امیرعلی رحیمی 8101100146

پروژه ی ششم سیگنال و سیستم ها

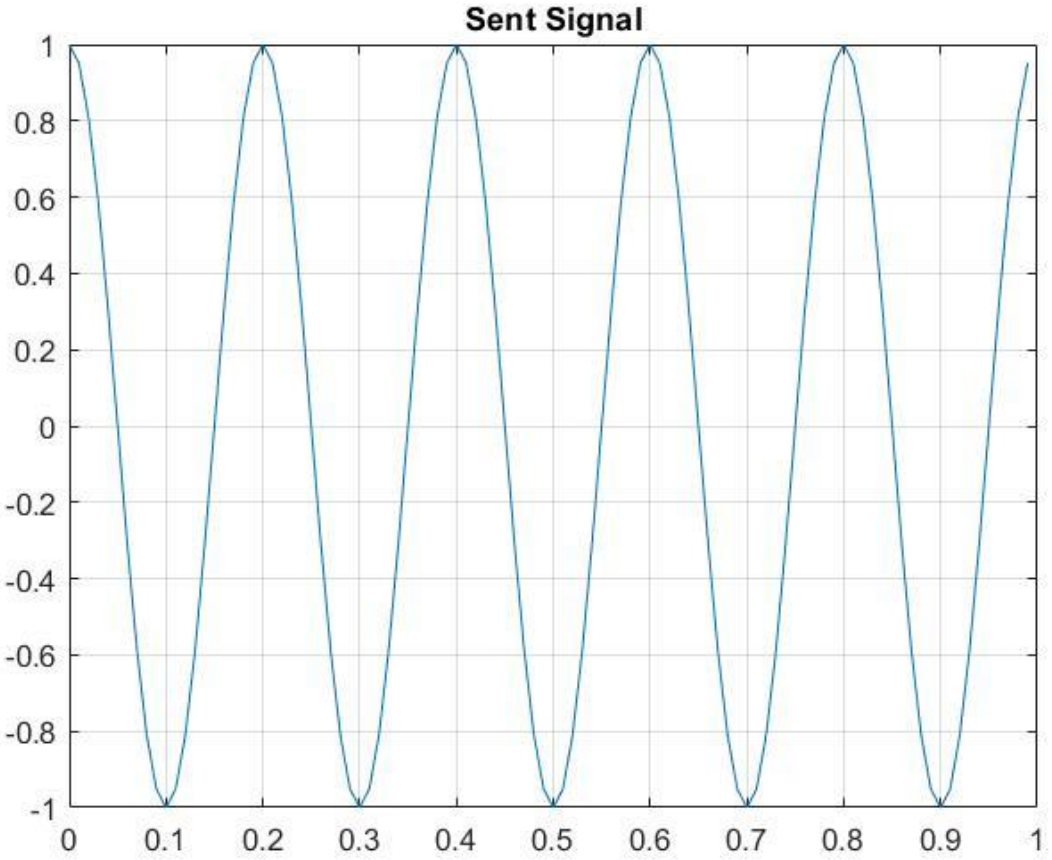
بخش اول)

1-1)

سیگنال را به وسیله کد زیر تشکیل میدهیم:

```
% p1_1 generating sent signal
tStart = 0;
tEnd = 1;
fs = 100;
t = tStart : 1/fs : tEnd - 1/fs;
fc = 5;
xSent = cos(2 * pi * fc * t);
figure;
plot(t, xSent);
title("Sent Signal");
grid on;
```

سیگنال ایجاد شده:

