Activity 9
Interpreting a Musical Score sheet

Marc Jerrone R. Castro
2015-07420

Musical Score Sheets

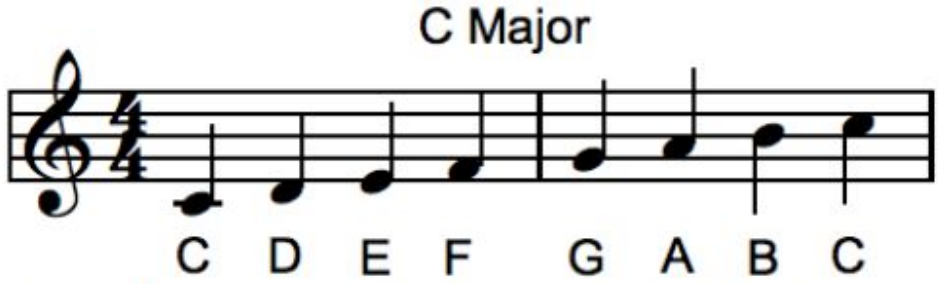

Figure 1: C-Major Scale (image from http://www.piano-lessons-made-simple.com/images/notes.png)

Musical score sheets are composed of symbols intended to symbolize or notate rhythm and tempo. It is typical composed of a five-lined staff where each space and each line represents a single note in a heptatonic (7-note) scale.

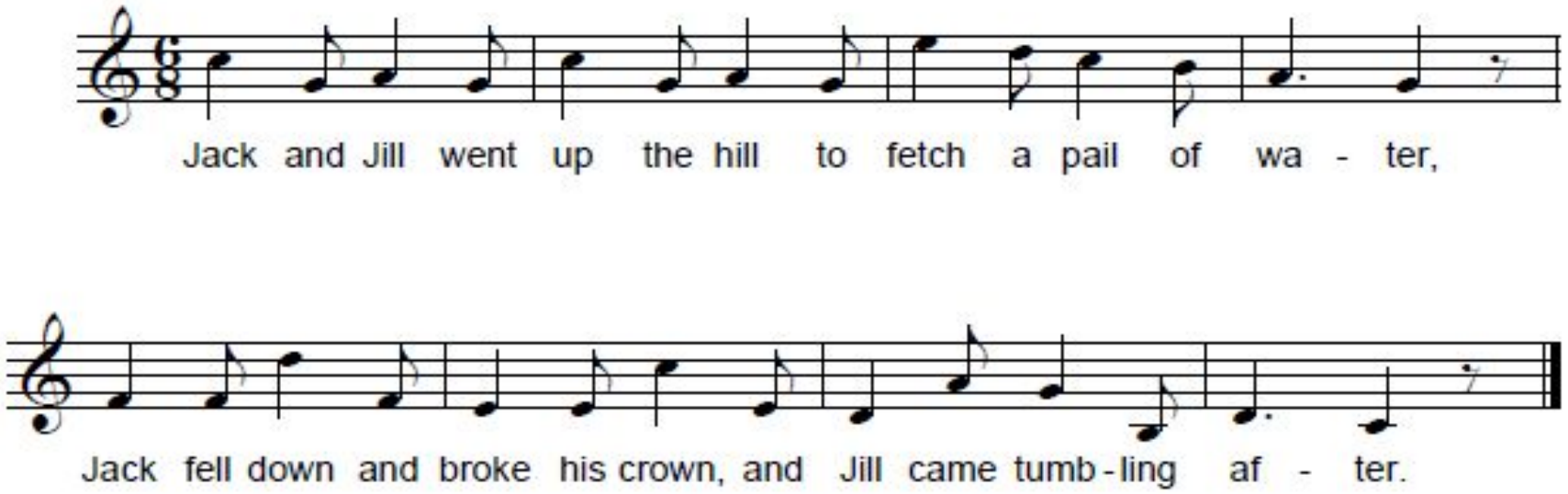

Figure 2. The following figure on the right is a music sheet for Jack and Jill. In this activity I attempt to parse the notes in the music sheet using a variety of **morphological techniques** and try to play the tune using the Scilab function, *sound*.

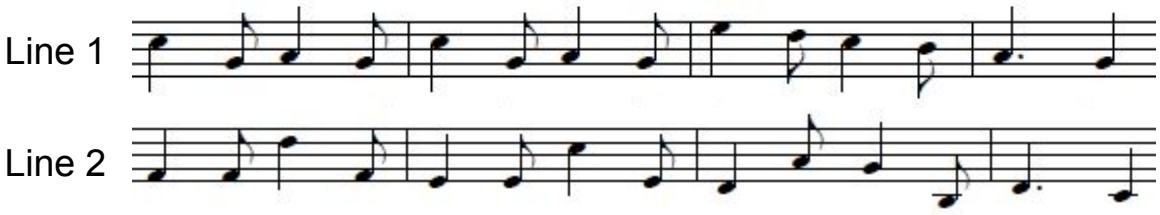

Figure 3. Parsed lines from Jack and Jill

I first manually separated the lines from the music sheet using GIMP. I opted to save the two lines separately so that I can easily apply morphological techniques to each line without the other line interfering.

