

# Perception Assignment 4: Motion

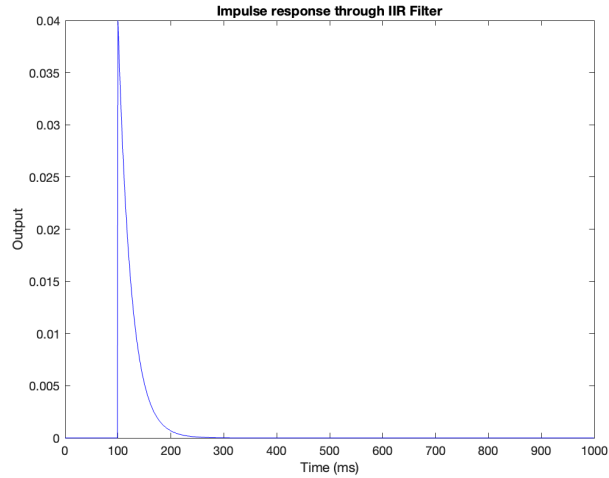
Rachel Chen

29/11/2023

# 1 Question 1

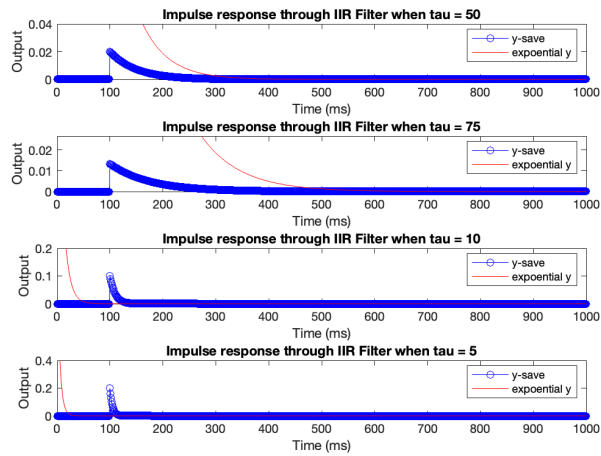
## 1.1 Question 1a

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  $\tau$  values.

```
1 %Ground setting up
2 deltaT = 1; %ms
3 duration = 1000; % ms
4 t = [0:deltaT:duration-deltaT];
5 x = zeros(size(t));
6 x(100) = 1;
7 tau = 25; % ms
8
9 y1 = iirFilter(x, t, deltaT, tau);
10 figure;
11 plot(t, y1, 'b-');
12 xlabel('Time (ms)');
13 ylabel('Output');
14 title('Impulse response through IIR Filter')
```

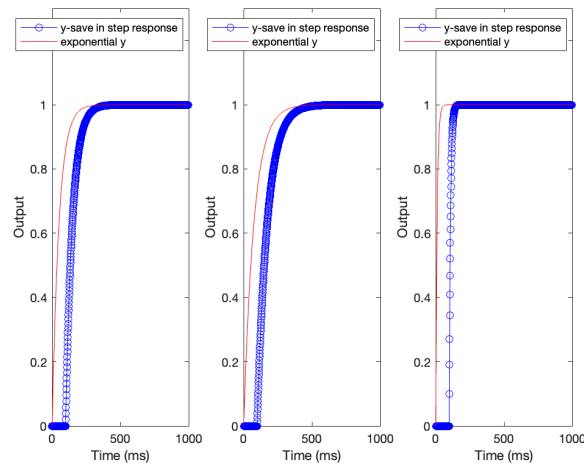
```

15
16 % The impulse at time-step = 100 ms
17 x = zeros(size(t));
18 x(100) = 1;
19 tauVals = [50, 75, 10, 5]; % different value of tau
20
21 figure;
22 for a = 1:length(tauVals)
23     tau = tauVals(a);
24     y1 = iirFilter(x, t, deltaT, tau);
25
26     subplot(length(tauVals),1, a);
27     plot(t, y1, 'bo-'); hold on;
28
29     % Lotting the exponential function
30     exponential_function = exp(-(t./tau));
31     plot(exponential_function, 'r-');
32
33     xlabel('Time (ms)')
34     ylabel('Output')
35     ylim([0, max(2*y1)])
36     title(['Impulse response through IIR Filter when tau = ', num2str(tau)])
37     legend('y-save', 'exponential y');
38 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  $\tau$  values.

```

1 x = zeros(size(t));
2 x(100:1000) = 1;
3 tauVals = [50, 75, 10]; % different value of tau
4
5 figure;
6 for a = 1:length(tauVals)
7     tau = tauVals(a);
8     y1 = iirFilter(x, t, deltaT, tau);
9
10    subplot(1,length(tauVals), a);
11    plot(t, y1, 'bo-'); hold on;
12
13    % Lotting the exponential function
14    exponential_function = 1 - exp(-(t./tau));
15    plot(exponential_function, 'r-');
16

```

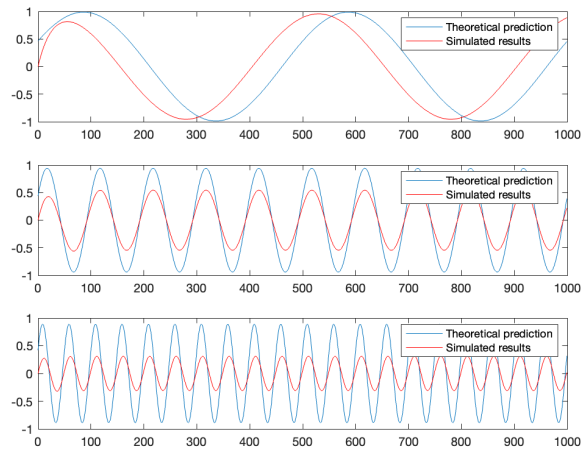
```

17     xlabel('Time (ms)')
18     ylabel('Output')
19     ylim([0, 1.3])
20     legend('y-save in step response', 'exponential y')
21
22 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.

