICE          torch::kMPS
30 // #define DEVICE        torch::kCPU
31 #endif
32
33
34 class TransmitterClass{
35 public:
36
37     // physical/intrinsic properties
38     torch::Tensor location; // location tensor
39     torch::Tensor pointing_direction; // pointing direction
40
41     // basic parameters
42     torch::Tensor Signal; // transmitted signal (LFM)
43     float azimuthal_angle; // transmitter's azimuthal pointing direction
44     float elevation_angle; // transmitter's elevation pointing direction
45     float azimuthal_beamwidth; // azimuthal beamwidth of transmitter
46     float elevation_beamwidth; // elevation beamwidth of transmitter
47     float range; // a parameter used for spotlight mode.
48
49     // transmitted signal attributes
50     float f_low; // lowest frequency of LFM
51     float f_high; // highest frequency of LFM
52     float fc; // center frequency of LFM
53     float bandwidth; // bandwidth of LFM
54
55     // shadowing properties
56     int azimuthQuantDensity; // quantization of angles along the azimuth
57     int elevationQuantDensity; // quantization of angles along the elevation
58     float rangeQuantSize; // range-cell size when shadowing
59     float azimuthShadowThreshold; // azimuth thresholding
60     float elevationShadowThreshold; // elevation thresholding
61     torch::Tensor checkBox; // box indicating whether a scatter for a range-angle pair has been found
62     torch::Tensor finalScatterBox; // a 3D tensor where the third dimension represents the vector length
63     torch::Tensor finalReflectivityBox; // to store the reflectivity
64
65
66

```



```

67 // Constructor
68 TransmitterClass(torch::Tensor location = torch::zeros({3,1}),
69                 torch::Tensor Signal    = torch::zeros({10,1}),
70                 float azimuthal_angle   = 0,
71                 float elevation_angle    = -30,
72                 float azimuthal_beamwidth = 30,
73                 float elevation_beamwidth = 30):
74     location(location),
75     Signal(Signal),
76     azimuthal_angle(azimuthal_angle),
77     elevation_angle(elevation_angle),
78     azimuthal_beamwidth(azimuthal_beamwidth),
79     elevation_beamwidth(elevation_beamwidth) {}
80
81 // overloading output
82 friend std::ostream& operator<<(std::ostream& os, TransmitterClass& transmitter){
83     os<<"\t> azimuth      : "<<transmitter.azimuthal_angle <<std::endl;
84     os<<"\t> elevation    : "<<transmitter.elevation_angle <<std::endl;
85     os<<"\t> azimuthal beamwidth: "<<transmitter.azimuthal_beamwidth<<std::endl;
86     os<<"\t> elevation beamwidth: "<<transmitter.elevation_beamwidth<<std::endl;
87     PRINTSMALLLINE
88     return os;
89 }
90
91 // overloading copyign operator
92 TransmitterClass& operator=(const TransmitterClass& other){
93
94     // checking self-assignment
95     if(this==&other){
96         return *this;
97     }
98
99     // allocating memory
100     this->location      = other.location;
101     this->Signal        = other.Signal;
102     this->azimuthal_angle = other.azimuthal_angle;
103     this->elevation_angle = other.elevation_angle;
104     this->azimuthal_beamwidth = other.azimuthal_beamwidth;
105     this->elevation_beamwidth = other.elevation_beamwidth;
106     this->range         = other.range;
107
108     // transmitted signal attributes
109     this->f_low         = other.f_low;
110     this->f_high        = other.f_high;
111     this->fc            = other.fc;
112     this->bandwidth     = other.bandwidth;
113
114     // shadowing properties
115     this->azimuthQuantDensity = other.azimuthQuantDensity;
116     this->elevationQuantDensity = other.elevationQuantDensity;
117     this->rangeQuantSize      = other.rangeQuantSize;
118     this->azimuthShadowThreshold = other.azimuthShadowThreshold;
119     this->elevationShadowThreshold = other.elevationShadowThreshold;
120     this->checkbox              = other.checkbox;
121     this->finalScatterBox     = other.finalScatterBox;
122     this->finalReflectivityBox = other.finalReflectivityBox;
123
124     // returning
125     return *this;
126
127 };
128
129 // subsetting scatterers
130 void subsetScatterers(ScattererClass* scatterers){
131
132
133     if (DEBUGMODE_TRANSMITTER) {
134         std::cout<<"scatterers->coordinates.shape      = ";
135         fPrintTensorSize(scatterers->coordinates);
136     }
137
138
139     // converting from cartesian tensors to spherical tensors

```