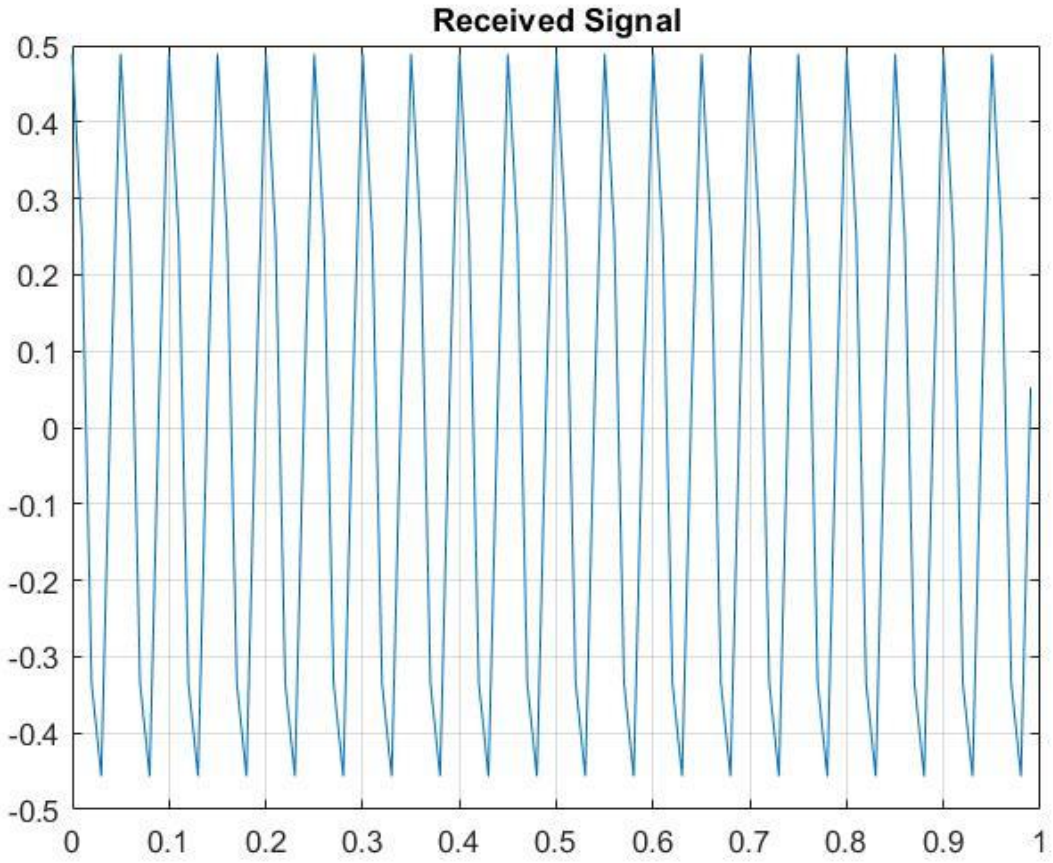


(2-1

کد مربوط به ایجاد سیگنال دریافتی:

```
15 -    v = 180 / 3.6;
16 -    R = 250000;
17 -    beta = 0.3;
18 -    alpha = 0.5;
19 -    fd = beta * v;
20 -    C = 3e8;
21 -    P = 2 / C;
22 -    td =  P * R;
23 -    receivedSignal = alpha * cos(2 * pi * (fc+fd) * (t-td));
24 -    figure;
25 -    plot(t, receivedSignal);
26 -    title("Received Signal");
27 -    grid on;
28
```

رسم سیگنال دریافتی:



(3-1

این کد، با به کارگیری تبدیل فوریه سریع (FFT)، پردازش‌های سیگنال را روی داده‌های دریافتی انجام می‌دهد و اطلاعات کلیدی را از آن‌ها می‌کشد. FFT سیگنال دریافت شده تنظیم می‌شود تا اجزای فرکانسی را در مرکز قرار دهد، و بعد از آن، مقدار و فاز سیگنال محاسبه می‌گردد. بخش فرکانسی اصلی و شاخص مربوط به آن تعیین می‌شوند که این اطلاعات، محاسبه تغییرات فرکانسی (fd) و تغییر زمانی (td) را میسر می‌سازد. کد پس از آن این تغییرات را به همراه مقادیر اساسی از پیش تعریف شده نظیر فرکانس نمونه‌برداری (fs)، فرکانس حامل (fc) و شاخص مدولاسیون (beta)، برای برآورد پارامترهای سرعت (V) و برد (R) در سیستم رادار مورد استفاده قرار می‌دهد. نتیجه، با نمایش سرعت برآورد شده به کیلومتر بر ساعت و برد برآورد شده به کیلومتر، ارائه می‌شود.