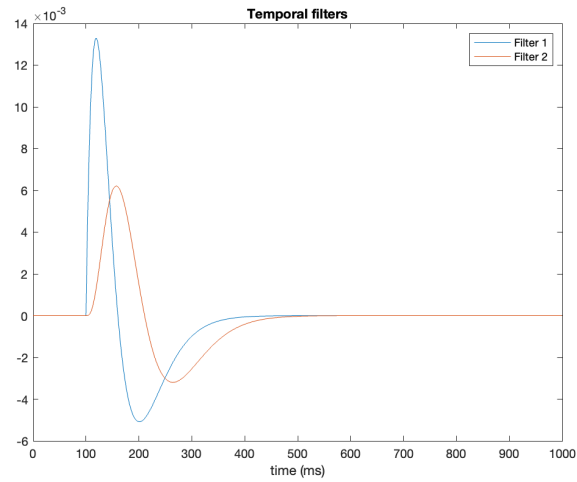
```

1 %% Question 1.c
2 %According to handout, the differential equation is: dy(t)/dt = -y(t) + x(t);
3 %Taking the Fourier transform of both sides, the equation in fourier domain
4 %will be: d-hat(w)y-hat(w) = -y-hat(w)+x-hat(w).
5 %The effective frequency response is: h-hat(w) = 1/[1 + d-hat(w)];
6 %Thus, the equation now will be: y-hat(w) = x-hat(w)/[1+d-hat(w)];
7 frequencys = [2, 10, 20];
8 amplitude.theory = 1;
9 phase.theory = 1.5;
10
11 for uu = 1:length(frequencys)
12
13     frequency = frequencys(uu);
14     %generate sinusodio wave
15     x.sinusoid = sin(2*pi* frequency/1000*t+phase.theory);
16     %simulated output
17     y1.sinusoid = iirFilter(x.sinusoid, t, deltaT, tau);
18
19     amplitude.response = amplitude.theory / (1+2*pi*frequency/1000);
20     phase.response = phase.theory / (pi / 2 + phase.theory);
21     theory-prediction = amplitude.response*sin(2*pi*frequency/1000*t+phase.response);
22
23     subplot(3,1,uu);
24     plot (t, theory-prediction, 'DisplayName', 'Theoretical prediction'); hold on
25     plot(t, y1.sinusoid, 'r','DisplayName', 'Simulated results'); hold on;
26     legend();
27 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).

```
1 deltaT = 1; %ms
2 duration = 1000; % ms
3 t = [0:deltaT:duration-deltaT];
4 x = zeros(size(t));
5 x(100) = 1;
6 tau = 25; % ms
7
8 [f1, f2] = lpFilter(x, t, deltaT, tau);
9 figure;
10 plot(t, f1, 'DisplayName', 'Filter 1'); hold on;
11 plot(t, f2, 'DisplayName', 'Filter 2')
12 xlabel('time (ms)')
13 title('Temporal filters')
14 legend()
```

### 3 Question 3

#### 3.1 Question 3a

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

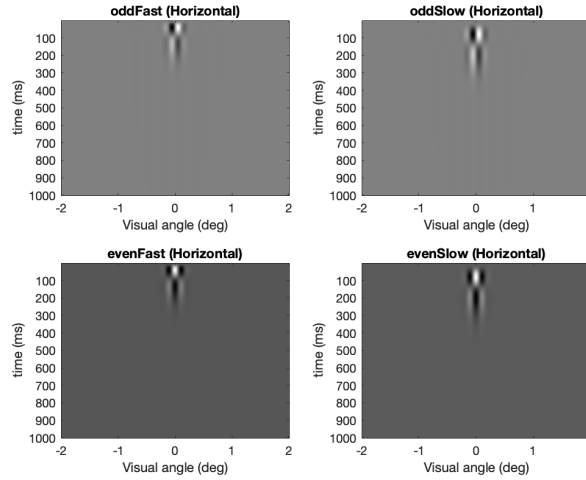
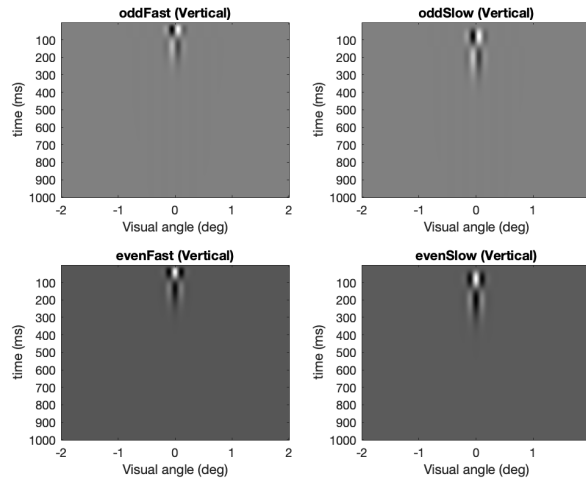


Figure 7: x-t slices of the impulse responses of the 4 filters that prefer Vertical



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.

```
1 deltaT = 1; %ms
2 deltaX=1/120; %spatial sampling rate
3 duration = 1000; %ms
4 t = 0:deltaT:duration-deltaT;
5 xXarray = -2:deltaX:2;
6 xYarray = -2:deltaX:2;
7 tau = 25; % ms
8 sigma = 0.1; %Gaussian sd
9 sf = 4;
10
11 %signal
12 x = zeros(length(xXarray), length(xYarray), length(t));
13 x(241, 241, 1) = 1; %input of 480 units, and 240 each onleft and right side of zero, +lso started from oringin
```

```

14
15 %Filtering
16 [f1, f2] = Q3.filters(x, t, deltaT, tau); % computes temporal filters
17
18 %Generate gabor
19 [evenFilt, oddFilt] = generate_gabor(xXarray, sigma, sf);
20
21 %Temoral and Horizontal spatial filters
22 [oddFast_h, evenFast_h, oddSlow_h, evenSlow_h] = temp_gabor(f1, f2, oddFilt, evenFilt);
23 upLeft = squeeze(oddFast_h(241, :, :));
24 upRight = squeeze(oddSlow_h(241, :, :));
25 downLeft = squeeze(evenFast_h(241, :, :));
26 downRight = squeeze(evenSlow_h(241, :, :));
27
28 figure1 = figure;
29 subplot(2,2,1); imagesc(upLeft); colormap(gray);
30 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
31 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddFast (Horizontal)');
32
33 subplot(2,2,2); imagesc(upRight); colormap(gray);
34 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
35 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddSlow (Horizontal)');
36
37 subplot(2,2,3); imagesc(downLeft); colormap(gray);
38 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
39 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenFast (Horizontal)');
40
41 subplot(2,2,4); imagesc(downRight); colormap(gray);
42 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
43 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenSlow (Horizontal)');
44
45 %Temoral and Vertical spatial filters
46 [oddFast_v, evenFast_v, oddSlow_v, evenSlow_v] = temp_gabor(f1, f2, oddFilt', evenFilt');
47 upLeftv = squeeze(oddFast_v(:, 241, :));
48 upRightv = squeeze(oddSlow_v(:, 241, :));
49 downLeftv = squeeze(evenFast_v(:, 241, :));
50 downRightv = squeeze(evenSlow_v(:, 241, :));
51
52 figure2 = figure;
53 subplot(2,2,1); imagesc(upLeftv); colormap(gray);
54 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
55 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddFast (Vertical)');
56
57 subplot(2,2,2); imagesc(upRightv); colormap(gray);
58 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
59 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('oddSlow (Vertical)');
60
61 subplot(2,2,3); imagesc(downLeftv); colormap(gray);
62 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
63 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('evenFast (Vertical)');
64
65 subplot(2,2,4); imagesc(downRightv); colormap(gray);
66 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
67 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

