Perception Assignment 4: Motion

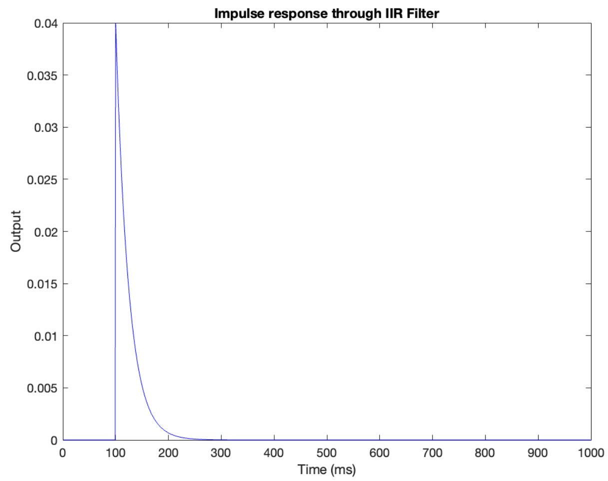
Rachel Chen

29/11/2023

1 Question 1

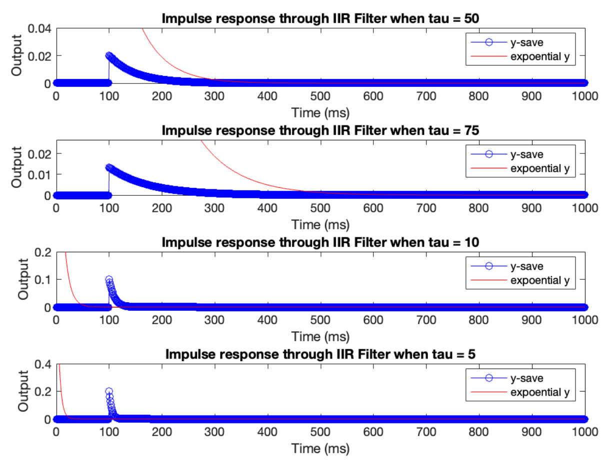
1.1 Question 1a

Figure 1: Simple impulse-response caption after passing through the IIR filter



This is the impulse response presentation for a signal passing through the IIR filter.

Figure 2: Impulse-response (IIR) v.s. Exponential function (with different tau value)



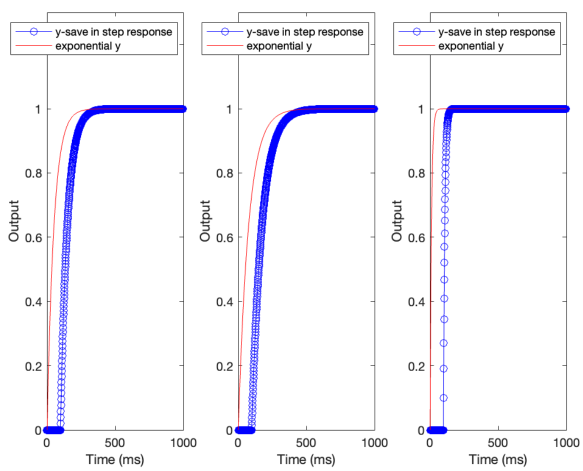
This is the plotting of simulated impulse-response reaction against exponential functions, under different τ values.

```
%Ground setting up
deltaT = 1; %ms
duration = 1000; % ms
t = [0:deltaT:duration-deltaT];
x = zeros(size(t));
x(100) = 1;
tau = 25; % ms

y1 = iirFilter(x, t, deltaT, tau);
figure;
plot(t, y1, 'b-');
xlabel('Time (ms)');
ylabel('Output');
title('Impulse response through IIR Filter')
```

```
% The impulse at time-step = 100 ms
x = zeros(size(t));
x(100) = 1;
tauVals = [50, 75, 10, 5]; % different value of tau

figure;
for a = 1:length(tauVals)
    tau = tauVals(a);
    y1 = iirFilter(x, t, deltaT, tau);

    subplot(length(tauVals),1,  a);
    plot(t, y1, 'bo-'); hold on;

    % Lotting the exponential function
    exponential_function = exp(-(t./tau));
    plot(exponential_function, 'r-');

    xlabel('Time (ms)')
    ylabel('Output')
    ylim([0, max(2*y1)])
    title(['Impulse response through IIR Filter when tau = ', num2str(tau)])
    legend('y-save', 'expoential y');
end
```

1.2 Question 1b

Figure 3: Step impulse-response caption (IIR) (with different tau value)



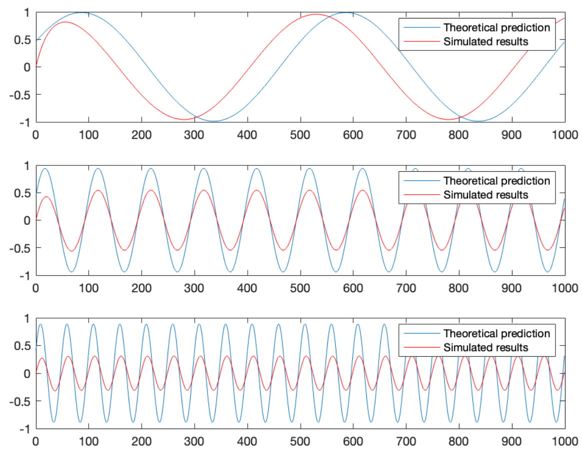
This is the time-step response of impulse-response reaction under different τ values.

```
x = zeros(size(t));
x(100:1000) = 1;
tauVals = [50, 75, 10]; % different value of tau

figure;
for a = 1:length(tauVals)
    tau = tauVals(a);
    y1 = iirFilter(x, t, deltaT, tau);

    subplot(1,length(tauVals),  a);
    plot(t, y1, 'bo-'); hold on;

    % Lotting the exponential function
    exponential_function = 1 - exp(-(t./tau));
    plot(exponential_function, 'r-');

```

```
    xlabel('Time (ms)')
    ylabel('Output')
    ylim([0, 1.3])
    legend('y-save in step response', 'exponential y')

end
```

1.3 Question 1c

Figure 4: Theoretical prediction against simulated results (with varied frequencies)

