Obstacle Avoidance and Control of Autonomous Cars Using Model Predictive Control

Dual Degree Project (Stage 1) Report

SUBMITTED BY

Om Prabhu

Roll No : 19D170018

SUPERVISORS

Prof. Avinash Bhardwaj

Mechanical Engineering, IIT Bombay

Prof. K.S. Mallikarjuna Rao

Industrial Engineering & Operations Research, IIT Bombay

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Department of Mechanical Engineering

October 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Aim

1.2 Organization of the report

Chapter 2

Theory

2.1 OCT system

2.1.1 Time Domain OCT

2.1.2 Frequency Domain OCT

2.2 Telesto - III

2.3 ThorImage

Chapter 3

Literature Review

3.1 Lin et al (2018) [?]

Matlab data processing

3.2 Wang and Nuttall (2010) [?]

Chapter 4

Current Work

4.1 Frequency Sweep

A frequency sweep, also referred to as frequency scanning or frequency modulation, is a deliberate variation of frequency over time. It entails the systematic alteration of a signal's frequency within a specified range, either continuously or in discrete steps. In our research experiment, we incorporate vibrometry measurements alongside OCT. This necessitates the use of a frequency sweep signal as input to a speaker. Currently, our setup lacks a dedicated Digital-to-Analog Converter (DAC) port to directly provide the sound signal. To circumvent this limitation, we are utilizing an external device to trigger a separate signal generator, which operates independently of the ThorImage software. Our objective is to develop code that seamlessly integrates with the C++ SDK and facilitates vibrometry experiments without the need for external interventions, streamlining the data acquisition process and enhancing the overall efficiency of our experiment.

MATLAB code for tone burst signal shown in Figure 4.1

```
clear all;
close all;
clc;
% Parameters
frequency = 1; % Frequency of the wave (in Hz)
maxAmplitude = 5; % Maximum amplitude of the wave
duration = 10; % Duration of the wave (in seconds)
samplingRate = 44100; % Sampling rate (in Hz)
maxDuration = 7; % Duration of pause at max amplitude (in seconds)
% Time vector
t = linspace(0, duration, duration * samplingRate);
% Generate the wave
maxSamples = round(maxDuration * samplingRate); % No.of samples in maxduration
amplitude = [linspace(0, maxAmplitude, numel(t)/2-maxSamples/2),
    maxAmplitude*ones(1, maxSamples), linspace(maxAmplitude,0,
```

```
    numel(t)/2-maxSamples/2)];
wave = amplitude .* sin(2*pi*frequency*t);
% Plot the wave
plot(t, wave)
xlabel('Time (s)')
ylabel('Amplitude')
title('Tone burst signal at single frequency')
```