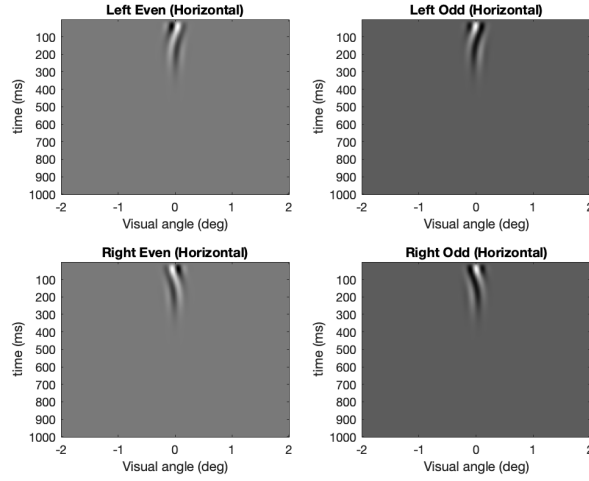
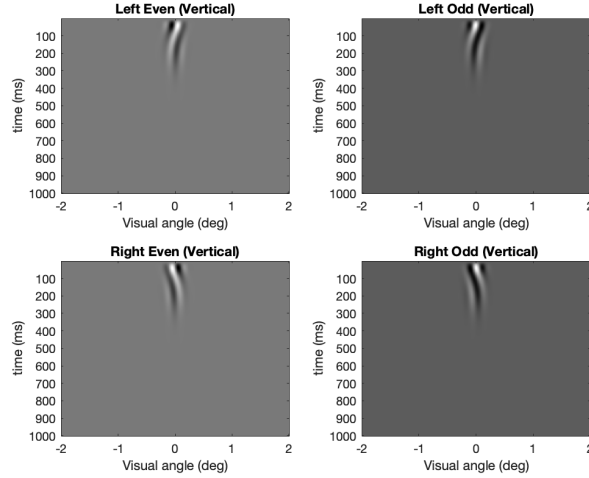


Figure 9: x-t slices of the impulse responses of the vertical selective filters



```

1 %Horizontal
2 [leftEven, leftOdd, rightEven, rightOdd] = selective_filter(oddFast.h, oddSlow.h, evenFast.h, evenSlow.h);
3 upLeft1 = squeeze(leftEven(241, :, :));
4 upRight1 = squeeze(leftOdd(241, :, :));
5 downLeft1 = squeeze(rightEven(241, :, :));
6 downRight1 = squeeze(rightOdd(241, :, :));
7
8 figure3 = figure;
9 subplot(2,2,1); imagesc(upLeft1); colormap(gray);
10 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
11 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Even (Horizontal)');
12
13 subplot(2,2,2); imagesc(upRight1); colormap(gray);
14 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
15 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Odd (Horizontal)');
16
17 subplot(2,2,3); imagesc(downLeft1); colormap(gray);
18 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
19 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Even (Horizontal)');

```



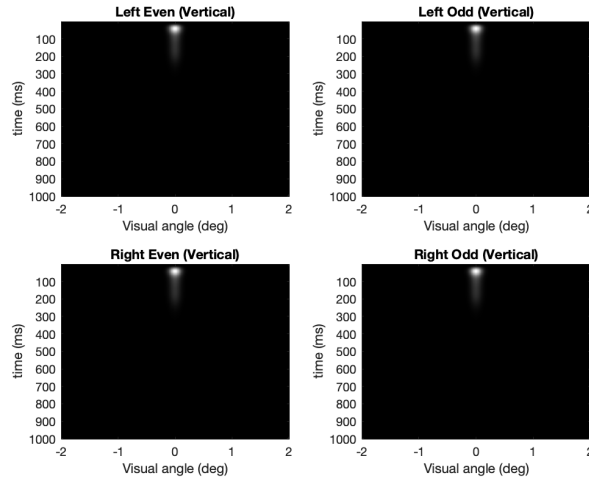
```

20
21 subplot(2,2,4); imagesc(downRight1); colormap(gray);
22 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
23 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Odd (Horizontal)');
24
25
26 %Vertical
27 [upEven, upOdd, downEven, downOdd] = selective_filter(oddFast_v, oddSlow_v, evenFast_v, evenSlow_v);
28 upLeftv1 = squeeze(upEven(:, 241, :));
29 upRightv1 = squeeze(upOdd(:, 241, :));
30 downLeftv1 = squeeze(downEven(:, 241, :));
31 downRightv1 = squeeze(downOdd(:, 241, :));
32
33 figure4 = figure;
34 subplot(2,2,1); imagesc(upLeftv1); colormap(gray);
35 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
36 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Even (Vertical)');
37
38 subplot(2,2,2); imagesc(upRightv1); colormap(gray);
39 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
40 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Odd (Vertical)');
41
42 subplot(2,2,3); imagesc(downLeftv1); colormap(gray);
43 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
44 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Even (Vertical)');
45
46 subplot(2,2,4); imagesc(downRightv1); colormap(gray);
47 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
48 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.