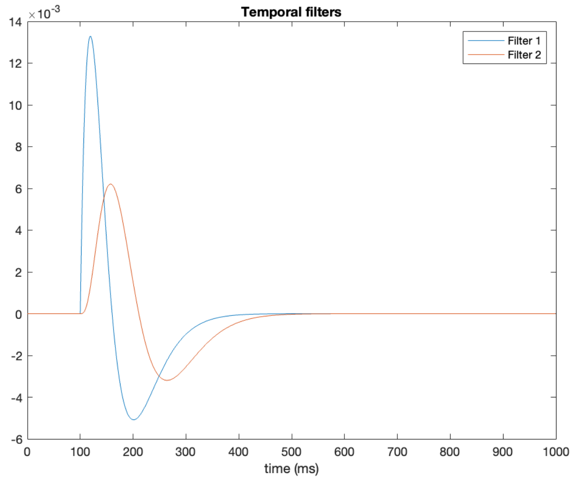


Inputting a sinusoid signal through the IIR filter. The simulated results and the theoretical predictions are not matched.

```
%% Question 1.c
%According to handout, the differential equation is:    dy(t)/dt = -y(t) + x(t);
%Taking the Fourier transform of both sides, the equation in fourier domain
%wil be: d-hat(w)y-hat(w) = -y-hat(w)+x-hat(w).
%The effective frequency response is: h-hat(w) = 1/[1 + d-hat(w)];
%Thus, the equation now will be: y-hat(w) = x-hat(w)/[1+d-hat(w)];
frequencys = [2, 10, 20];
amplitude_theory = 1;
phase_theory = 1.5;

for uu = 1:length(frequencys)

frequency = frequencys(uu);
%generate sinusodio wave
x_sinusoid =sin(2*pi* frequency/1000*t+phase_theory);
%simulated output
y1_sinusoid = iirFilter(x_sinusoid, t, deltaT, tau);

amplitude_response = amplitude_theory / (1+2*pi*frequency/1000);
phase_response = phase_theory / (pi / 2 + phase_theory);
theory_prediction = amplitude_response*sin(2*pi*frequency/1000*t+phase_response);

subplot(3,1,uu);
plot (t, theory_prediction, 'DisplayName', 'Theoretical prediction'); hold on
plot(t, y1_sinusoid , 'r','DisplayName', 'Simulated results'); hold on;
legend();
end
```

2 Question 2

Figure 5: Exponential low-pass filters



The exponential low-pass filter is applied, obtaining two temporal filters: f1(fast filter) and f2 (slow filter).

```
deltaT = 1; %ms
duration = 1000; % ms
t = [0:deltaT:duration-deltaT];
x = zeros(size(t));
x(100) = 1;
tau = 25; % ms

[f1, f2] = lpFilter(x, t, deltaT, tau);
figure;
plot(t, f1, 'DisplayName', 'Filter 1'); hold on;
plot(t, f2, 'DisplayName', 'Filter 2')
xlabel('time (ms)')
title('Temporal filters')
legend()
```

3 Question 3

3.1 Question 3a

Figure 6: x-t slices of the impulse responses of the 4 filters that prefer Horizontal

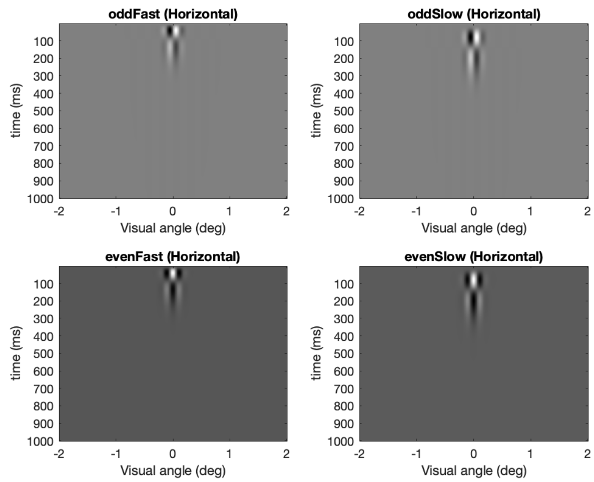
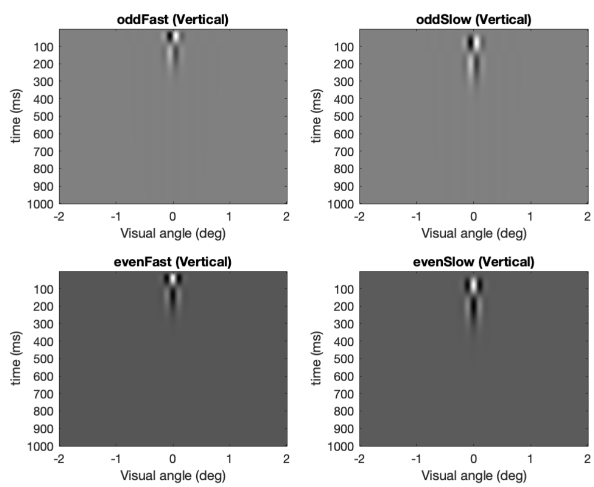


Figure 7: x-t slices of the impulse responses of the 4 filters that prefer Vertical



Calculations are performed for both temporal and spatial filters. Following this, a convolution process is applied to yield oddFast, oddSlow, evenFast, and evenSlow. The direction in which the spatial filters are set influences the direction of the results obtained from the convolution.

```
deltaT = 1; %ms
deltaX=1/120; %spatial sampling rate
duration = 1000; %ms
t = 0:deltaT:duration-deltaT;
xXarray = -2:deltaX:2;
xYarray = -2:deltaX:2;
tau = 25; % ms
sigma = 0.1; %Gaussian sd
sf = 4;

%signal
x = zeros(length(xXarray), length(xYarray), length(t));
x(241, 241, 1) = 1; %input of 480 units, and 240 each onleft and right side of zero, +1so started from oringin
```

```
%Filtering
[f1, f2] = Q3_filters(x, t, deltaT, tau); % computes temporal filters

%Generate gabor
[evenFilt, oddFilt] = generate_gabor(xXarray, sigma, sf);

%Temoral and Horizontal spatial filters
[oddFast_h, evenFast_h, oddSlow_h, evenSlow_h] = temp_gabor(f1, f2, oddFilt, evenFilt);
upLeft = squeeze(oddFast_h(241, :, :))';
upRight = squeeze(oddSlow_h(241, :, :))';
downLeft = squeeze(evenFast_h(241, :, :))';
downRight = squeeze(evenSlow_h(241, :, :))';

 figure1 = figure;
 subplot(2,2,1); imagesc(upLeft); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddFast (Horizontal)');

 subplot(2,2,2); imagesc(upRight); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddSlow (Horizontal)');

 subplot(2,2,3); imagesc(downLeft); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenFast (Horizontal)');

 subplot(2,2,4); imagesc(downRight); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenSlow (Horizontal)');

%Temoral and Vertical spatial filters
[oddFast_v, evenFast_v, oddSlow_v, evenSlow_v] = temp_gabor(f1, f2, oddFilt', evenFilt');
upLeftv = squeeze(oddFast_v(:, 241, :))';
upRightv = squeeze(oddSlow_v(:, 241, :))';
downLeftv = squeeze(evenFast_v(:, 241, :))';
downRightv = squeeze(evenSlow_v(:, 241, :))';

 figure2 = figure;
 subplot(2,2,1); imagesc(upLeftv); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddFast (Vertical)');

 subplot(2,2,2); imagesc(upRightv); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddSlow (Vertical)');

 subplot(2,2,3); imagesc(downLeftv); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenFast (Vertical)');

 subplot(2,2,4); imagesc(downRightv); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenSlow (Vertical)');
```