کد مربوط به پیاده سازی گفته شده:

```
29      % p1_3 finding values
30
31 -    fourier = fftshift(fft(receivedSignal));
32 -    [maxValue, maxIndex] = max(abs(fourier));
33 -    foundFd = abs(maxIndex - floor(fs / 2) - 1) - fc;
34 -    phase = angle(fourier);
35 -    foundTd = phase(maxIndex) / (2 * pi * (fd + fc));
36 -    foundV = foundFd / beta;
37 -    foundR = foundTd * C / 2;
38 -    fprintf("estimated V(km/h): %f\n", foundV * 3.6);
39 -    fprintf("estimated R(km): %f\n", foundR / 1000);
```

خروجی کد:

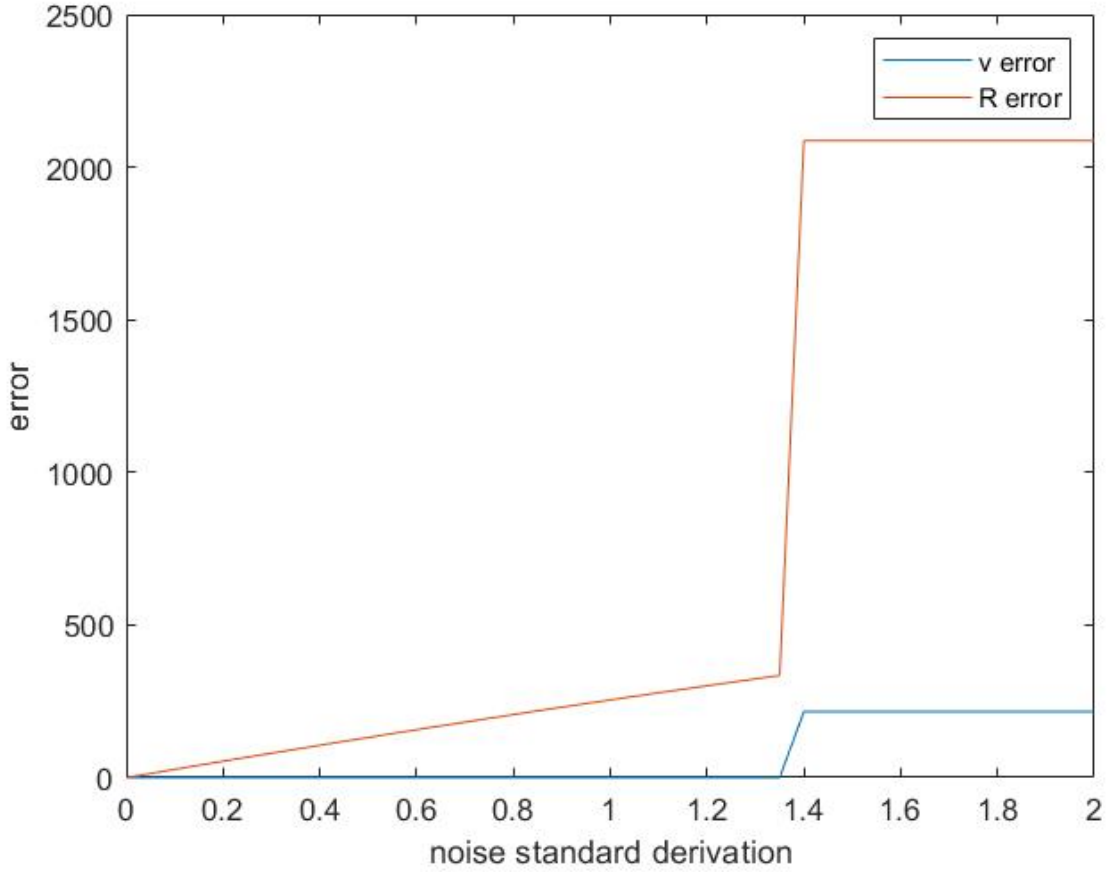
```
estimated V(km/h): 180.000000
estimated R(km): 250.000000
```