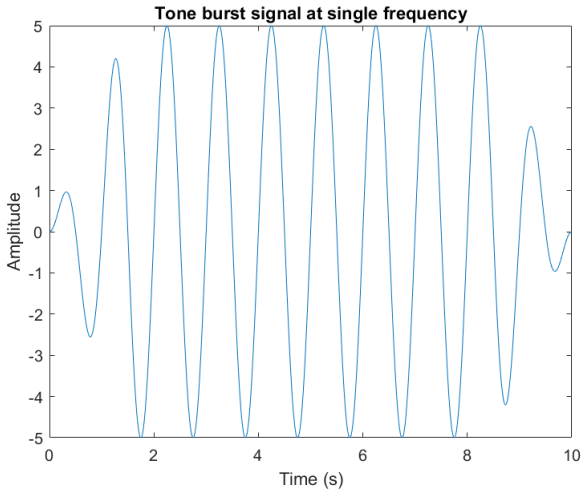

Figure 4.1: Tone burst signal for single frequency in time domain

MATLAB code for frequency sweep signal shown in Figure 4.2

```
clear all;
close all;
clc;
% Parameters
frequencyA = 1; % Starting frequency
frequencyB = 5; % Ending frequency
numFrequencies = 10; % Number of frequencies
maxAmplitude = 5; % Maximum amplitude of the wave
Totduration = 10; % Total duration of the wave (in sec)
dutycycle = 0.8; % duty cycle for single frequency
samplingRate = 44100; % Sampling rate (in Hz)
maxDuration = 7; % Duration of pause at max amplitude (in sec)
%duty cycle allotment
duration = dutycycle*Totduration; % duration of wave (in sec)
Noduration = (1-dutycycle)*Totduration; % duration of free time
% Time vector
t = linspace(0, Totduration, Totduration * samplingRate);
```

```
% Calculate frequency step
frequencyStep = (frequencyB - frequencyA) / (numFrequencies - 1);
% Initialize the waveform
wave = zeros(1, numel(t) * numFrequencies);
% Generate waves for each frequency
for i = 1:numFrequencies
    % Calculate current frequency
    currentFrequency = frequencyA + (i-1) * frequencyStep;
    % Generate the wave
    maxSamples = round(maxDuration * samplingRate);
        % No.of samples in maxduration
    nodursamples = round(Noduration * samplingRate);
        % Number of samples for the 0% duty cycle
    amplitude = [linspace(0, maxAmplitude, numel(t)/2-maxSamples/2-
        nodursamples/2), maxAmplitude*ones(1, maxSamples),
        linspace(maxAmplitude,0,numel(t)/2-maxSamples/2-
        nodursamples/2), maxAmplitude*zeros(1, nodursamples)];
    currentWave = amplitude .* sin(2*pi*currentFrequency*t);
    % Accumulate the waveforms
    wave((i-1)*numel(t)+1:i*numel(t)) = currentWave;
end
t1 = linspace(0,numFrequencies*Totduration,
    numFrequencies*Totduration*samplingRate);
% Plot the combined wave
plot(t1, wave)
xlabel('Time (s)')
ylabel('Amplitude')
title('frequency sweep signal')
```

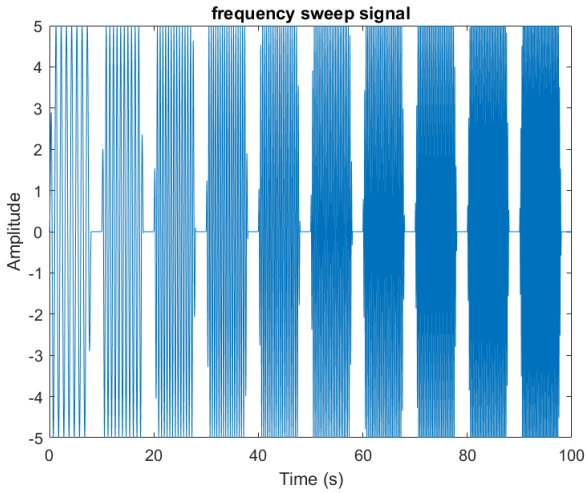

Figure 4.2: frequency sweep signal

4.2 C++ SDK

There are four distinct code files within the C++ SDK, each designed to serve specific functionalities. Let's delve into a detailed discussion of each of these files.

4.2.1 Simple Spectral Radar

This file comprises fundamental demonstration programs that illustrate how to utilize the SDK.

Simple Measurement

This code generates a simple B-scan pattern, allowing users to define the desired range and number of A-scans. Subsequently, it saves the processed data to a CSV file at the specified location.

Export Data and Image

This code generates a basic B-scan pattern, permitting users to define the desired range and number of A-scans. Additionally, it saves the processed data to both a CSV file and an image file at the specified location.

Averaging and Imaging Speed

To enhance image quality, this code offers two options. The first option is to reduce the scan speed, which increases the integration cycle time, resulting in improved image quality. The second option involves performing A and B-scans multiple times and calculating averages. This code executes these operations and generate a B-scan pattern and then saves the processed image at a user-specified location.

Volume Scan Pattern

This code creates a volumetric scan pattern, enabling users to specify the number of A-scans, B-scans, and the desired level of B-scan averaging.