3.2 Question 3b

Figure 8: x-t slices of the impulse responses of the horizontal selective filters

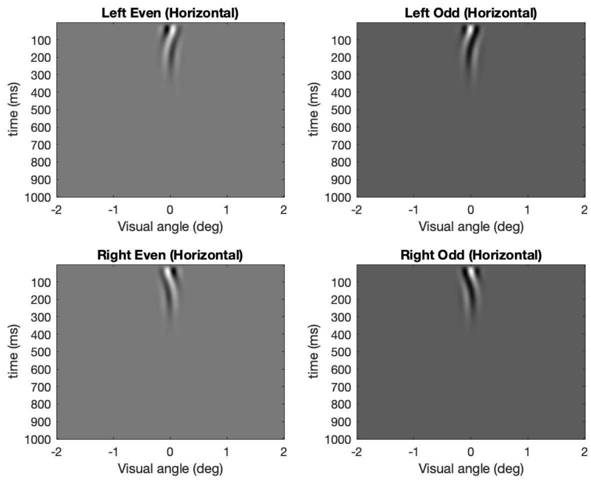
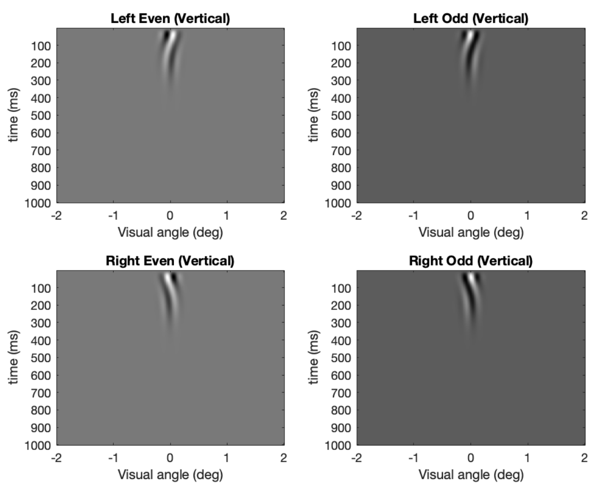


Figure 9: x-t slices of the impulse responses of the vertical selective filters

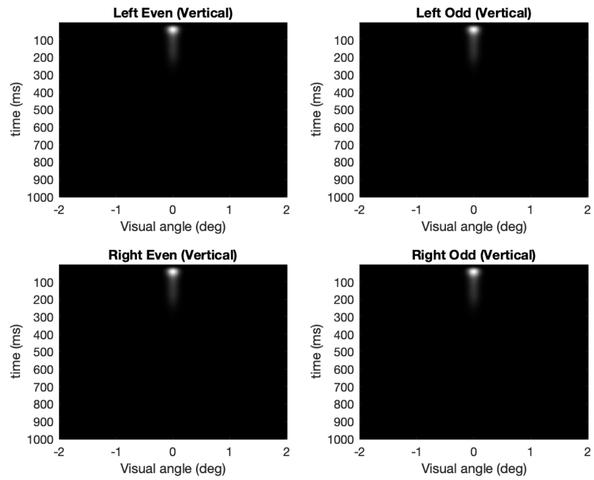



```
%Horizontal
 [leftEven, leftOdd, rightEven, rightOdd] = selective_filter(oddFast_h, oddSlow_h, evenFast_h, evenSlow_h);
 upLeft1 = squeeze(leftEven(241, :, :))';
 upRight1 = squeeze(leftOdd(241, :, :))';
 downLeft1 = squeeze(rightEven(241, :, :))';
 downRight1 = squeeze(rightOdd(241, :, :))';

 figure3 = figure;
 subplot(2,2,1); imagesc(upLeft1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Even (Horizontal)');

 subplot(2,2,2); imagesc(upRight1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Odd (Horizontal)');

 subplot(2,2,3); imagesc(downLeft1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Even (Horizontal)');
```

```
 subplot(2,2,4); imagesc(downRight1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Odd (Horizontal)');


 %Vertical
[upEven, upOdd, downEven, downOdd] = selective_filter(oddFast_v, oddSlow_v, evenFast_v, evenSlow_v);
upLeftv1 = squeeze(upEven(:, 241, :))';
upRightv1 = squeeze(upOdd(:, 241, :))';
downLeftv1 = squeeze(downEven(:, 241, :))';
downRightv1 = squeeze(downOdd(:, 241, :))';

 figure4 = figure;
 subplot(2,2,1); imagesc(upLeftv1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Even (Vertical)');

 subplot(2,2,2); imagesc(upRightv1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Odd (Vertical)');

 subplot(2,2,3); imagesc(downLeftv1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Even (Vertical)');

 subplot(2,2,4); imagesc(downRightv1); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Odd (Vertical)');
```