(4-1

با کمک کد زیر، در یک حلقه نویز هایی با واریانس های متفاوت را به سیگنال اضافه میکنیم و آخرین جایی که سیگنال به نویز مقاوم است را خطای زیر 10 درصد جواب بدست آمده با جواب مورد نظر تعریف میکنیم.

```
for i = 1 : length(noisePower)
    thisNoisePower = noisePower(i);
    receivedSignal = alpha * cos(2 * pi * (fd + fc)*(t - td)) + thisNoisePower * noise;
    fourier = fftshift(fft(receivedSignal));
    [maxValue, maxIndex] = max(abs(fourier));
    foundFd = abs(maxIndex - floor(fs / 2) - 1) - fc;
    phase = angle(fourier);
    foundTd = phase(maxIndex) / (2 * pi * (fd + fc));
    foundV = foundFd / beta;
    foundR = foundTd * C / 2;
    if (abs(foundV - v) < threshold * v)
        bestVNoisePower = thisNoisePower;
    end
    if (abs(foundR - R) < threshold * R)
        bestRNoisePower = thisNoisePower;
    end
    vError(i) = abs(foundV - v) * 3.6;
    rError(i) = abs(foundR - R) / 1000;
end
```

در ادامه، خطای مربوط به v و r را پلات میکنیم:



همانطور که در شکل مشخص است، پارامتر v نسبت به نویز مقاوم تر است از پارامتر r.

آخرین انحراف معیار نویزی که در آن، v و r را با خطای کمتر از 10 درصد بدست آوردیم:

```
last noise standard derivation for v is: 1.350000
last noise standard derivation for R is: 0.050000
>>
```

5-1)

سیگنال دریافتی را به شکل زیر تشکیل میدهیم(جمع دو سیگنال x1 و x2 است.)