Modify Scan Pattern

In a basic measurement setup, this code initially establishes horizontal scan patterns typically centered at (0.0, 0.0). It offers functions for rotation and zoom around the origin, followed by a shift of the origin point. Subsequently, a B-scan pattern is generated with these adjustments.

Continuous Measurement

This code offers functionality to contunously acquire multiple B-scans for the same defined B-scan pattern

4.2.2 Create Free Form Scan Patterns

This code serves as a demonstration of how to generate scan points using the Freeform Scan Pattern feature, which can be applied within ThorImage or integrated into other functions within the SDK. These scan points are formatted and saved in a standard TXT file, allowing for easy modification. The code enables the creation of scan pattern points for simple geometric shapes like circles, triangles, crosses, and more.

4.2.3 External Processing

This code offers a simple example of an external processing routine that reads raw OCT data from an input file, applies a basic processing step to each B-scan, and then writes the processed data to an output file. Its primary purpose is to showcase how external processing can be integrated with ThorImage to facilitate custom data manipulation.

4.2.4 Advanced Spectral Radar

Write OCT File

This program lets us to write data acquired with the SDK to an oct-file which can be viewed with ThorImageOCT. The generated oct-file is then stored in the same folder where the SDK is present.

Read OCT File

This program lets us read an oct-file with the SDK which has been acquired and saved with ThorImageOCT.

Write OCT File with free form scan pattern

In contrast to the previous Write OCT file function, this code allows for the utilization of custom, free-form scan patterns such as circles, triangles, or crosses, rather than being restricted to the standard straight-line pattern. Subsequently, the program generates an OCT file that can be easily read via ThorImage for further analysis and visualization.

Processing Chain

The processing chain comprises multiple sequential steps, and using the SDK, it's feasible to access and work with the processed data at various intermediate stages, not just at the final step. This function empowers us to process the raw data acquired from the scan pattern using our customized processing routines. These routines encompass examples such as "Offset corrected spectrum," "Spectrum output," "DC corrected spectrum output," "Apodized spectrum output," and more.

Advanced Modification of Scan Pattern

Using this code, it's possible to construct a scan pattern that consists of multiple consecutive B-scans acquired directly one after another. All B-scans within this pattern must have the same number of A-scans, enabling the creation of patterns involving rotating B-scans. One method to achieve such a pattern is to initially create a volume pattern and subsequently modify it using the functions 'shiftScanPatternEx' and 'rotateScanPatternEx'. These functions allow for the shifting and rotation of individual B-scans within the volume pattern. This code demonstrates the usage of the 'rotateScanPatternEx' function, while the use of 'shiftScanPatternEx' follows a similar approach.

Free Form Scan Pattern

The freeform scan pattern functions provide users with the flexibility to generate 2D and 3D scan patterns of virtually any shape. These patterns can be defined using two distinct approaches. Firstly, users can specify only the edge points of the pattern, relying on interpolation methods to calculate the actual scan points within. Alternatively, users have the option to manually create all scan positions, which will be utilized exactly as provided without any interpolation.

Removing Apo from Scan Pattern

In certain scenarios, it can be beneficial to eliminate the acquisition of additional apodization spectra during the processing routine to expedite the acquisition process. This can be achieved by conducting the acquisition of these apodization spectra before initiating the measurement, subsequently using them in the processing chain. By setting the number of apodization spectra to zero, no apodization is applied during the scan, resulting in faster scanning. Additionally, this reduces the time required for the scanner to reach the starting position of each scan, as only one flyback time is needed instead of two in the absence of apodization spectra acquisition.

Doppler OCT

This code initially executes a standard processing routine to generate a B-scan pattern. Subsequently, it performs additional processing steps to perform Doppler OCT imaging. Users have the flexibility to define the output parameters for the Doppler processing and specify averaging parameters for the processing routine. Ultimately, the code carries out Doppler processing to obtain Doppler OCT images.

Speckle Variance OCT

This code begins by running a standard processing routine to create a volume scan pattern. It then proceeds to execute additional processing steps aimed at conducting Speckle Variance OCT imaging. Users are given the option to define averaging parameters for the processing routine as needed. Finally, the code calculates the speckle variance and exports the resulting data in CSV format to a user-specified location.