```

140 torch::Tensor scatterers_spherical = fCart2Sph(scatterers->coordinates);
141 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass::subsetScatterers 140 \n";
142 scatterers_spherical = scatterers_spherical.to(DEVICE);
143 if (DEBUGMODE_TRANSMITTER){
144     std::cout<<"scatterers_spherical.shape          = ";
145     fPrintTensorSize(scatterers_spherical); PRINTSMALLLINE
146 }
147
148 // printing some status
149 if (DEBUGMODE_TRANSMITTER){
150     PRINTSPACE
151     PRINTLINE
152     std::cout<<"\t TransmitterClass > this->azimuthal_angle = " <<this->azimuthal_angle <<std::endl;
153     std::cout<<"\t TransmitterClass > this->elevation_angle = " <<this->elevation_angle <<std::endl;
154     std::cout<<"\t TransmitterClass > this->azimuthal_beamwidth = " <<this->azimuthal_beamwidth
155         <<std::endl;
156     std::cout<<"\t TransmitterClass > this->elevation_beamwidth = " <<this->elevation_beamwidth
157         <<std::endl;
158     PRINTLINE
159     PRINTSPACE
160 }
161
162 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass >reached line 136 \n";
163
164 // finding points that are in the cone
165 torch::Tensor scatter_boolean = \
166     (torch::square((scatterers_spherical[0] - \
167         torch::tensor({this->azimuthal_angle}).to(torch::kFloat).to(DEVICE))))/torch::square(torch::tensor({this->
168     + \
169     torch::square((scatterers_spherical[1] - \
170         torch::tensor({this->elevation_angle}).to(torch::kFloat).to(DEVICE))))/torch::square(torch::tensor({this->
171         \
172     < 1);
173
174 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass >reached line 141 \n";
175 if (DEBUGMODE_TRANSMITTER){
176     std::cout<<"scatter_boolean.shape          = ";
177     fPrintTensorSize(scatter_boolean);
178     PRINTSMALLLINE;
179 }
180
181 // subsetting points within the elliptical beam
182 auto mask = (scatter_boolean == 1); // creating a mask
183 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass >reached line 146 \n";
184 scatterers->coordinates = scatterers->coordinates.index({torch::indexing::Slice(), mask});
185 if (DEBUGMODE_TRANSMITTER) {
186     std::cout<<"scatterers->coordinates.shape          = ";
187     fPrintTensorSize(scatterers->coordinates); PRINTSMALLLINE;
188 }
189 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass >reached line 148 \n";
190 scatterers->reflectivity = scatterers->reflectivity.index({torch::indexing::Slice(), mask});
191 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass >reached line 150 \n";
192
193 // this is where histogram shadowing comes in (later)
194
195 if (DEBUGMODE_TRANSMITTER) std::cout<<"\t TransmitterClass > reached line 156 \n";
196
197 }
198
199 };

```

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

```

```
67     PRINTSMALLLINE
68     return os;
69 }
70
71 // overloading the "=" operator
72 ULAClass& operator=(const ULAClass& other){
73     // checking if copying to the same object
74     if(this == &other){
75         return *this;
76     }
77
78     // copying everything
79     this->num_sensors      = other.num_sensors;
80     this->inter_element_spacing = other.inter_element_spacing;
81     this->coordinates      = other.coordinates.clone();
82     this->sampling_frequency = other.sampling_frequency;
83     this->recording_period  = other.recording_period;
84     this->sensorDirection  = other.sensorDirection.clone();
85
86     // returning
87     return *this;
88 }
89
90 };
```