3.3 Question 3c

Figure 10: x-t slices of the impulse responses of the energy responses



The energy responses for four filters after passing through spatial filter is calculated, based on results of Question 3b.

```
[energyA_h, energyB_h] = generate_energy(oddFast_h, oddSlow_h, evenFast_h, evenSlow_h);
upLeft_energy = squeeze(energyA_h(241, :, :))';
upRighty_energy = squeeze(energyB_h(241, :, :))';

[energyA_v, energyB_v] = generate_energy(oddFast_v, oddSlow_v, evenFast_v, evenSlow_v);
downLeft_energy = squeeze(energyA_v(:, 241, :))';
downRighty_energy = squeeze(energyB_v(:, 241, :))';

figure5 = figure;
 subplot(2,2,1); imagesc(upLeft_energy); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Even (Vertical)');

 subplot(2,2,2); imagesc(upRighty_energy); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Odd (Vertical)');

 subplot(2,2,3); imagesc(downLeft_energy); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Even (Vertical)');

 subplot(2,2,4); imagesc(downRighty_energy); colormap(gray);
 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Odd (Vertical)');
```

3.4 Question 3d

Figure 11: Selective for leftward motion

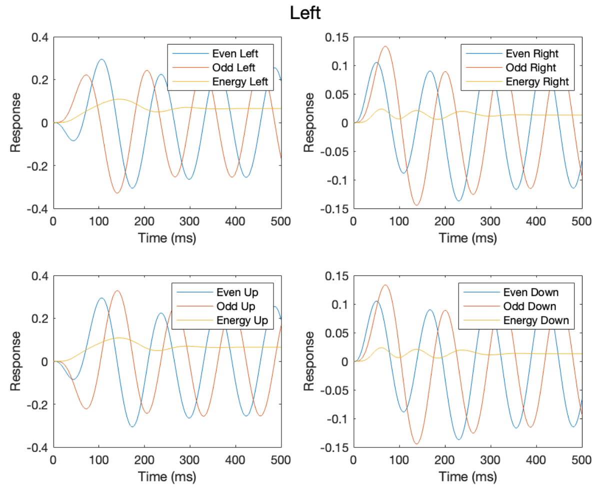


Figure 12: Selective for rightward motion

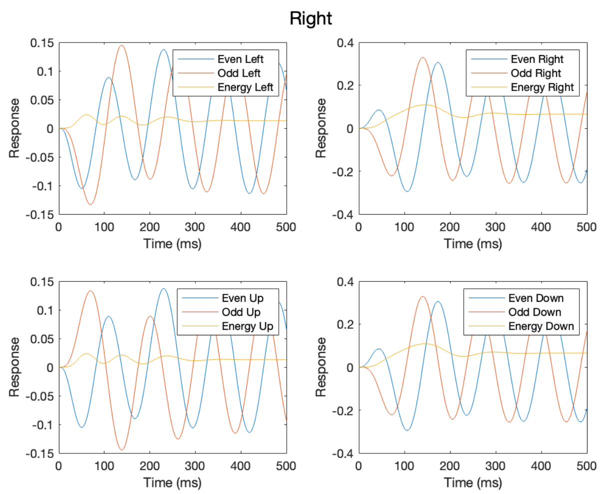


Figure 13: Selective for upward motion

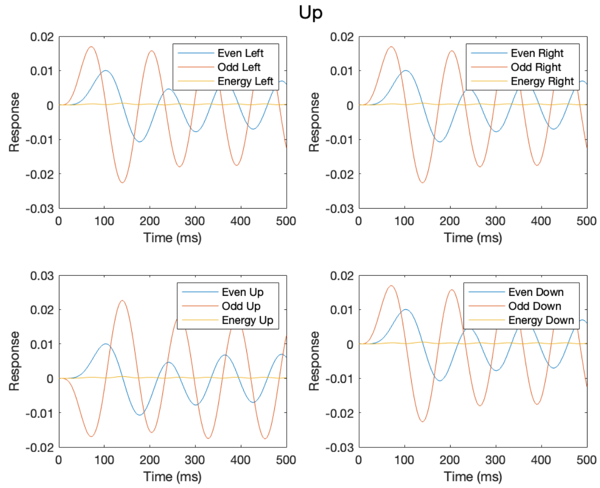
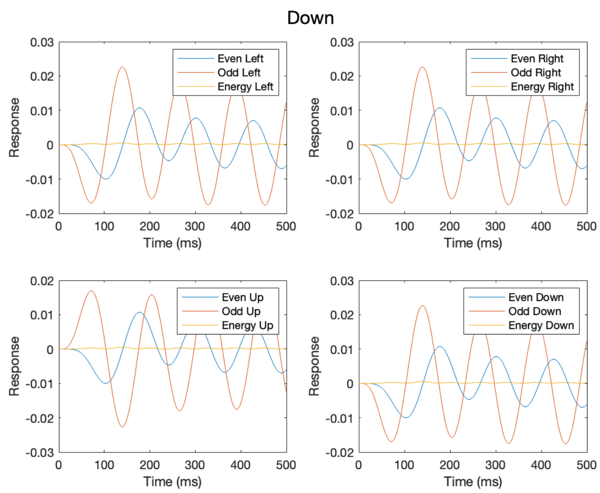


Figure 14: Selective for downward motion



We aim to obtain the response of neurons oriented in four directions by passing grating through them. The input sinusodial signal is: 1) spatial frequency: 4 cycles per degree; 2) -2 to 2 degrees in increments of $\frac{1}{120}$ degree. Therefore, the frequency would be 30/31. For 8hz with in 1 second, making its phase unit to be $\frac{1000}{8}$, which is 125. Therefore, according to the handout, in order to generate the shifts in phases, the shifts would either be $\frac{2pi}{125}$ or $\frac{-2pi}{125}$, depending on the selective direction. therefore, $\frac{2pi}{125}$ would be for left and up; $\frac{-2pi}{125}$ would be for right and down.