External Trigger Modus

This code offers the functionality to externally trigger the acquisition of A-scans. This capability allows users to synchronize measurements from various modalities, such as vibrometry and synchronized positioning, with an OCT measurement. It's crucial to ensure that the trigger signal initiates after the 'startMeasurement()' function and concludes after the 'stopMeasurement()' function. This demo program provides clear guidance on when to apply and disable the external trigger for proper synchronization.

4.2.5 Validation experiment

We acquired a simple B-scan pattern using ThorImage software and recreated the identical setup using the C++ SDK. Subsequently, we processed the data obtained through the SDK to generate images for a comparative analysis with those generated by ThorImage. The findings from this comparison are displayed below.

As depicted in Figure 4.4, we observe a B-scan with range of 2.19 mm and a depth of 3.56 mm. Within this range, 10,000 pixels correspond to 10,000 A-scans. To replicate a similar scan pattern as seen in ThorImage software, we provided the C++ SDK's "Modify Scan Pattern" function with these parameters, accounting for a noticeable origin shift present in ThorImage. The resulting scan pattern from the SDK was saved in a CSV file, a snapshot of which is displayed in Figure 4.5. By comparing Figure 4.4 and Figure 4.5, we observe that the depth information, in regions with multiple layers, appears to align between the ThorImage-generated data and that obtained through the SDK.