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 "TransmitterClass.h"
2  #include "ULAClass.h"
3  #include <iostream>
4  #include <ostream>
5  #include <torch/torch.h>
6  #include <cmath>
7
8  #pragma once
9
10 // including class-definitions
11 #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"
12
13 // hash defines
14 #ifndef PRINTSPACE
15 #define PRINTSPACE    std::cout<<"\n\n\n\n\n\n\n\n\n\n"<<std::endl;
16 #endif
17 #ifndef PRINTSMALLLINE
18 #define PRINTSMALLLINE std::cout<<"-----"<<std::endl;
19 #endif
20 #ifndef PRINTLINE
21 #define PRINTLINE    std::cout<<"===== "<<std::endl;
22 #endif
23
24 #ifndef DEVICE
25 #define DEVICE    torch::kMPS
26 // #define DEVICE    torch::kCPU
27 #endif
28
29
30
31
32 // #define PRINTSPACE    std::cout<<"\n\n\n\n\n\n\n\n\n\n"<<std::endl;
33 // #define PRINTSMALLLINE std::cout<<"-----"<<std::endl;
34 // #define PRINTLINE    std::cout<<"===== "<<std::endl;
35 #define PI    3.14159265
36 #define DEBUGMODE_AUV false
37
38
39 class AUVClass{
40 public:
41     // Intrinsic attributes
42     torch::Tensor location;    // location of vessel
43     torch::Tensor velocity;    // current speed of the vessel [a vector]
44     torch::Tensor acceleration;    // current acceleration of vessel [a vector]
45     torch::Tensor pointing_direction;    // direction to which the AUV is pointed
46
47     // uniform linear-arrays
48     ULAClass ULA_fls;    // front-looking SONAR ULA
49     ULAClass ULA_port;    // mounted ULA [object of class, ULAClass]
50     ULAClass ULA_starboard;    // mounted ULA [object of class, ULAClass]
51
52     // transmitters
53     TransmitterClass transmitter_fls;    // transmitter for front-looking SONAR
54     TransmitterClass transmitter_port;    // mounted transmitter [obj of class, TransmitterClass]
55     TransmitterClass transmitter_starboard;    // mounted transmitter [obj of class, TransmitterClass]
56
57     // derived or dependent attributes
58     torch::Tensor signalMatrix_1;    // matrix containing the signals obtained from ULA_1
59     torch::Tensor largeSignalMatrix_1;    // matrix holding signal of synthetic aperture
60     torch::Tensor beamformedLargeSignalMatrix;    // each column is the beamformed signal at each stop-hop
61
62     // plotting mode
63     bool plottingmode;    // to suppress plotting associated with classes
64
65     // spotlight mode related
66     torch::Tensor absolute_coords_patch_cart;    // cartesian coordinates of patch

```

```

67
68 // Synthetic Aperture Related
69 torch::Tensor ApertureSensorLocations; // sensor locations of aperture
70
71
72 /*
73 =====
74 Aim: stepping motion
75 -----
76 */
77 void step(float timestep){
78
79     // updating location
80     this->location = this->location + this->velocity * timestep;
81
82     // updating attributes of members
83     this->updateAttributes();
84 }
85
86 /*
87 =====
88 Aim: updateAttributes
89 -----
90 */
91 void updateAttributes(){
92
93     // updating coordinates of sensors
94     this->ULA_fls.coordinates = this->ULA_fls.coordinates + this->location;
95     this->ULA_port.coordinates = this->ULA_port.coordinates + this->location;
96     this->ULA_starboard.coordinates = this->ULA_starboard.coordinates + this->location;
97
98     // updating transmitter locations
99     this->transmitter_fls = this->location;
100    this->transmitter_port = this->location;
101    this->transmitter_starboard = this->location;
102 }
103
104
105
106 /*
107 =====
108 Aim: operator overriding for printing
109 -----
110 */
111 friend std::ostream& operator<<(std::ostream& os, AUVClass &auv){
112     os<<"\t location = "<<torch::transpose(auv.location, 0, 1)<<std::endl;
113     os<<"\t velocity = "<<torch::transpose(auv.velocity, 0, 1)<<std::endl;
114     return os;
115 }
116
117
118 /*
119 =====
120 Aim: Changing Basis
121 Note:
122     - The subset-function in the transmitter class assumes the subsetting with the current basis.
123     - However, this is not ideal since we want the subsetting to be with respect to the AUV.
124     - So this function essentially changes the coordinates of the scatterers to that of the AUV's location
125       and pointing direction.
126     - For now, we make the assumption that our AUV doesn't roll. That is, its belly is always to the
127       sea-floor.
128     - we apply to the floor-scatterer coordinates, the very operations we need to apply to the pointing
129       vector to make it point in the y-direction.
130     - not every operation works though cause the determinant should be one to ensure only rotations and no
131       compressions take place.
132     - so we're gonna have to combine three transformations: yaw corrections and pitch correction.
133 -----
134 */
135 void subsetScatterers(ScattererClass* scatterers,\
136                      TransmitterClass* transmitterObj){
137
138     // first, translate based on the AUV's current location
139     if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 83"<<std::endl;

```