```

1 [energyA_h, energyB_h] = generate_energy(oddFast_h, oddSlow_h, evenFast_h, evenSlow_h);
2 upLeft_energy = squeeze(energyA_h(241, :, :));
3 upRighty_energy = squeeze(energyB_h(241, :, :));
4
5 [energyA_v, energyB_v] = generate_energy(oddFast_v, oddSlow_v, evenFast_v, evenSlow_v);
6 downLeft_energy = squeeze(energyA_v(:, 241, :));
7 downRighty_energy = squeeze(energyB_v(:, 241, :));
8
9 figure5 = figure;
10 subplot(2,2,1); imagesc(upLeft_energy); colormap(gray);
11 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
12 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Even (Vertical)');
13
14 subplot(2,2,2); imagesc(upRighty_energy); colormap(gray);
15 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
16 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Left Odd (Vertical)');
17
18 subplot(2,2,3); imagesc(downLeft_energy); colormap(gray);
19 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
20 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Even (Vertical)');
21
22 subplot(2,2,4); imagesc(downRighty_energy); colormap(gray);
23 xticks([1, 121, 241, 361, 481]); xticklabels([-2, -1, 0, 1, 2]); yticks(0:100:1000)
24 xlabel('Visual angle (deg)'); ylabel('time (ms)'); title('Right Odd (Vertical)');

```

### 3.4 Question 3d

Figure 11: Selective for leftward motion

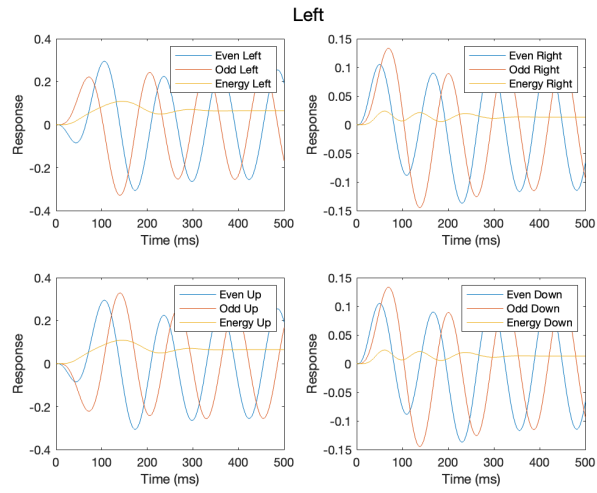


Figure 12: Selective for rightward motion

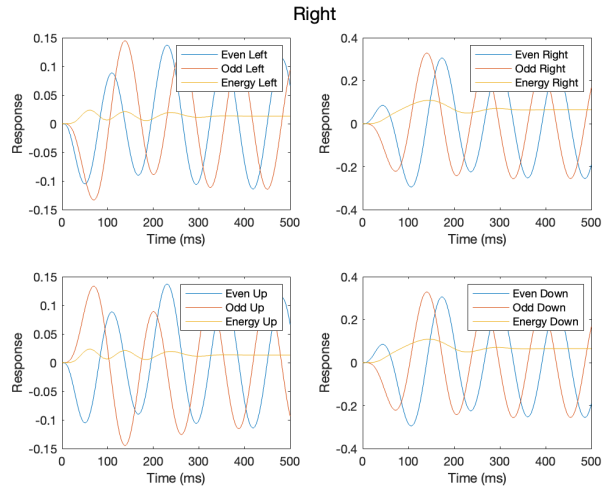


Figure 13: Selective for upward motion

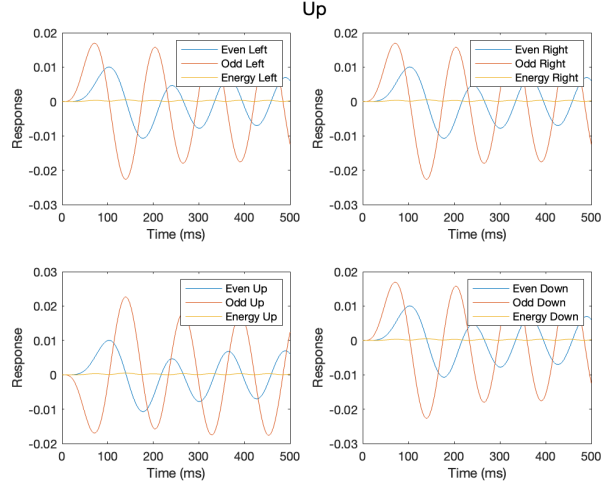
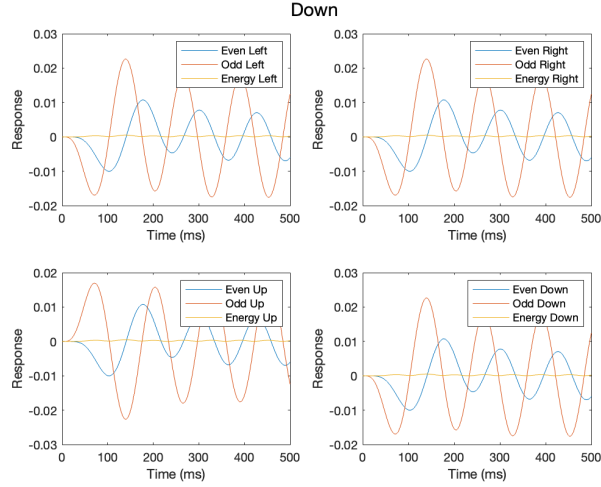


Figure 14: Selective for downward motion



We aim to obtain the response of neurons oriented in four directions by passing grating through them. The input sinusoidal 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  $\frac{2\pi i}{125}$  or  $\frac{-2\pi i}{125}$ , depending on the selective direction. therefore,  $\frac{2\pi i}{125}$  would be for left and up;  $\frac{-2\pi i}{125}$  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.

```

1  iniPhase = 0;
2  shiftPhases = [2*pi/125, -2*pi/125];
3  sf = 30;
4  amplitude = 1;
5
6  %Horizontally
7  for Phase_shift = shiftPhases
8
9      [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA-q, energyB-q] = Q3d.horizontal.h(xXarray, xYarray, t, deltaT, tau,
10         Phase_shift, sf, oddFilt, evenFilt);
11      [evenUpw, oddUpw, evenDownw, oddDownw, energyA-w, energyB-w] = Q3d.vertical.h(xXarray, xYarray, t, deltaT, tau, amplitude,
12         Phase_shift, sf, oddFilt, evenFilt);

```