Figure 4.3: Validation Experimental Setup

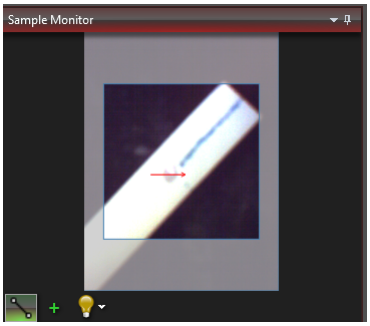

Figure 4.4: View of the sample for the experiment

Figure 4.5: Scan pattern specified in ThorImage software

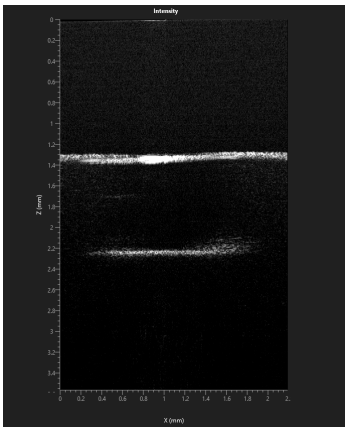

Figure 4.6: Obtained OCT image using ThorImage software

	A	B	C	D	E	F	G	H	I	J	K	L
1	18.2086	36.1738	37.3805	32.8234	37.4937	40.6337	33.8585	26.9528	36.8735	33.5327	41.8828	36.3484
2	33.9641	35.1246	41.6972	32.7152	22.8204	32.208	29.8755	33.4524	27.4284	26.3796	29.5035	32.4466
3	21.1591	27.1199	43.1181	33.3094	35.6302	29.9567	28.3932	34.4412	37.5956	30.7337	37.1182	26.9488
4	33.5093	33.8464	39.8499	24.8211	39.5357	35.1728	32.6548	31.5991	36.2472	33.1697	24.5816	28.1807
5	33.0181	36.7513	30.9602	43.7503	41.4524	32.8923	32.6793	31.6321	30.3872	39.3788	37.9065	31.7103
6	28.9511	28.0226	40.6841	34.2936	35.3531	29.1944	30.7391	29.0511	39.8	41.049	32.2444	29.0891
7	13.6586	35.995	41.364	35.8337	35.2438	43.0309	18.7498	42.9634	32.0352	34.0591	33.2798	37.1852
8	35.0926	43.371	42.8097	39.4017	38.0086	36.3862	37.7741	28.4366	38.5186	34.6521	34.2328	25.9776
9	39.7471	38.4311	42.5453	39.5022	36.8212	41.0435	35.5775	32.9541	37.1331	30.4181	31.4148	34.7094
10	36.5711	34.2188	44.8842	38.3527	40.6212	34.0368	31.4986	34.3609	34.7359	40.4945	35.9499	35.505
11	24.4724	29.0983	41.4967	35.0058	27.6226	11.1893	36.1324	31.383	31.2509	35.3314	35.3292	34.9208
12	33.6006	31.086	34.1115	24.0521	30.8524	24.7591	32.1115	27.1241	37.8123	27.2005	35.1086	37.0116
13	21.3423	36.5724	36.1613	39.2669	21.7709	42.1535	40.1299	29.682	32.5024	38.0383	41.6001	37.9303
14	32.2409	34.2144	33.1163	34.5867	31.219	29.9832	31.0727	23.3217	34.9195	22.7843	29.9472	27.4715
15	7.53896	40.6787	39.9829	39.0095	37.2989	36.2311	33.5871	26.8545	38.7636	27.6189	39.5461	34.2897
16	-7.59768	36.3722	37.4245	42.1775	36.1856	33.763	37.0207	39.1264	28.7487	37.4548	24.8908	22.8225
17	30.9906	31.0869	32.5942	38.3821	36.8604	34.6557	11.4955	40.0799	34.4323	39.5972	39.4166	30.5443
18	35.3967	30.6544	42.7283	41.8373	41.6858	40.3316	39.6323	35.7468	34.5005	37.9325	38.2945	40.8124
19	22.6972	36.0911	31.9597	31.6572	38.2079	36.438	37.6076	35.8624	34.0894	32.335	29.3798	31.3201
20	29.2536	27.556	42.859	32.2799	42.7984	41.5064	37.5596	27.0309	39.2547	40.934	39.5703	26.6541
21	37.3862	27.0002	38.5644	41.0317	36.4933	40.6412	36.0521	32.3401	25.4469	31.9605	34.8794	35.266
22	19.9985	26.1804	39.8883	34.4603	38.8697	28.0961	33.625	29.4465	36.6283	39.4307	12.0934	36.0199
23	33.6757	21.1348	39.5834	36.6143	31.7989	41.1335	39.1723	29.9461	34.4527	27.3756	40.969	36.1018

Figure 4.7: Intensity data obained using SDK

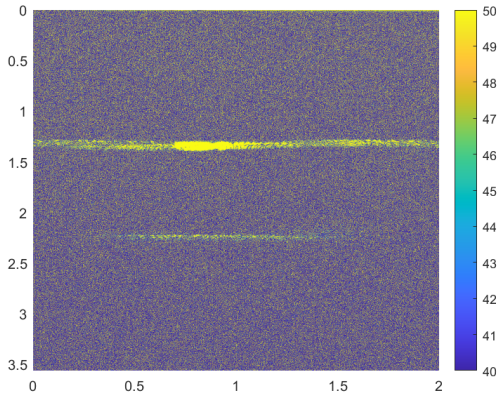

Figure 4.8: Post processed OCT image from data obtained using SDK

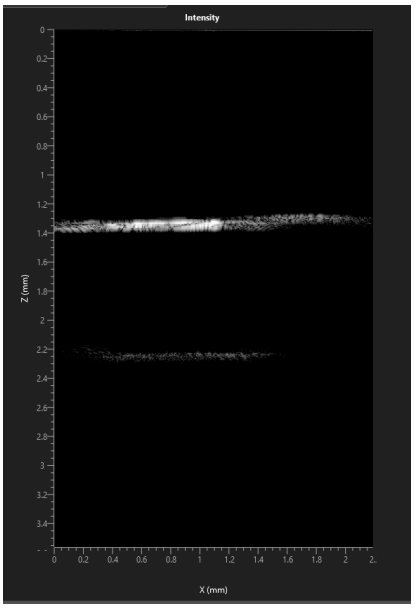

Figure 4.9: .oct file obtained using SDK opened in ThorImage

4.3 Handheld Otoscope

Chapter 5

Future Work