The results (8 graphs) are plotting of the four directions of stimulus motion and the four neuron directional preferences. On each plot, three distinct curves, that represent the responses from the even and odd linear filters, along with the energy response, is presented.

```
iniPhase = 0;
shiftPhases = [2*pi/125, -2*pi/125];
sf = 30;
amplitude = 1;

%Horizontally
for Phase_shift = shiftPhases

[evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_h(xXarray, xYarray, t, deltaT, tau
            Phase_shift, sf, oddFilt, evenFilt);
[evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_h(xXarray, xYarray, t, deltaT, tau, amplit
            Phase_shift, sf, oddFilt, evenFilt);
```

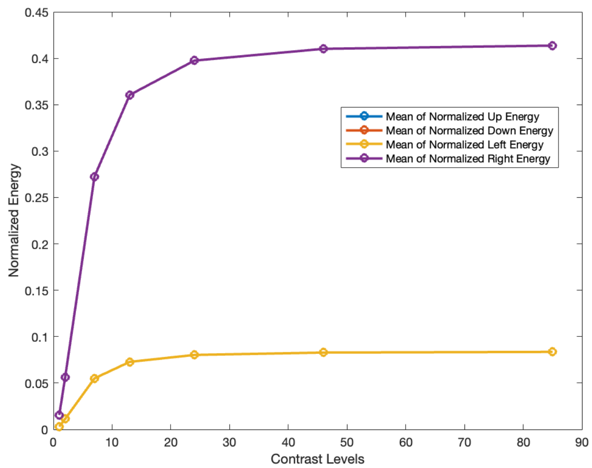
```
if  Phase_shift > 0
    direction_h = 'Left';
else
    direction_h = 'Right';
end

%Plotting
figure6 = figure;
sgtitle(direction_h)

subplot (2, 2, 1);
plot(t, squeeze(evenLeftq(241, 241, :)), 'DisplayName', 'Even Left'); hold on;
plot(t, squeeze(oddLeftq(241, 241, :)), 'DisplayName', 'Odd Left'); hold on;
plot(t, squeeze(energyA_q(241, 241, :)), 'DisplayName', 'Energy Left'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()

subplot (2, 2, 2);
plot(t, squeeze(evenRightq(241, 241, :)), 'DisplayName', 'Even Right'); hold on;
plot(t, squeeze(oddRightq(241, 241, :)), 'DisplayName', 'Odd Right'); hold on;
plot(t, squeeze(energyB_q(241, 241, :)), 'DisplayName', 'Energy Right'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()

subplot (2, 2, 3);
plot(t, squeeze(evenUpw(241, 241, :)), 'DisplayName', 'Even Up'); hold on;
plot(t, squeeze(oddUpw(241, 241, :)), 'DisplayName', 'Odd Up'); hold on;
plot(t, squeeze(energyA_w(241, 241, :)), 'DisplayName', 'Energy Up'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()

subplot (2, 2, 4);
plot(t, squeeze(evenDownw(241, 241, :)), 'DisplayName', 'Even Down'); hold on;
plot(t, squeeze(oddDownw(241, 241, :)), 'DisplayName', 'Odd Down'); hold on;
plot(t, squeeze(energyB_w(241, 241, :)), 'DisplayName', 'Energy Down'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
end

%% Vertically
for Phase_shift = shiftPhases

[evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_v(xXarray, xYarray, t, deltaT, tau
            Phase_shift, sf, oddFilt, evenFilt);
[evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_v(xXarray, xYarray, t, deltaT, tau, amplit
            Phase_shift, sf, oddFilt, evenFilt);

if  Phase_shift > 0
    direction_v = 'Up';
else
    direction_v = 'Down';
end

%Plotting
figure7 = figure;
sgtitle(direction_v)

subplot (2, 2, 1);
plot(t, squeeze(evenLeftq(241, 241, :)), 'DisplayName', 'Even Left'); hold on;
plot(t, squeeze(oddLeftq(241, 241, :)), 'DisplayName', 'Odd Left'); hold on;
plot(t, squeeze(energyA_q(241, 241, :)), 'DisplayName', 'Energy Left'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()

subplot (2, 2, 2);
plot(t, squeeze(evenRightq(241, 241, :)), 'DisplayName', 'Even Right'); hold on;
plot(t, squeeze(oddRightq(241, 241, :)), 'DisplayName', 'Odd Right'); hold on;
plot(t, squeeze(energyB_q(241, 241, :)), 'DisplayName', 'Energy Right'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()

subplot (2, 2, 3);
plot(t, squeeze(evenUpw(241, 241, :)), 'DisplayName', 'Even Up'); hold on;
plot(t, squeeze(oddUpw(241, 241, :)), 'DisplayName', 'Odd Up'); hold on;
plot(t, squeeze(energyA_w(241, 241, :)), 'DisplayName', 'Energy Up'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()

```

```
subplot (2, 2, 4);
plot(t, squeeze(evenDownw(241, 241, :)), 'DisplayName', 'Even Down'); hold on;
plot(t, squeeze(oddDownw(241, 241, :)), 'DisplayName', 'Odd Down'); hold on;
plot(t, squeeze(energyB_w(241, 241, :)), 'DisplayName', 'Energy Down'); hold on;
xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
end
```

4 Question 4

4.1 Question 4a

Figure 15: Normalized energies for rightward drifting gratings (with different contrast levels)



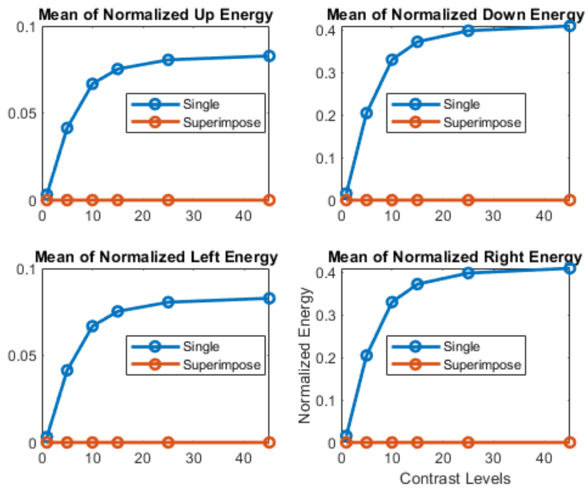
In this task, the motion of a sinusoidal grating to the right is presented under varying contrast levels (1, 2, 7, 13, 24, 46, 85). For each level of contrast, the reaction of neurons selective to four different directions is calculated and normalized. With the rightward grating, the neuron selective to the rightward direction exhibits the highest energy response, followed by the leftward-selective neuron. The neurons selective to upward and downward directions show negligible responses, since it's direction-selective neuron. Altering the contrast demonstrates an increase in neuronal response, which eventually reaches a saturation point.

```
%Right-wards grating
clear; close all; clc;

deltaT = 1; %ms
deltaX=1/120; %spatial sampling rate
duration = 1000; %ms
t = 0:deltaT:duration-deltaT;
xXarray = -2:deltaX:2;
xYarray = -2:deltaX:2;
tau = 25; % ms
sigma = 0.1; %Gaussian sd
sf = 30;
iniPhase = 0;
contrasts = [1, 2, 7, 13, 24, 46, 85]; % contrasts levels
[evenFilt, oddFilt] =generate_gabor(xXarray, sigma, 4);
energy_container = zeros(length(contrasts), 4);
Phase_shift = -2*pi/125; % rightwards shifts


for cc = 1:length (contrasts)
    contrast = contrasts(cc);
    amplitude = contrast*(1/100);

    [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_h(xXarray, xYarray, t, deltaT,
            Phase_shift, sf, oddFilt, evenFilt);
    [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_h(xXarray, xYarray, t, deltaT, tau, an
            Phase_shift, sf, oddFilt, evenFilt);

    timeseries = 241; time = 500;
    [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(energyA_q, energyB_q, ene
    energy_container(cc, :) = [mean(leftEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(rightEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(upEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(downEnergyNorm(timeseries, timeseries, time:end))];
```

```
end

figure();
plot(contrasts, energy_container(:, 1), 'o-', 'DisplayName','Mean of Normalized Up Energy', 'LineWidth', 2); hold on;
plot(contrasts, energy_container(:, 2), 'o-', 'DisplayName','Mean of Normalized Down Energy', 'LineWidth', 2); hold on;
plot(contrasts, energy_container(:, 3), 'o-', 'DisplayName','Mean of Normalized Left Energy', 'LineWidth', 2); hold on;
plot(contrasts, energy_container(:, 4), 'o-', 'DisplayName','Mean of Normalized Right Energy', 'LineWidth', 2); hold on;
xlabel('Contrast Levels'); ylabel('Normalized Energy'); legend('location', 'best');
```