```

13
14 if Phase_shift > 0
15     direction_h = 'Left';
16 else
17     direction_h = 'Right';
18 end
19
20 %Plotting
21 figure6 = figure;
22 sgtitle(direction_h)
23
24 subplot (2, 2, 1);
25 plot(t, squeeze(evenLeftq(241, 241, :)), 'DisplayName', 'Even Left'); hold on;
26 plot(t, squeeze(oddLeftq(241, 241, :)), 'DisplayName', 'Odd Left'); hold on;
27 plot(t, squeeze(energyA.q(241, 241, :)), 'DisplayName', 'Energy Left'); hold on;
28 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
29
30 subplot (2, 2, 2);
31 plot(t, squeeze(evenRightq(241, 241, :)), 'DisplayName', 'Even Right'); hold on;
32 plot(t, squeeze(oddRightq(241, 241, :)), 'DisplayName', 'Odd Right'); hold on;
33 plot(t, squeeze(energyB.q(241, 241, :)), 'DisplayName', 'Energy Right'); hold on;
34 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
35
36 subplot (2, 2, 3);
37 plot(t, squeeze(evenUpw(241, 241, :)), 'DisplayName', 'Even Up'); hold on;
38 plot(t, squeeze(oddUpw(241, 241, :)), 'DisplayName', 'Odd Up'); hold on;
39 plot(t, squeeze(energyA.w(241, 241, :)), 'DisplayName', 'Energy Up'); hold on;
40 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
41
42 subplot (2, 2, 4);
43 plot(t, squeeze(evenDownw(241, 241, :)), 'DisplayName', 'Even Down'); hold on;
44 plot(t, squeeze(oddDownw(241, 241, :)), 'DisplayName', 'Odd Down'); hold on;
45 plot(t, squeeze(energyB.w(241, 241, :)), 'DisplayName', 'Energy Down'); hold on;
46 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
47 end
48
49 %% Vertically
50 for Phase_shift = shiftPhases
51
52 [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA.q, energyB.q] = Q3d.horizontal_v(xXarray, xYarray, t, deltaT, tau,
53     Phase_shift, sf, oddFilt, evenFilt);
54 [evenUpw, oddUpw, evenDownw, oddDownw, energyA.w, energyB.w] = Q3d.vertical_v(xXarray, xYarray, t, deltaT, tau, amplifi
55     Phase_shift, sf, oddFilt, evenFilt);
56
57 if Phase_shift > 0
58     direction_v = 'Up';
59 else
60     direction_v = 'Down';
61 end
62
63 %Plotting
64 figure7 = figure;
65 sgtitle(direction_v)
66
67 subplot (2, 2, 1);
68 plot(t, squeeze(evenLeftq(241, 241, :)), 'DisplayName', 'Even Left'); hold on;
69 plot(t, squeeze(oddLeftq(241, 241, :)), 'DisplayName', 'Odd Left'); hold on;
70 plot(t, squeeze(energyA.q(241, 241, :)), 'DisplayName', 'Energy Left'); hold on;
71 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
72
73 subplot (2, 2, 2);
74 plot(t, squeeze(evenRightq(241, 241, :)), 'DisplayName', 'Even Right'); hold on;
75 plot(t, squeeze(oddRightq(241, 241, :)), 'DisplayName', 'Odd Right'); hold on;
76 plot(t, squeeze(energyB.q(241, 241, :)), 'DisplayName', 'Energy Right'); hold on;
77 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
78
79 subplot (2, 2, 3);
80 plot(t, squeeze(evenUpw(241, 241, :)), 'DisplayName', 'Even Up'); hold on;
81 plot(t, squeeze(oddUpw(241, 241, :)), 'DisplayName', 'Odd Up'); hold on;
82 plot(t, squeeze(energyA.w(241, 241, :)), 'DisplayName', 'Energy Up'); hold on;
83 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
84

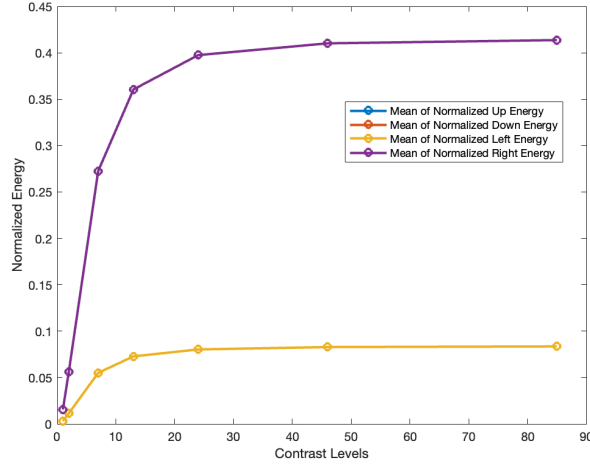
```

```
85 subplot (2, 2, 4);
86 plot(t, squeeze(evenDownw(241, 241, :)), 'DisplayName', 'Even Down'); hold on;
87 plot(t, squeeze(oddDownw(241, 241, :)), 'DisplayName', 'Odd Down'); hold on;
88 plot(t, squeeze(energyB_w(241, 241, :)), 'DisplayName', 'Energy Down'); hold on;
89 xlim([0 500]); xlabel('Time (ms)'); ylabel('Response'); legend()
90 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.

```
1 %Right-wards grating
2 clear; close all; clc;
3
4 deltaT = 1; %ms
5 deltaX=1/120; %spatial sampling rate
6 duration = 1000; %ms
7 t = 0:deltaT:duration-deltaT;
8 xXarray = -2:deltaX:2;
9 xYarray = -2:deltaX:2;
10 tau = 25; % ms
11 sigma = 0.1; %Gaussian sd
12 sf = 30;
13 iniPhase = 0;
14 contrasts = [1, 2, 7, 13, 24, 46, 85]; % contrasts levels
15 [evenFilt, oddFilt] = generate_gabor(xXarray, sigma, 4);
16 energy_container = zeros(length(contrasts), 4);
17 Phase_shift = -2*pi/125; % rightwards shifts
18
19
20 for cc = 1:length(contrasts)
21     contrast = contrasts(cc);
22     amplitude = contrast*(1/100);
23
24     [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA-q, energyB-q] = Q3d.horizontal.h(xXarray, xYarray, t, deltaT,
25         Phase_shift, sf, oddFilt, evenFilt);
26     [evenUpw, oddUpw, evenDownw, oddDownw, energyA-w, energyB-w] = Q3d.vertical.h(xXarray, xYarray, t, deltaT, tau, an
27         Phase_shift, sf, oddFilt, evenFilt);
28
29     timeseries = 241; time = 500;
30     [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(energyA-q, energyB-q, ene
31     energy_container(cc, :) = [mean(leftEnergyNorm(timeseries, timeseries, time:end)), ...
32         mean(rightEnergyNorm(timeseries, timeseries, time:end)), ...
33         mean(upEnergyNorm(timeseries, timeseries, time:end)), ...
34         mean(downEnergyNorm(timeseries, timeseries, time:end))];
```

```

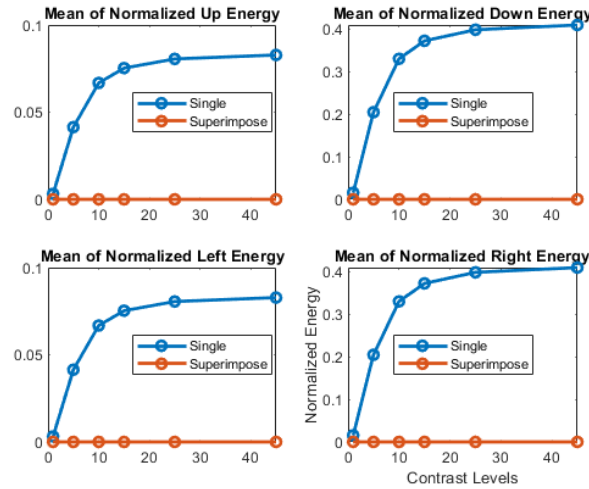
35 end
36
37 figure();
38 plot(contrasts, energy_container(:, 1), 'o-', 'DisplayName', 'Mean of Normalized Up Energy', 'LineWidth', 2);
   hold on;
39 plot(contrasts, energy_container(:, 2), 'o-', 'DisplayName', 'Mean of Normalized Down Energy', 'LineWidth', 2);
   hold on;
40 plot(contrasts, energy_container(:, 3), 'o-', 'DisplayName', 'Mean of Normalized Left Energy', 'LineWidth', 2);
   hold on;
41 plot(contrasts, energy_container(:, 4), 'o-', 'DisplayName', 'Mean of Normalized Right Energy', 'LineWidth', 2);
   hold on;
42 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 weakened 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.