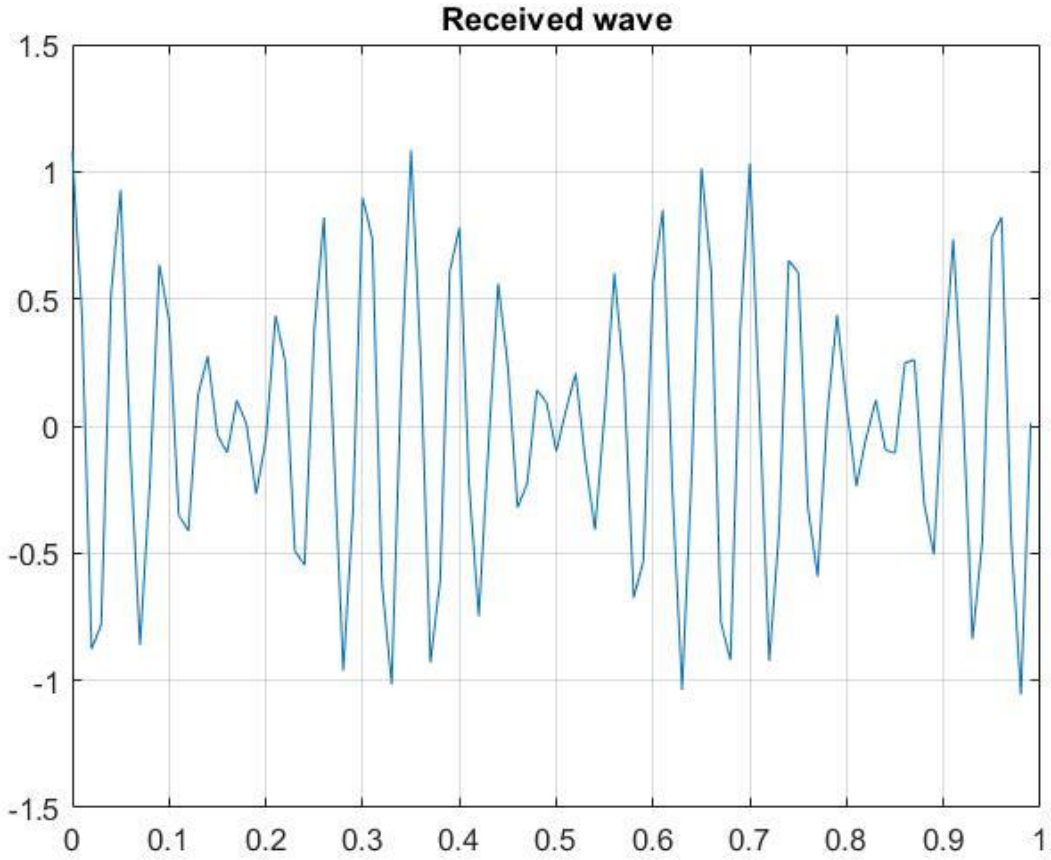
```
t_start = 0;
t_end = 1;
fc = 5;
fs = 100;
C = 3e8;
P = 2 / C;
beta = 0.3;

t = t_start: 1/fs: t_end-1/fs;
alpha1 = 0.5;
R1 = 250e3;
v1 = 180 / 3.6;
fd1 = beta * v1;
td1 = P * R1;

alpha2 = 0.6;
R2 = 200e3;
v2 = 216 / 3.6;
fd2 = beta * v2;
td2 = P * R2;

x_received = alpha1 * cos(2 * pi * (fd1 + fc) * (t - td1)) + alpha2 * cos(2 * pi * (fd2 + fc) * (t - td2));
```

رسم سیگنال مورد نظر:



6-1)

ایده پیاده سازی این است که سیگنال دریافت شده را از طریق عملیات تبدیل فوریه سریع (FFT) پردازش می‌کینم تا اجزای فرکانسی آن را تحلیل کند. پس از به‌دست آوردن دامنه و فاز سیگنال جابجا شده با FFT، کد با استفاده از تابع 'findpeaks'، اوج‌ها در طیف را شناسایی کرده و آن‌ها را به ترتیب نزولی مرتب می‌کند. در ادامه، fds، tds، Vs و Rs برای دو پیک برتر را محاسبه می‌کند. سپس نتایج، که شامل سرعت برآوردی به کیلومتر بر ساعت و برد برآوردی به کیلومتر هستند، نمایش داده می‌شوند.

پیاده سازی گفته شده به شکل زیر است:

```
28 -    fourier = fftshift(fft(x_received));
29 -    [pks, locs] = findpeaks(abs(fourier));
30 -    [pks, idx] = sort(pks, 'descend');
31 -    locs = locs(idx);
32 -    fds = zeros(1, 2);
33 -    tds = zeros(1, 2);
34 -    phase = angle(fourier);
35
36 -    for i = 1:2
37 -        fds(i) = abs(locs(2*i) - fs/2 - 1) - fc;
38 -        tds(i) = abs(phase(locs(2*i))) / (2 * pi * (fds(i) + fc));
39 -    end
40 -    vs = zeros(1, 2);
41 -    Rs = zeros(1, 2);
42 -    for i = 1:2
43 -        vs(i) = fds(i) / beta;
44 -        Rs(i) = tds(i) / P;
45 -    end
46 -    fprintf("R(1) = %f(km), v(1) = %f(km/h)\n", Rs(1) / 1000, vs(1) * 3.6);
47 -    fprintf("R(2) = %f(km), v(2) = %f(km/h)\n", Rs(2) / 1000, vs(2) * 3.6);
```

که خروجی ای کد به شکل زیر میباشد:

```
R(1) = 200.000000(km), v(1) = 216.000000(km/h)
R(2) = 250.000000(km), v(2) = 180.000000(km/h)
>>
```