4.2 Question 4b

Figure 16: Normalized energies for rightward drifting gratings and superimposing (with different contrast levels)



In this task, it's a cross-orientation experiment. The contrast steps for rightward grating is between 10% to 50% (1, 5, 10, 15, 25, 45). The contrast steps for upward grating is fixed at 50%. The plotted results are when rightward grating is and is not superimposing with upward grating.

I don't know why I didn't get values for super-imposing situation here and this section is costing my MATLAB to take more than 30 minutes to run. However, theoretically, my anticipation would be different from the graph I obtained:

Specifically, when there is no upward grating, the neuron for rightward movement exhibit strongest reaction. When imposing the upward grating, the response would be weaken as the contrast level increase. For leftward preferred neuron, the trend should be the same but slightly less pronounced, because the direction of grating is not aligned with the neuron's preference. For upward and downward neurons, when the two gratings are superimposing, the response should be decreased as contrast level increase.

```
clear all; close all; clc;

deltaT = 1; %ms
deltaX=1/120; %spatial sampling rate
duration = 1000; %ms
t = 0:deltaT:duration-deltaT;
xXarray = -2:deltaX:2;
xYarray = -2:deltaX:2;
tau = 25; % ms
sigma = 0.1; %Gaussian sd
sf = 30;
iniPhase = 0;
contrasts_right = [1, 5, 10, 15, 25, 45]; % contrasts levels
%contrasts_right = [1, 10,  45];
contrast_up = 50; %50%
[evenFilt, oddFilt] =generate_gabor(xXarray, sigma, 4);
Phase_shift_R = -2*pi/125; % rightwards shifts
Phase_shift = 2*pi/125; % rightwards shifts

%Whether there is upwards signals or whether there is not upwards signals
energy_container_no = zeros(length(contrasts_right), 4);
energy_container_yes = zeros(length(contrasts_right), 4);

%When there is no upwards
for cc = 1:length (contrasts_right)
    contrast = contrasts_right(cc);
    amplitude_right = contrast*(1/100);

    [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_h(xXarray, xYarray, t, deltaT,
        Phase_shift_R, sf, oddFilt, evenFilt);
```

```
    [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_h(xXarray, xYarray, t, deltaT, tau, ar
        Phase_shift_R, sf, oddFilt, evenFilt);

    timeseries = 241; time = 500;
    [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(energyA_q, energyB_q, ene
    energy_container_no(cc, :) = [mean(leftEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(rightEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(upEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(downEnergyNorm(timeseries, timeseries, time:end))];
end

%When there is upwards
for kk = 1:length (contrasts_right)
    contrast = contrasts_right(kk);
    amplitude_right = contrast*(1/100);

    [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, upEven, upOdd, downEven, downOdd, upEnergy, dowr
        generate_superimpose(xXarray, xYarray, t, deltaT, tau, amplitude_right, iniPhase, Phase_shift, sf, oddFilt, ev

    timeseries = 241; time = 500;
    [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(leftEnergy, rightEnergy,
    energy_container_yes(kk, :) = [mean(leftEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(rightEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(upEnergyNorm(timeseries, timeseries, time:end)), ...
        mean(downEnergyNorm(timeseries, timeseries, time:end))];
end

figure();
subplot(2,2,1)
plot (contrasts_right, energy_container_no(:,1), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
plot (contrasts_right, energy_container_yes(:,1), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
legend ('Location','best'); title('Mean of Normalized Up Energy'); hold on;

subplot(2,2,2)
plot (contrasts_right, energy_container_no(:,2), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
plot (contrasts_right, energy_container_yes(:,2), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
legend ('Location','best'); title('Mean of Normalized Down Energy'); hold on;

subplot(2,2,3)
plot (contrasts_right, energy_container_no(:,3), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
plot (contrasts_right, energy_container_yes(:,3), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
legend ('Location','best'); title('Mean of Normalized Left Energy'); hold on;

subplot(2,2,4)
plot (contrasts_right, energy_container_no(:,4), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
plot (contrasts_right, energy_container_yes(:,4), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
legend ('Location','best'); title('Mean of Normalized Right Energy'); hold on;

xlabel('Contrast Levels'); ylabel('Normalized Energy'); legend('location', 'best');
```