```

1 clear all; close all; clc;
2
3 deltaT = 1; %ms
4 deltaX=1/120; %spatial sampling rate
5 duration = 1000; %ms
6 t = 0:deltaT:duration-deltaT;
7 xXarray = -2:deltaX:2;
8 xYarray = -2:deltaX:2;
9 tau = 25; % ms
10 sigma = 0.1; %Gaussian sd
11 sf = 30;
12 iniPhase = 0;
13 contrasts_right = [1, 5, 10, 15, 25, 45]; % contrasts levels
14 %contrasts_right = [1, 10, 45];
15 contrast_up = 50; %50%
16 [evenFilt, oddFilt] =generate_gabor(xXarray, sigma, 4);
17 Phase_shift_R = -2*pi/125; % rightwards shifts
18 Phase_shift = 2*pi/125; % rightwards shifts
19
20 %Whether there is upwards signals or whether there is not upwards signals
21 energy_container_no = zeros(length(contrasts_right), 4);
22 energy_container_yes = zeros(length(contrasts_right), 4);
23
24 %When there is no upwards
25 for cc = 1:length(contrasts_right)
26     contrast = contrasts_right(cc);
27     amplitude_right = contrast*(1/100);
28
29     [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d.horizontal_h(xXarray, xYarray, t, deltaT,
30         Phase_shift_R, sf, oddFilt, evenFilt);

```

```

31     [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d.verticalh(xXarray, xYarray, t, deltaT, tau, an
32         Phase_shift_R, sf, oddFilt, evenFilt);
33
34     timeseries = 241; time = 500;
35     [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(energyA_q, energyB_q, en
36     energy_container_no(cc, :) = [mean(leftEnergyNorm(timeseries, timeseries, time:end)), ...
37         mean(rightEnergyNorm(timeseries, timeseries, time:end)), ...
38         mean(upEnergyNorm(timeseries, timeseries, time:end)), ...
39         mean(downEnergyNorm(timeseries, timeseries, time:end))];
40 end
41
42 %When there is upwards
43 for kk = 1:length (contrasts_right)
44     contrast = contrasts_right(kk);
45     amplitude_right = contrast*(1/100);
46
47     [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, upEven, upOdd, downEven, downOdd, upEnergy, down
48     generate_superimpose(xXarray, xYarray, t, deltaT, tau, amplitude_right, iniPhase, Phase_shift, sf, oddFilt, ev
49
50     timeseries = 241; time = 500;
51     [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(leftEnergy, rightEnergy,
52     energy_container_yes(kk, :) = [mean(leftEnergyNorm(timeseries, timeseries, time:end)), ...
53         mean(rightEnergyNorm(timeseries, timeseries, time:end)), ...
54         mean(upEnergyNorm(timeseries, timeseries, time:end)), ...
55         mean(downEnergyNorm(timeseries, timeseries, time:end))];
56 end
57
58 figure();
59 subplot(2,2,1)
60 plot (contrasts_right, energy_container_no(:,1), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
61 plot (contrasts_right, energy_container_yes(:,1), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
62 legend ('Location','best'); title('Mean of Normalized Up Energy'); hold on;
63
64 subplot(2,2,2)
65 plot (contrasts_right, energy_container_no(:,2), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
66 plot (contrasts_right, energy_container_yes(:,2), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
67 legend ('Location','best'); title('Mean of Normalized Down Energy'); hold on;
68
69 subplot(2,2,3)
70 plot (contrasts_right, energy_container_no(:,3), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
71 plot (contrasts_right, energy_container_yes(:,3), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
72 legend ('Location','best'); title('Mean of Normalized Left Energy'); hold on;
73
74 subplot(2,2,4)
75 plot (contrasts_right, energy_container_no(:,4), 'o-', 'DisplayName', 'Single', 'LineWidth', 2); hold on;
76 plot (contrasts_right, energy_container_yes(:,4), 'o-', 'DisplayName', 'Superimpose', 'LineWidth', 2); hold on;
77 legend ('Location','best'); title('Mean of Normalized Right Energy'); hold on;
78
79 xlabel('Contrast Levels'); ylabel('Normalized Energy'); legend('location', 'best');

```

## 5 Functions

### 5.1 Question 1 and 2 functions

```
1 %Impulse response filter
2 %This function compute the response obtained through applying IIR filter
3 function y1 = iirFilter(x, t, deltaT, tau)
4 y1 = zeros(length(t),1);
5 for tt = 1:length(t) - 1
6     deltaY1 = (deltaT/tau) * (-y1(tt) + x(tt));
7     y1(tt + 1) = y1(tt) + deltaY1;
8 end
9 end
10
11 function [f1, f2] = lpFilter(x, t, deltaT, tau)
12 y = zeros(length(t),7);
13 for tt = 1:length(t) - 1
14     for type = 1:7
15         if type == 1 || type == 2
16             deltaY = (deltaT/tau) * (-y(tt, type) + x(tt));
17             y(tt + 1, type) = y(tt, type) + deltaY;
18         else
19             deltaY = (deltaT / tau) * (-y(tt, type) + y(tt, type - 1));
20             y(tt + 1, type) = y(tt, type) + deltaY;
21         end
22     end
23 end
24 %filter 1
25 f1 = y(:, 3) - y(:, 5);
26 %filter 2
27 f2 = y(:, 5) - y(:, 7); % Slow filter
28 end
```

### 5.2 Question 3 functions

```
1 %Time filter
2 function [f1, f2] = Q3.filters(x, t, deltaT, tau)
3
4 [x_size, y_size, t_length] = size(x);
5 y = zeros(x_size, y_size, t_length, 7);
6 f1 = zeros(x_size, y_size, t_length);
7 f2 = zeros(x_size, y_size, t_length);
8
9 for tt = 1:t_length - 1
10     for type = 1:7
11         if type == 1
12             deltaY = (deltaT / tau) * (-y(:, :, tt, type) + x(:, :, tt));
13             y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
14         else
15             deltaY = (deltaT / tau) * (-y(:, :, tt, type) + y(:, :, tt, type - 1));
16             y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
17         end
18     end
19     % filter 1
20     f1(:, :, tt) = y(:, :, tt, 3) - y(:, :, tt, 5);
21     % filter 2
22     f2(:, :, tt) = y(:, :, tt, 5) - y(:, :, tt, 7);
23 end
24 end
25
26 %Generate Gabor
27 function [evenFilt, oddFilt] = generate_gabor(x, sigma, sf)
28 evenFilt = exp(-(x.^2)./(2*sigma^2)) .* cos(2*pi*sf*x);
29 oddFilt = exp(-(x.^2)./(2*sigma^2)) .* sin(2*pi*sf*x);
30 integral = sum(evenFilt.^2 + oddFilt.^2);
31 evenFilt = evenFilt / integral;
32 oddFilt = oddFilt / integral;
```