(7-1

اگر سرعت دو جسم یکسان باشد، تمایز بین آنها ممکن نیست زیرا فرکانس‌هایشان (Vs) در سیگنالی که دریافت می‌کنیم، در تبدیل فوریه سریع (FFT) با یکدیگر ادغام خواهند شد. در پردازش سیگنال رادار، قابلیت تشخیص و تفکیک بین هدف‌ها به رزولوشن فرکانس بستگی دارد. زمانی که سرعت‌ها یکسان هستند، فرکانس‌ها در نتیجه FFT متمایز نیستند و برای تخمین دقیق، نیاز به تفاوت مینیمم سرعتی است که مطابق با دقت تحلیل فرکانسی باشد.

(8-1

می‌شود بین دو شی که فاصله‌شان (R) متفاوت است تفاوت قائل شد. کدی که به کار می‌رود بیشینه‌های موجود در تبدیل فوریه سریع (FFT) را برای دریافت اطلاعات تغییر فرکانس (fd) و تأخیر زمانی (td) استفاده می‌کند. این پارامترها بیشتر تحت تأثیر فرکانس‌ها تعیین می‌شوند، بنابراین تغییر در فاصله‌ها (R) زیاد روی توانایی ما برای تشخیص فاصله و سرعت اشیاء تأثیر نمی‌گذارد.

(9-1

در این قسمت، میتوانیم دقیقا مطابق قسمت 5 عمل کرده و همان استراتژی را پیاده سازی کنیم.

بخش دوم)

(1-2

ابتدا با استفاده از کد زیر، نوت ها را در یک سلول ذخیره میکنیم و در ادامه یک سلول دیگر تعریف میکنیم که در آن دیتای آهنگ مورد نظر را قرار میدهیم. به این ترتیب که هر سطر آن نام یک نوت و طول زمانی که پلی میشود قرار دارد.

```
noteNames = {'c', 'c#', 'd', 'd#', 'e', 'f', 'f#', 'g', 'g#', 'a', 'a#', 'b'};
frequencies = [523.25, 554.37, 587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880.00, 932.33, 987.77];
notes = [noteNames; num2cell(frequencies)];

fs = 8000;
T = 0.5;
tau = 0.025;

music_data = {{'d', T/2}, {'d', T/2}, {'g', T}, {'f#', T}, {'d', T},...
    {'d', T/2}, {'e', T/2}, {'e', T/2}, {'d', T/2}, {'f#', T/2}, {'d', T/2}, {'e', T/2}, {'d', T/2}, {'e', T/2},
    {'d', T}, {'e', T}, {'f#', T}, {'e', T}, ...
    {'d', T/2}, {'e', T/2}, {'e', T/2}, {'d', T/2}, {'f#', T/2}, {'d', T/2}, {'e', T},...
    {'d', T}, {'e', T/2}, {'d', T/2}, {'f#', T}, {'e', T}, ...
    {'d', T}, {'e', T/2}, {'d', T/2}, {'f#', T}, {'e', T}, ...
    {'d', T/2}, {'d', T/2}, {'e', T}, {'f#', T/2}, {'e', T/2}, {'f#', T}, ...
    {'f#', T/2}, {'e', T/2}, {'f#', T}, {'f#', T}, {'d', T}
};
```

سپس با استفاده از کد زیر، آن را تبدیل به یک آرایه میکنیم که در آن سیگنال موسیقی ما ذخیره شده است. در نهایت با استفاده از دستور sound آن را پلی میکنیم.

```
19 -    musicNotes = cell(1, length(music_data));
20 -    times = zeros(1, length(music_data));
21
22 -    for i = 1:length(music_data)
23 -        musicNotes{i} = music_data{i}{1};
24 -        times(i) = music_data{i}{2};
25 -    end
26
27 -    musicLength = (sum(times) + length(times) * tau) * fs;
28 -    music = zeros(1, musicLength);
29
30 -    for i = 1:length(musicNotes)
31 -        startIndex = round((sum(times(1:i-1)) + (i-1) * tau) * fs) + 1;
32 -        endIndex = round((sum(times(1:i-1)) + times(i) + (i-1) * tau) * fs);
33 -        index = find(strcmp(notes(1, :), musicNotes{i}));
34 -        frequency = frequencies(index);
35 -        t = (startIndex-1)/fs:1/fs:(endIndex-1)/fs;
36 -        music(startIndex:endIndex) = sin(2 * pi * frequency * t);
37 -    end
38
39 -    sound(music, fs);
```