```

136 scatterers->coordinates = scatterers->coordinates - this->location;
137
138 // find the azimuth and elevation of pointing-vector
139 torch::Tensor pointing_direction_spherical = fCart2Sph(this->pointing_direction);
140 pointing_direction_spherical = pointing_direction_spherical.to(DEVICE);
141 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 88"<<std::endl;
142
143 // transforming the matrix accordingly
144 torch::Tensor yawCorrectionMatrix = createYawCorrectionMatrix(pointing_direction_spherical, 90);
145 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 144"<<std::endl;
146 torch::Tensor pitchCorrectionMatrix = createPitchCorrectionMatrix(pointing_direction_spherical, 0);
147 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 139"<<std::endl;
148
149 // sending both to the right device
150 yawCorrectionMatrix = yawCorrectionMatrix.to(DEVICE);
151 pitchCorrectionMatrix = pitchCorrectionMatrix.to(DEVICE);
152
153
154 // combine the two to minimize MIPS
155 torch::Tensor PitchYawCorrectionMatrix = torch::matmul(yawCorrectionMatrix, \
156                                                         pitchCorrectionMatrix).to(DEVICE);
157 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 145"<<std::endl;
158
159 // multiply the two with the coordinates to change the coordinates.
160 scatterers->coordinates = torch::matmul(PitchYawCorrectionMatrix, \
161                                         scatterers->coordinates);
162 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 150"<<std::endl;
163
164 // calling the method associated with the transmitter
165 transmitterObj->subsetScatterers(scatterers);
166 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 154"<<std::endl;
167
168 // de-correcting relative rotation-transformations
169 yawCorrectionMatrix = createYawCorrectionMatrix(pointing_direction_spherical, \
170                                                 pointing_direction_spherical[0].item<float>());
171 pitchCorrectionMatrix = createPitchCorrectionMatrix(pointing_direction_spherical, \
172                                                     pointing_direction_spherical[1].item<float>());
173 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 161"<<std::endl;
174
175 // combine the two to minimize MIPS
176 PitchYawCorrectionMatrix = torch::matmul(yawCorrectionMatrix, \
177                                         pitchCorrectionMatrix).to(DEVICE);
178 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 166"<<std::endl;
179
180 // multiply the two with the coordinates to change the coordinates.
181 scatterers->coordinates = torch::matmul(PitchYawCorrectionMatrix, \
182                                         scatterers->coordinates);
183 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 171"<<std::endl;
184
185 // de-correcting relative translational transformation
186 scatterers->coordinates = scatterers->coordinates + this->location;
187 if(DEBUGMODE_AUV) std::cout<<"\t AUV: line 175"<<std::endl;
188
189 }
190
191
192 // pitch-correction matrix
193 torch::Tensor createYawCorrectionMatrix(torch::Tensor pointing_direction_spherical, \
194                                       float target_azimuth_deg){
195
196     // building parameters
197     torch::Tensor azimuth_correction =
198         torch::tensor({target_azimuth_deg}).to(torch::kFloat).to(DEVICE) - \
199         pointing_direction_spherical[0];
200     torch::Tensor azimuth_correction_radians = azimuth_correction * PI / 180;
201
202     torch::Tensor yawCorrectionMatrix = \
203         torch::tensor({torch::cos(azimuth_correction_radians).item<float>(), \
204                       torch::cos(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 + \
205                               azimuth_correction_radians).item<float>(), \
206                       (float)0, \
207                       torch::sin(azimuth_correction_radians).item<float>(), \
208                       torch::sin(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 +

```