```

33 end
34
35 %Convolve filter
36 function [oddFast, evenFast, oddSlow, evenSlow] = temp_gabor(f1, f2, oddFilt, evenFilt)
37     [lx, ly, lt] = size(f1);
38     oddFast = zeros(lx, ly, lt);
39     oddSlow = zeros(lx, ly, lt);
40     evenSlow = zeros(lx, ly, lt);
41     evenFast = zeros(lx, ly, lt);
42
43     for tt = 1:lt
44         oddFast(:, :, tt) = conv2(f1(:, :, tt), oddFilt, 'same');
45         evenFast(:, :, tt) = conv2(f1(:, :, tt), evenFilt, 'same');
46         oddSlow(:, :, tt) = conv2(f2(:, :, tt), oddFilt, 'same');
47         evenSlow(:, :, tt) = conv2(f2(:, :, tt), evenFilt, 'same');
48     end
49 end
50
51 %Side-selective filter
52 function [evenLeft, oddLeft, evenRight, oddRight] = selective_filter(oddFast, oddSlow, evenFast, evenSlow)
53     evenLeft = oddFast + evenSlow;
54     oddLeft = -oddSlow + evenFast;
55     evenRight = -oddFast + evenSlow;
56     oddRight = oddSlow + evenFast;
57 end
58
59 %Energy calculation
60 function [energyA, energyB] = generate_energy(oddFast, oddSlow, evenFast, evenSlow)
61     evenLeft = oddFast + evenSlow;
62     oddLeft = -oddSlow + evenFast;
63     evenRight = -oddFast + evenSlow;
64     oddRight = oddSlow + evenFast;
65     energyA = evenLeft.^2 + oddLeft.^2;
66     energyB = evenRight.^2 + oddRight.^2;
67 end
68
69
70
71 %Horizontal drifting
72 function [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d.horizontal_h(xXarray, xYarray, t, deltaT, tau,
73     Phase_shift, sf, oddFilt, evenFilt)
74
75 sinusoid_input = zeros(length(xXarray), length(xYarray), length(t));
76
77 %generating a sinusoid input
78 for tt = 1:1000
79     phase = phase + Phase_shift;
80     [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
81     sinusoid_input(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
82 end
83
84 %Applying temporal filter first
85 [f1, f2] = Q3.filters(sinusoid_input, t, deltaT, tau);
86
87 %Convoluting filters by honrizontal filters
88 [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);
89
90 %Side-selective
91 [evenLeftq, oddLeftq, evenRightq, oddRightq] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);
92
93 %Compute energie for horizontal
94 [energyA_q, energyB_q] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);
95
96 end
97
98 %Horizontal drifting
99 function [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d.vertical_h(xXarray, xYarray, t, deltaT, tau,
100     Phase_shift, sf, oddFilt, evenFilt)
101
102 sinusoid_input2 = zeros(length(xXarray), length(xYarray), length(t));
103
104 %generating a sinusoid input

```

```

105     for tt = 1:1000
106         phase = phase + Phase_shift;
107         [Horizontal.vale, Vertical.value] = meshgrid(xXarray .* sf);
108         sinusoid.input2(:, :, tt) = amplitude * sin(Horizontal.vale + phase);
109     end
110
111     %Applying temporal filter first
112     [f1, f2] = Q3.filters(sinusoid.input2, t, deltaT, tau);
113
114     %Convolving filters by honrizontal filters
115     [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);
116
117     %Side-selective
118     [evenUpw, oddUpw, evenDownw, oddDownw] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);
119
120     %Compute energie for horizontal
121     [energyA.w, energyB.w] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);
122
123 end
124
125 %Vertical drifting
126 function [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA.q, energyB.q] = Q3d.horizontal.v(xXarray, xYarray, t, deltaT, tau, Phase_shift, sf, oddFilt, evenFilt)
127
128
129 sinusoid.input = zeros(length(xXarray), length(xYarray), length(t));
130
131     %generating a sinusoid input
132     for tt = 1:1000
133         phase = phase + Phase_shift;
134         [Horizontal.vale, Vertical.value] = meshgrid(xXarray .* sf);
135         sinusoid.input(:, :, tt) = amplitude * sin(Vertical.value + phase);
136     end
137
138     %Applying temporal filter first
139     [f1, f2] = Q3.filters(sinusoid.input, t, deltaT, tau);
140
141     %Convolving filters by honrizontal filters
142     [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);
143
144     %Side-selective
145     [evenLeftq, oddLeftq, evenRightq, oddRightq] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);
146
147     %Compute energie for horizontal
148     [energyA.q, energyB.q] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);
149
150 end
151
152 %Vertical drifting
153 function [evenUpw, oddUpw, evenDownw, oddDownw, energyA.w, energyB.w] = Q3d.vertical.v(xXarray, xYarray, t, deltaT, tau, Phase_shift, sf, oddFilt, evenFilt)
154
155
156 sinusoid.input2 = zeros(length(xXarray), length(xYarray), length(t));
157
158     %generating a sinusoid input
159     for tt = 1:1000
160         phase = phase + Phase_shift;
161         [Horizontal.vale, Vertical.value] = meshgrid(xXarray .* sf);
162         sinusoid.input2(:, :, tt) = amplitude * sin(Vertical.value + phase);
163     end
164
165     %Applying temporal filter first
166     [f1, f2] = Q3.filters(sinusoid.input2, t, deltaT, tau);
167
168     %Convolving filters by honrizontal filters
169     [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);
170
171     %Side-selective
172     [evenUpw, oddUpw, evenDownw, oddDownw] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);
173
174     %Compute energie for horizontal
175     [energyA.w, energyB.w] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);
176

```

177 end

## 5.3 Question 4 functions