5 Functions

5.1 Question 1 and 2 functions

```
%Impulse response filter
%This function compute the response obtained through applying IIR filter
function y1 = iirFilter(x, t, deltaT, tau)
y1 = zeros(length(t),1);
for tt = 1:length(t) - 1
    deltaY1 = (deltaT/tau) * (-y1(tt) + x(tt));
    y1(tt + 1) = y1(tt) + deltaY1;
end
end

function [f1, f2] = lpFilter(x, t, deltaT, tau)
y = zeros(length(t),7);
for tt = 1:length(t) - 1
    for type = 1:7
        if type == 1 || type == 2
            deltaY = (deltaT/tau) * (-y(tt, type) + x(tt));
            y(tt + 1, type) = y(tt, type) + deltaY;
        else
            deltaY = (deltaT / tau) * (-y(tt, type) + y(tt, type - 1));
            y(tt + 1, type) = y(tt, type) + deltaY;
        end
    end
end
%filter 1
f1 = y(:, 3) - y(:, 5);
%filter 2
f2 = y(:, 5) - y(:, 7); % Slow filter
end
```

5.2 Question 3 functions

```
%Time filter
function [f1, f2] = Q3_filters(x, t, deltaT, tau)

    [x_size, y_size, t_length] = size(x);
    y = zeros(x_size, y_size, t_length, 7);
    f1 = zeros(x_size, y_size, t_length);
    f2 = zeros(x_size, y_size, t_length);

    for tt = 1:t_length - 1
        for type = 1:7
            if type == 1
                deltaY = (deltaT / tau) * (-y(:, :, tt, type) + x(:, :, tt));
                y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
            else
                deltaY = (deltaT / tau) * (-y(:, :, tt, type) + y(:, :, tt, type - 1));
                y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
            end
        end
        % filter 1
        f1(:, :, tt) = y(:, :, tt, 3) - y(:, :, tt, 5);
         % filter 2
        f2(:, :, tt) = y(:, :, tt, 5) - y(:, :, tt, 7);
    end
end

%Generate Gabor
function [evenFilt, oddFilt] = generate_gabor(x, sigma, sf)
    evenFilt = exp(-(x.^2)./(2*sigma^2)) .* cos(2*pi*sf*x);
    oddFilt = exp(-(x.^2)./(2*sigma^2)) .* sin(2*pi*sf*x);
    integral = sum(evenFilt.^2 + oddFilt.^2);
    evenFilt = evenFilt / integral;
    oddFilt = oddFilt / integral;
```

```
end

%Convolve filter
function [oddFast, evenFast, oddSlow, evenSlow] = temp_gabor(f1, f2, oddFilt, evenFilt)
    [lx, ly, lt] = size(f1);
    oddFast = zeros(lx, ly, lt);
    oddSlow = zeros(lx, ly, lt);
    evenSlow = zeros(lx, ly, lt);
    evenFast = zeros(lx, ly, lt);

    for tt = 1:lt
        oddFast(:, :, tt) = conv2(f1(:, :, tt), oddFilt, 'same');
        evenFast(:, :, tt) = conv2(f1(:, :, tt), evenFilt, 'same');
        oddSlow(:, :, tt) = conv2(f2(:, :, tt), oddFilt, 'same');
        evenSlow(:, :, tt) = conv2(f2(:, :, tt), evenFilt, 'same');
    end
end

%Side-selective filter
function [evenLeft, oddLeft, evenRight, oddRight] = selective_filter(oddFast, oddSlow, evenFast, evenSlow)
    evenLeft = oddFast + evenSlow;
    oddLeft = -oddSlow + evenFast;
    evenRight = -oddFast + evenSlow;
    oddRight = oddSlow + evenFast;
end

%Energy calculation
function [energyA, energyB] = generate_energy(oddFast, oddSlow, evenFast, evenSlow)
evenLeft = oddFast + evenSlow;
oddLeft = -oddSlow + evenFast;
evenRight = -oddFast + evenSlow;
oddRight = oddSlow + evenFast;
energyA = evenLeft.^2 + oddLeft.^2;
energyB = evenRight.^2. + oddRight.^2;
end



%Horizontal drifting
function [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_h(xXarray, xYarray, t, de
            Phase_shift, sf, oddFilt, evenFilt)

sinusoid_input = zeros(length(xXarray), length(xYarray), length(t));

    %generating a sinusoid input
    for tt = 1:1000
        phase = phase + Phase_shift;
        [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        sinusoid_input(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
    end

    %Applying temporal filter first
    [f1, f2] = Q3_filters(sinusoid_input, t, deltaT, tau);

    %Convolving filters by honrizontal filters
    [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);

    %Side-selective
    [evenLeftq, oddLeftq, evenRightq, oddRightq] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);

    %Compute energie for horizontal
    [energyA_q, energyB_q] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);

end

%Horizontal drifting
function [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_h(xXarray, xYarray, t, deltaT, ta
            Phase_shift, sf, oddFilt, evenFilt)

sinusoid_input2 = zeros(length(xXarray), length(xYarray), length(t));

    %generating a sinusoid input
```

```
    for tt = 1:1000
        phase = phase + Phase_shift;
        [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        sinusoid_input2(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
    end

    %Applying temporal filter first
    [f1, f2] = Q3_filters(sinusoid_input2, t, deltaT, tau);

    %Convolving filters by honrizontal filters
    [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);

    %Side-selective
    [evenUpw, oddUpw, evenDownw, oddDownw] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);

    %Compute energie for horizontal
    [energyA_w, energyB_w] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);

end

%Vertical drifting
function [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_v(xXarray, xYarray, t, de
            Phase_shift, sf, oddFilt, evenFilt)

sinusoid_input = zeros(length(xXarray), length(xYarray), length(t));

    %generating a sinusoid input
    for tt = 1:1000
        phase = phase + Phase_shift;
        [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        sinusoid_input(:, :, tt) = amplitude * sin(Vertical_value + phase);
    end

    %Applying temporal filter first
    [f1, f2] = Q3_filters(sinusoid_input, t, deltaT, tau);

    %Convolving filters by honrizontal filters
    [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);

    %Side-selective
    [evenLeftq, oddLeftq, evenRightq, oddRightq] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);

    %Compute energie for horizontal
    [energyA_q, energyB_q] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);

end

%Vertical drifting
function [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_v(xXarray, xYarray, t, deltaT, ta
            Phase_shift, sf, oddFilt, evenFilt)

sinusoid_input2 = zeros(length(xXarray), length(xYarray), length(t));

    %generating a sinusoid input
    for tt = 1:1000
        phase = phase + Phase_shift;
        [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        sinusoid_input2(:, :, tt) = amplitude * sin(Vertical_value + phase);
    end

    %Applying temporal filter first
    [f1, f2] = Q3_filters(sinusoid_input2, t, deltaT, tau);

    %Convolving filters by honrizontal filters
    [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);

    %Side-selective
    [evenUpw, oddUpw, evenDownw, oddDownw] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);

    %Compute energie for horizontal
    [energyA_w, energyB_w] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);
```

```
end
```