```

207         azimuth_correction_radians).item<float>(), \
208         (float)0, \
209         (float)0, \
210         (float)1}).reshape({3,3}).to(torch::kFloat).to(DEVICE);
211
212     // returning the matrix
213     return yawCorrectionMatrix;
214 }
215
216 // pitch-correction matrix
217 torch::Tensor createPitchCorrectionMatrix(torch::Tensor pointing_direction_spherical, \
218         float target_elevation_deg){
219
220     // building parameters
221     torch::Tensor elevation_correction =
222         torch::tensor({target_elevation_deg}).to(torch::kFloat).to(DEVICE) - \
223         pointing_direction_spherical[1];
224     torch::Tensor elevation_correction_radians = elevation_correction * PI / 180;
225
226     // creating the matrix
227     torch::Tensor pitchCorrectionMatrix = \
228         torch::tensor({(float)1, \
229             (float)0, \
230             (float)0, \
231             (float)0, \
232             torch::cos(elevation_correction_radians).item<float>(), \
233             torch::cos(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 + \
234                 elevation_correction_radians).item<float>(), \
235             (float)0, \
236             torch::sin(elevation_correction_radians).item<float>(), \
237             torch::sin(torch::tensor({90}).to(torch::kFloat).to(DEVICE)*PI/180 + \
238                 elevation_correction_radians).item<float>())}).reshape({3,3}).to(torch::kFloat);
239
240     // returning the matrix
241     return pitchCorrectionMatrix;
242 }
};

```

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 void SeafloorSetup(ScattererClass* scatterers) {
14
15     // sea-floor bounds
16     int bed_width = 100; // width of the bed (x-dimension)
17     int bed_length = 100; // length of the bed (y-dimension)
18
19     // scatter-intensity
20     int bed_width_density = 100; // density of points along x-dimension
21     int bed_length_density = 100; // density of points along y-dimension
22
23     // setting up coordinates
24     auto xpoints = torch::linspace(0, \
25                                     bed_width, \
26                                     bed_width * bed_width_density).to(DEVICE);
27     auto ypoints = torch::linspace(0, \
28                                     bed_length, \
29                                     bed_length * bed_length_density).to(DEVICE);
30
31     // creating mesh
32     auto mesh_grid = torch::meshgrid({xpoints, ypoints}, "ij");
33     auto X = mesh_grid[0];
34     auto Y = mesh_grid[1];
35     X = torch::reshape(X, {1, X.numel()});
36     Y = torch::reshape(Y, {1, Y.numel()});
37
38     // creating heights of scatterers
39     torch::Tensor Z = torch::zeros({1, Y.numel()}).to(DEVICE);
40
41     // setting up floor coordinates
42     torch::Tensor floorScatter_coordinates = torch::cat({X, Y, Z}, 0);
43     torch::Tensor floorScatter_reflectivity = torch::ones({3, Y.numel()}).to(DEVICE);
44
45     // populating the values of the incoming argument.
46     scatterers->coordinates = floorScatter_coordinates; // assigning coordinates
47     scatterers->reflectivity = floorScatter_reflectivity; // assigning reflectivity
48 }