Recalling from earlier lessons, I decided to parse them as backgroundless .png files as to reduce the effect of the white background for further processing and minimize the effect of the propagating errors during image manipulation. Afterwards, I perform image manipulation by converting the image into a binary image where its pixel values are constricted from 0 or 1. I chose to use a threshold of 0.2 as I found ,through trial and error, that the resulting image was clearer and more distinct for this threshold. I then used the *imcomplement* to inverse the image such that all 1s are converted to 0s and all 0s are now 1s. The resulting inverted image can be seen in Figure 4. I then performed morphological equations (OpenImage) which basically performs erosion on the image and then dilation using circle of radius 2 as its structuring element which resulted to Figure 5.

```
3 sheet = imread("A2.png");
4 sheet = im2bw(sheet,0.2);
5 sheet = imcomplement(sheet);
6 kernel_a = CreateStructureElement('circle',2);
7 sheet = OpenImage(sheet,kernel_a);
```

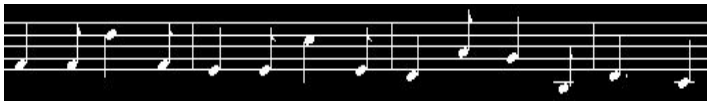

Figure 4. Inverted Image of the Second Line

Figure 5. Resulting image after perform morphological techniques on the inverted image. Only blobs of pixels remain which can then be parsed using IPD's *SearchBlobs()*.

I then used the following snippet of code which I got from a blog[1] that I found. It determines the location of the blob using the *SearchBlobs()* function and determines mean value of all its pixel values at each coordinate and assign it as the pixel coordinates of the centroid for that particular blob. We can then verify on the variable browser the locations of each blob.

```
Object = SearchBlobs(sheet);
xcoord = zeros(1,max(Object));
ycoord = zeros(1,max(Object));
for i=1:max(Object)
    [y,x]=find(Object==i)
    xmean=mean(x)
    ymean=mean(y)
    xcoord(i)=xmean
    ycoord(i)=ymean
end
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	34.3438	34.7692	17.5	34.3438	37.5	37.5	20.2308	37.5	40.2581	27.5	30.5	47.5	40.4074	44.5
2														
3														
4														

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	12.75	50.2308	78.5	116.75	153.5	190.5	222.7692	260.5	297.7419	335	370.5	408	445.5185	495.5
2														
3														

Figure 6. X (bottom) and Y (top) pixel coordinates of the centroids of each blob. These coordinates were then used to determine the specific range of pixels at which we assign certain frequencies to play.

[1] https://appphy186haroldco.blogspot.com/2016/12/activity-9-playing-musical-notes.html

```
31
32  note=zeros(1,size(y_coord,2))
33  for j=1:size(y_coord,2)
34      if y_coord(1,j)>46 & y_coord(1,j)<49
35          note(1,j)=B3
36      end
37      if y_coord(1,j)>43 & y_coord(1,j)<46
38          note(1,j)=C
39      end
40      if y_coord(1,j)>39 & y_coord(1,j)<42
41          note(1,j)=D
42      end
43      if y_coord(1,j)>36 & y_coord(1,j)<39
44          note(1,j)=E
45      end
46      if y_coord(1,j)>33 & y_coord(1,j)<35
47          note(1,j)=F
48      end
49      if y_coord(1,j)>31 & y_coord(1,j)<33
50          note(1,j)=G
51      end
52      if y_coord(1,j)>28 & y_coord(1,j)<31
53          note(1,j)=A
54      end
55      if y_coord(1,j)>24 & y_coord(1,j)<28
56          note(1,j)=B
57      end
58      if y_coord(1,j)>21 & y_coord(1,j)<24
59          note(1,j)=C5
60      end
61          if y_coord(1,j)>18 & y_coord(1,j)<21
62          note(1,j)=D5
63      end
64      if y_coord(1,j)>14 & y_coord(1,j)<17
65          note(1,j)=E5
66      end
67  end
68
```

```
20  B3 =246.94*2
21  C =261.63*2
22  D =293.66*2
23  E =329.63*2
24  F =349.23*2
25  G =392.00*2
26  A =440.00*2
27  B =493.88*2
28  C5 =523.25 *2
29  D5 = 587.33*2
30  E5 = 659.25*2
```

As mentioned earlier, we assign specific frequencies (Hz) to each musical key found within the musical sheet. Each key location was determined using the y-coordinate of its centroid and cross validating it with sample music sheets. Afterwards, we determine a certain range of error that we expect other blobs of identical frequencies to be found in. The following code was used to assign the range at which each key is assigned.

```
68
69 spacing=diff(x_coord)
70 timing=zeros(1,size(x_coord,2))
71 for j=1:size(spacing,2)
72     if spacing(j)>30
73         timing(j)=2
74     end
75     if spacing(j)<30
76         timing(j)=1
77     end
78 end
79
```

On the other hand, the timings for each of the keys was determined using the *diff()* function which basically calculates the space or the distance between consecutive keys. As a means to split half-notes and quarter-notes we apply a threshold of 30 pixels such that if the difference in between the nodes is greater than 30, then we take the timing as that of a half-note. Else, if the difference is less than 30, we take it as a quarter-note. However, although I made this precaution, the sheet music that I used for this experiment had no half notes and as such ,the keys were recorded as quarter-notes. One must also take into account the limitations of the algorithms to detect “dotted” notes as they were simply parsed out during the application of morphological transformations. As such, upon rendering the sound - it slightly deviates from the original as no “dotted” notes were registered.

```
function n = note_func(f, t)
n = sin(2*%pi*f*linspace(0,t,8192*t));
endfunction;

v=[]
for i=1:size(note,2)
    v=cat(2,v,note_func(note(1,i),(timing(1,i))))
end
sound(v,8192)
wavwrite(v,"H2.wav")
```

We then use the following snippet of code to generate the sound we desire. We utilize the sound function denoted by *note_func(f,t).*

The resulting sound files can be accessed using the following link,
https://drive.google.com/drive/folders/11AL0MUc8amXCkFPEmJ8vCI0ESTKpLtYn?usp=sharing

Self- Evaluation

Quality of Presentation = 4/5

Technical Correctness = 5/5

Initiative = 0

Total : 9/10