5.3 Question 4 functions

```
%Time filter
function [f1, f2] = Q3_filters(x, t, deltaT, tau)

    [x_size, y_size, t_length] = size(x);
    y = zeros(x_size, y_size, t_length, 7);
    f1 = zeros(x_size, y_size, t_length);
    f2 = zeros(x_size, y_size, t_length);

    for tt = 1:t_length - 1
        for type = 1:7
            if type == 1
                deltaY = (deltaT / tau) * (-y(:, :, tt, type) + x(:, :, tt));
                y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
            else
                deltaY = (deltaT / tau) * (-y(:, :, tt, type) + y(:, :, tt, type - 1));
                y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
            end
        end
        % filter 1
        f1(:, :, tt) = y(:, :, tt, 3) - y(:, :, tt, 5);
         % filter 2
        f2(:, :, tt) = y(:, :, tt, 5) - y(:, :, tt, 7);
    end
end

%Generate Gabor
function [evenFilt, oddFilt] = generate_gabor(x, sigma, sf)
    evenFilt = exp(-(x.^2)./(2*sigma^2)) .* cos(2*pi*sf*x);
    oddFilt = exp(-(x.^2)./(2*sigma^2)) .* sin(2*pi*sf*x);
    integral = sum(evenFilt.^2 + oddFilt.^2);
    evenFilt = evenFilt / integral;
    oddFilt = oddFilt / integral;
end

%Convolve filter
function [oddFast, evenFast, oddSlow, evenSlow] = temp_gabor(f1, f2, oddFilt, evenFilt)
    [lx, ly, lt] = size(f1);
    oddFast = zeros(lx, ly, lt);
    oddSlow = zeros(lx, ly, lt);
    evenSlow = zeros(lx, ly, lt);
    evenFast = zeros(lx, ly, lt);

    for tt = 1:lt
        oddFast(:, :, tt) = conv2(f1(:, :, tt), oddFilt, 'same');
        evenFast(:, :, tt) = conv2(f1(:, :, tt), evenFilt, 'same');
        oddSlow(:, :, tt) = conv2(f2(:, :, tt), oddFilt, 'same');
        evenSlow(:, :, tt) = conv2(f2(:, :, tt), evenFilt, 'same');
    end
end

%Side-selective filter
function [evenLeft, oddLeft, evenRight, oddRight] = selective_filter(oddFast, oddSlow, evenFast, evenSlow)
    evenLeft = oddFast + evenSlow;
    oddLeft = -oddSlow + evenFast;
    evenRight = -oddFast + evenSlow;
    oddRight = oddSlow + evenFast;
end

%Energy calculation
function [energyA, energyB] = generate_energy(oddFast, oddSlow, evenFast, evenSlow)
evenLeft = oddFast + evenSlow;
oddLeft = -oddSlow + evenFast;
evenRight = -oddFast + evenSlow;
oddRight = oddSlow + evenFast;
energyA = evenLeft.^2 + oddLeft.^2;
energyB = evenRight.^2. + oddRight.^2;
```

```
end



%Horizontal drifting
function [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d_horizontal_h(xXarray, xYarray, t, de
            Phase_shift, sf, oddFilt, evenFilt)

sinusoid_input = zeros(length(xXarray), length(xYarray), length(t));

    %generating a sinusoid input
    for tt = 1:1000
        phase = phase + Phase_shift;
        [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        sinusoid_input(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
    end

    %Applying temporal filter first
    [f1, f2] = Q3_filters(sinusoid_input, t, deltaT, tau);

    %Convolving filters by honrizontal filters
    [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);

    %Side-selective
    [evenLeftq, oddLeftq, evenRightq, oddRightq] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);

    %Compute energie for horizontal
    [energyA_q, energyB_q] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);

end

%Vertical drifting
function [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d_vertical_h(xXarray, xYarray, t, deltaT, ta
            Phase_shift, sf, oddFilt, evenFilt)

sinusoid_input2 = zeros(length(xXarray), length(xYarray), length(t));

    %generating a sinusoid input
    for tt = 1:1000
        phase = phase + Phase_shift;
        [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        sinusoid_input2(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
    end

    %Applying temporal filter first
    [f1, f2] = Q3_filters(sinusoid_input2, t, deltaT, tau);

    %Convolving filters by honrizontal filters
    [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);

    %Side-selective
    [evenUpw, oddUpw, evenDownw, oddDownw] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);

    %Compute energie for horizontal
    [energyA_w, energyB_w] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);

end

%Normalization
function [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(leftEnergy, rightEne
sdN = 0.02;
sumEnergy = leftEnergy + rightEnergy + upEnergy + downEnergy;
leftEnergyNorm = leftEnergy ./ (sumEnergy + sdN^2);
rightEnergyNorm = rightEnergy ./ (sumEnergy + sdN^2);
upEnergyNorm = upEnergy ./ (sumEnergy + sdN^2);
downEnergyNorm = downEnergy ./ (sumEnergy + sdN^2);
end

%Superimposing
function [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, upEven, upOdd, downEven, downOdd, upEnergy,
            generate_superimpose(xXarray, xYarray, t, deltaT, tau, amplitude, phase, Phase_shift, sf, oddFilt, evenFil
```

```
superimpose_input= zeros(length(xXarray), length(xYarray), length(t));

for tt = 1:length(t)
    [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
        phase_right = phase - Phase_shift;
        rightwardsinput = amplitude * sin(Horizontal_vale + phase_right);
        phase_up = phase + Phase_shift;
        upwardsinput = 0.5 * sin(Vertical_value + phase_up);
        superimpose_input(:, :, tt) = rightwardsinput + upwardsinput;
end
[f1, f2] = Q3_filters(superimpose_input, t, deltaT, tau);

 %Convolving filters by honrizontal filters
    [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);
[oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);
    %Side-selective
    [leftEven, leftOdd, rightEven, rightOdd] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);
[upEven, upOdd, downEven, downOdd] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);

%Compute energie for horizontal
    [leftEnergy, rightEnergy] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);
    [upEnergy, downEnergy] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);
end
```