```

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 // #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/ScattererClass.h"
13 // #include "/Users/vrsreeganesh/Documents/GitHub/AUV/Code/C++/include/TransmitterClass.h"
14
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, pulselength, floor(pulselength *
29     sampling_frequency)).to(DEVICE);
30     float K                  = (f2 - f1)/pulselength;
31     torch::Tensor Signal     = K * timearray;
32     Signal                   = torch::mul((f1 + Signal), timearray);
33     Signal                   = cos(Signal);
34
35     // Setting up transmitter
36     torch::Tensor location   = torch::zeros({3,1}).to(DEVICE); // location of transmitter
37     float azimuthal_angle_fls = 90;                // initial pointing direction
38     float azimuthal_angle_port = 180;              // initial pointing direction
39     float azimuthal_angle_starboard = 0;           // initial pointing direction
40
41     float elevation_angle    = -60;                // initial pointing direction
42
43     float azimuthal_beamwidth = 10;                // azimuthal beamwidth of the signal cone
44     float elevation_beamwidth = 10;                // elevation beamwidth of the signal cone
45
46     float azimuthShadowThreshold = 0.5;            // azimuth threshold
47     float elevationShadowThreshold = 0.5;          // elevation threshold
48
49     int azimuthQuantDensity = 20; // quantization density along azimuth (used for shadowing)
50     int elevationQuantDensity = 20; // quantization density along elevation (used for shadowing)
51     float rangeQuantSize = 20; // cell-dimension (used for shadowing)
52
53
54     // populating transmitter-fls
55     transmitter_fls->location = location;           // Assigning location
56     transmitter_fls->Signal    = Signal;           // Assigning signal
57     transmitter_fls->azimuthal_angle = azimuthal_angle_fls; // assigning azimuth angle
58     transmitter_fls->elevation_angle = elevation_angle;    // assigning elevation angle
59     transmitter_fls->azimuthal_beamwidth = azimuthal_beamwidth; // assigning azimuth-beamwidth
60     transmitter_fls->elevation_beamwidth = elevation_beamwidth; // assigning elevation-beamwidth
61     // updating quantization densities
62     transmitter_fls->azimuthQuantDensity = azimuthQuantDensity; // assigning azimuth quant density
63     transmitter_fls->elevationQuantDensity = elevationQuantDensity; // assigning elevation quant density
64     transmitter_fls->rangeQuantSize = rangeQuantSize; // assigning range-quantization
65     transmitter_fls->azimuthShadowThreshold = azimuthShadowThreshold; // azimuth-threshold in shadowing
66     transmitter_fls->elevationShadowThreshold = elevationShadowThreshold; // elevation-threshold in shadowing

```