(2-2

در این قسمت دقیقا مانند قسمت قبل عمل میکنیم با این تفاوت که به جای music_data، یک موزیک جدید ایجاد کرده و به جای آن میگذاریم.

```
 9 -    myMusicData = {
10          {'e', T}, {'d', T}, {'c', T}, {'d', T}, ...
11          {'e', T}, {'e', T}, {'d', T*2}, ...
12          {'e', T}, {'d', T}, {'c', T}, {'d', T}, ...
13          {'e', T}, {'e', T}, {'d', T*2},
14      };
```

سپس با استفاده از دستور audiowrite، آن را در فایل mysong.wav ذخیره میکنیم.(مابقی کد دقیقا مشابه قسمت قبل میباشد.

```
37 -    audiowrite('mysong.wav', music, fs);
```

(3-2

ایده ی پیاده سازی این است که صدا را با تقسیم کردن آن به قطعات بر اساس زمان هایی که مقدار 0 است(zeroSegmant) پردازش می‌کنیم. سپس هر قطعه را از طریق تبدیل فوریه (FFT) مورد تحلیل قرار می دهیم و طیف فرکانس آن را آشکار می کند. با شناسایی پیک در FFT، فرکانس غالب در هر قطعه را تعیین می کنیم. سپس، این فرکانس را با استفاده از فرکانس‌های کلیدی از پیش تعریف‌شده ذخیره‌شده در سلول، به نزدیک‌ترین کلید موسیقی نگاشت می‌کنیم. کد را برای هر تکه تکرار می کنیم که در نتیجه آن کلید موسیقی شناسایی‌شده را برای هر کدام نمایش می‌دهد.

پیاده سازی گفته شده به شکل زیر است:

```
39      % part 2_3
40 -    zeroParts = zeros(1, round(tau * fs));
41 -    for i = 1:length(musicNotes)
42 -        noteEndIndex = round((sum(times(1:i)) + i * tau) * fs);
43 -        music(noteEndIndex + 1 : noteEndIndex + length(zeroParts)) = zeroParts;
44 -    end
45
46 -    for i = 1:length(musicNotes)
47 -        noteStartIndex = round((sum(times(1:i-1)) + (i-1) * tau) * fs) + 1;
48 -        noteEndIndex = round((sum(times(1:i-1)) + times(i) + (i-1) * tau) * fs);
49 -        segment = music(noteStartIndex:noteEndIndex);
50 -        fourier = fftshift(fft(segment));
51 -        magnitude = abs(fourier);
52 -        phase = angle(fourier);
53 -        f = (-fs/2):(fs/length(segment)):(fs/2)-(fs/length(segment));
54 -        [~, peakIndex] = max(magnitude);
55 -        peakFrequency = abs(peakIndex * fs / length(segment) - fs / 2 - 1);
56 -        frequencyDifferences = abs(frequencies - peakFrequency);
57 -        [~, noteIndex] = min(frequencyDifferences);
58 -        identifiedNote = noteNames{noteIndex};
59 -        fprintf("Note: %s:", identifiedNote);
60 -    end
```

خروجی این کد به ازایی موزیک جدیدی که در قسمت قبل ایجاد کردیم:

```
Note: e
Note: d
Note: c
Note: d
Note: e
Note: e
Note: d
Note: e
Note: d
Note: c
Note: d
Note: e
Note: e
Note: d
```