```
1 %Time filter
2 function [f1, f2] = Q3.filters(x, t, deltaT, tau)
3
4     [x_size, y_size, t_length] = size(x);
5     y = zeros(x_size, y_size, t_length, 7);
6     f1 = zeros(x_size, y_size, t_length);
7     f2 = zeros(x_size, y_size, t_length);
8
9     for tt = 1:t_length - 1
10         for type = 1:7
11             if type == 1
12                 deltaY = (deltaT / tau) * (-y(:, :, tt, type) + x(:, :, tt));
13                 y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
14             else
15                 deltaY = (deltaT / tau) * (-y(:, :, tt, type) + y(:, :, tt, type - 1));
16                 y(:, :, tt + 1, type) = y(:, :, tt, type) + deltaY;
17             end
18         end
19         % filter 1
20         f1(:, :, tt) = y(:, :, tt, 3) - y(:, :, tt, 5);
21         % filter 2
22         f2(:, :, tt) = y(:, :, tt, 5) - y(:, :, tt, 7);
23     end
24 end
25
26 %Generate Gabor
27 function [evenFilt, oddFilt] = generate_gabor(x, sigma, sf)
28     evenFilt = exp(-(x.^2)./(2*sigma^2)) .* cos(2*pi*sf*x);
29     oddFilt = exp(-(x.^2)./(2*sigma^2)) .* sin(2*pi*sf*x);
30     integral = sum(evenFilt.^2 + oddFilt.^2);
31     evenFilt = evenFilt / integral;
32     oddFilt = oddFilt / integral;
33 end
34
35 %Convolve filter
36 function [oddFast, evenFast, oddSlow, evenSlow] = temp_gabor(f1, f2, oddFilt, evenFilt)
37     [lx, ly, lt] = size(f1);
38     oddFast = zeros(lx, ly, lt);
39     oddSlow = zeros(lx, ly, lt);
40     evenSlow = zeros(lx, ly, lt);
41     evenFast = zeros(lx, ly, lt);
42
43     for tt = 1:lt
44         oddFast(:, :, tt) = conv2(f1(:, :, tt), oddFilt, 'same');
45         evenFast(:, :, tt) = conv2(f1(:, :, tt), evenFilt, 'same');
46         oddSlow(:, :, tt) = conv2(f2(:, :, tt), oddFilt, 'same');
47         evenSlow(:, :, tt) = conv2(f2(:, :, tt), evenFilt, 'same');
48     end
49 end
50
51 %Side-selective filter
52 function [evenLeft, oddLeft, evenRight, oddRight] = selective_filter(oddFast, oddSlow, evenFast, evenSlow)
53     evenLeft = oddFast + evenSlow;
54     oddLeft = -oddSlow + evenFast;
55     evenRight = -oddFast + evenSlow;
56     oddRight = oddSlow + evenFast;
57 end
58
59 %Energy calculation
60 function [energyA, energyB] = generate_energy(oddFast, oddSlow, evenFast, evenSlow)
61     evenLeft = oddFast + evenSlow;
62     oddLeft = -oddSlow + evenFast;
63     evenRight = -oddFast + evenSlow;
64     oddRight = oddSlow + evenFast;
65     energyA = evenLeft.^2 + oddLeft.^2;
66     energyB = evenRight.^2 + oddRight.^2;
```

```

67 end
68
69
70
71 %Horizontal drifting
72 function [evenLeftq, oddLeftq, evenRightq, oddRightq, energyA_q, energyB_q] = Q3d.horizontal_h(xXarray, xYarray, t, deltaT, tau,
73     Phase_shift, sf, oddFilt, evenFilt)
74
75 sinusoid_input = zeros(length(xXarray), length(xYarray), length(t));
76
77 %generating a sinusoid input
78 for tt = 1:1000
79     phase = phase + Phase_shift;
80     [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
81     sinusoid_input(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
82 end
83
84 %Applying temporal filter first
85 [f1, f2] = Q3.filters(sinusoid_input, t, deltaT, tau);
86
87 %Convolving filters by honrizontal filters
88 [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);
89
90 %Side-selective
91 [evenLeftq, oddLeftq, evenRightq, oddRightq] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);
92
93 %Compute energie for horizontal
94 [energyA_q, energyB_q] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);
95
96 end
97
98 %Vertical drifting
99 function [evenUpw, oddUpw, evenDownw, oddDownw, energyA_w, energyB_w] = Q3d.vertical_h(xXarray, xYarray, t, deltaT, tau,
100     Phase_shift, sf, oddFilt, evenFilt)
101
102 sinusoid_input2 = zeros(length(xXarray), length(xYarray), length(t));
103
104 %generating a sinusoid input
105 for tt = 1:1000
106     phase = phase + Phase_shift;
107     [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
108     sinusoid_input2(:, :, tt) = amplitude * sin(Horizontal_vale + phase);
109 end
110
111 %Applying temporal filter first
112 [f1, f2] = Q3.filters(sinusoid_input2, t, deltaT, tau);
113
114 %Convolving filters by honrizontal filters
115 [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);
116
117 %Side-selective
118 [evenUpw, oddUpw, evenDownw, oddDownw] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);
119
120 %Compute energie for horizontal
121 [energyA_w, energyB_w] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);
122
123 end
124
125 %Normalization
126 function [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = generate_normalization(leftEnergy, rightEnergy, upEnergy, downEnergy, sdN);
127 sdN = 0.02;
128 sumEnergy = leftEnergy + rightEnergy + upEnergy + downEnergy;
129 leftEnergyNorm = leftEnergy ./ (sumEnergy + sdN^2);
130 rightEnergyNorm = rightEnergy ./ (sumEnergy + sdN^2);
131 upEnergyNorm = upEnergy ./ (sumEnergy + sdN^2);
132 downEnergyNorm = downEnergy ./ (sumEnergy + sdN^2);
133 end
134
135 %Superimposing
136 function [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] =
137     generate_superimpose(xXarray, xYarray, t, deltaT, tau, amplitude, phase, Phase_shift, sf, oddFilt, evenFilt)
138

```

```

139 superimpose_input= zeros(length(xXarray), length(xYarray), length(t));
140
141 for tt = 1:length(t)
142     [Horizontal_vale, Vertical_value] = meshgrid(xXarray .* sf);
143     phase_right = phase - Phase_shift;
144     rightwardsinput = amplitude * sin(Horizontal_vale + phase_right);
145     phase_up = phase + Phase_shift;
146     upwardsinput = 0.5 * sin(Vertical_value + phase_up);
147     superimpose_input(:, :, tt) = rightwardsinput + upwardsinput;
148 end
149 [f1, f2] = Q3.filters(superimpose_input, t, deltaT, tau);
150
151 %Convolving filters by honrizontal filters
152 [oddFastq, evenFastq, oddSlowq, evenSlowq] = temp_gabor(f1, f2, oddFilt, evenFilt);
153 [oddFastw, evenFastw, oddSloww, evenSloww] = temp_gabor(f1, f2, oddFilt, evenFilt);
154 %Side-selective
155 [leftEven, leftOdd, rightEven, rightOdd] = selective_filter(oddFastq, oddSlowq, evenFastq, evenSlowq);
156 [upEven, upOdd, downEven, downOdd] = selective_filter(oddFastw, evenFastw, oddSloww, evenSloww);
157
158 %Compute energie for horizontal
159 [leftEnergy, rightEnergy] = generate_energy(oddFastq, oddSlowq, evenFastq, evenSlowq);
160 [upEnergy, downEnergy] = generate_energy(oddFastw, oddSloww, evenFastw, evenSloww);
161 end

```