```

68 // signal related
69 transmitter_fls->f_low    = f1;          // assigning lower frequency
70 transmitter_fls->f_high   = f2;          // assigning higher frequency
71 transmitter_fls->fc       = fc;          // assigning center frequency
72 transmitter_fls->bandwidth = bandwidth; // assigning bandwidth
73
74
75
76
77 // populating transmitter-portside
78 transmitter_port->location    = location; // Assigning location
79 transmitter_port->Signal      = Signal;   // Assigning signal
80 transmitter_port->azimuthal_angle = azimuthal_angle_port; // assigning azimuth angle
81 transmitter_port->elevation_angle = elevation_angle; // assigning elevation angle
82 transmitter_port->azimuthal_beamwidth = azimuthal_beamwidth; // assigning azimuth-beamwidth
83 transmitter_port->elevation_beamwidth = elevation_beamwidth; // assigning elevation-beamwidth
84 // updating quantization densities
85 transmitter_port->azimuthQuantDensity = azimuthQuantDensity; // assigning azimuth quant density
86 transmitter_port->elevationQuantDensity = elevationQuantDensity; // assigning elevation quant density
87 transmitter_port->rangeQuantSize = rangeQuantSize; // assigning range-quantization
88 transmitter_port->azimuthShadowThreshold = azimuthShadowThreshold; // azimuth-threshold in shadowing
89 transmitter_port->elevationShadowThreshold = elevationShadowThreshold; // elevation-threshold in shadowing
90 // signal related
91 transmitter_port->f_low    = f1;          // assigning lower frequency
92 transmitter_port->f_high   = f2;          // assigning higher frequency
93 transmitter_port->fc       = fc;          // assigning center frequency
94 transmitter_port->bandwidth = bandwidth; // assigning bandwidth
95
96
97
98 // populating transmitter-starboard
99 transmitter_starboard->location    = location;
100 transmitter_starboard->Signal      = Signal;
101 transmitter_starboard->azimuthal_angle = azimuthal_angle_starboard;
102 transmitter_starboard->elevation_angle = elevation_angle;
103 transmitter_starboard->azimuthal_beamwidth = azimuthal_beamwidth;
104 transmitter_starboard->elevation_beamwidth = elevation_beamwidth;
105 // updating quantization densities
106 transmitter_starboard->azimuthQuantDensity = azimuthQuantDensity;
107 transmitter_starboard->elevationQuantDensity = elevationQuantDensity;
108 transmitter_starboard->rangeQuantSize = rangeQuantSize;
109 transmitter_starboard->azimuthShadowThreshold = azimuthShadowThreshold;
110 transmitter_starboard->elevationShadowThreshold = elevationShadowThreshold;
111 // signal related
112 transmitter_starboard->f_low    = f1;          // assigning lower frequency
113 transmitter_starboard->f_high   = f2;          // assigning higher frequency
114 transmitter_starboard->fc       = fc;          // assigning center frequency
115 transmitter_starboard->bandwidth = bandwidth; // assigning bandwidth
116
117 }

```

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  #ifndef DEVICE
8      #define DEVICE      torch::kMPS
9      // #define DEVICE    torch::kCPU
10 #endif
11
12
13
14 // =====
15 void ULASetup(ULAClass* ula_1,
16              ULAClass* ula_2) {
17
18     // setting up ula
19     int num_sensors      = 64;                // number of sensors
20     float sampling_frequency = 160e3;         // sampling frequency
21     float inter_element_spacing = 1500/(2*sampling_frequency); // space between samples
22     float recording_period   = 1;             // sampling-period
23
24     // building the direction for the sensors
25     torch::Tensor ULA_direction = torch::tensor({0,1,0}).reshape({3,1}).to(torch::kFloat).to(DEVICE);
26     ULA_direction   = ULA_direction/torch::linalg_norm(ULA_direction, 2, 0, true, torch::kFloat).to(DEVICE);
27     ULA_direction   = ULA_direction * inter_element_spacing;
28
29     // building the coordinates for the sensors
30     torch::Tensor ULA_coordinates = torch::mul(torch::linspace(0, num_sensors-1, num_sensors).to(DEVICE), \
31                                                ULA_direction);
32
33     // assigning values
34     ula_1->num_sensors      = num_sensors;    // assigning number of sensors
35     ula_1->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
36     ula_1->coordinates      = ULA_coordinates; // assigning ULA coordinates
37     ula_1->sampling_frequency = sampling_frequency; // assigning sampling frequencys
38     ula_1->recording_period  = recording_period; // assigning recording period
39     ula_1->sensorDirection   = ULA_direction;  // ULA direction
40
41     ula_1->num_sensors      = num_sensors;    // assigning number of sensors
42     ula_1->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
43     ula_1->coordinates      = ULA_coordinates; // assigning ULA coordinates
44     ula_1->sampling_frequency = sampling_frequency; // assigning sampling frequencys
45     ula_1->recording_period  = recording_period; // assigning recording period
46     ula_1->sensorDirection   = ULA_direction;  // ULA direction
47
48     // assigning values
49     ula_2->num_sensors      = num_sensors;    // assigning number of sensors
50     ula_2->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
51     ula_2->coordinates      = ULA_coordinates; // assigning ULA coordinates
52     ula_2->sampling_frequency = sampling_frequency; // assigning sampling frequencys
53     ula_2->recording_period  = recording_period; // assigning recording period
54     ula_2->sensorDirection   = ULA_direction;  // ULA direction
55
56     ula_2->num_sensors      = num_sensors;    // assigning number of sensors
57     ula_2->inter_element_spacing = inter_element_spacing; // assigning inter-element spacing
58     ula_2->coordinates      = ULA_coordinates; // assigning ULA coordinates
59     ula_2->sampling_frequency = sampling_frequency; // assigning sampling frequencys
60     ula_2->recording_period  = recording_period; // assigning recording period
61     ula_2->sensorDirection   = ULA_direction;  // ULA direction
62
63 }

```

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({0,0,30}).reshape({3,1}).to(torch::kFloat).to(DEVICE);
17     // starting location of AUV
18     torch::Tensor velocity      = torch::tensor({1,0, 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

Chapter 9

Reading

9.1 Primary Books

- 1.

9.2 Interesting Papers