Assignment 1

Procedure to run the program:

After loading the program files to a particular directory type “make” and then to execute the program type the command “./omr image.png” where image can be either music1.png or music2.png or music3.png.

Design Decisions:

General Convolution:

In general convolution the kernel is flipped and iterated over the intended image. Here we multiply the pixels in the corresponding positions of the kernel and the image and finally take its sum. Based on the sum we obtain the convolved image. Here we have handled the boundary conditions by mirroring the image.

Sobel Operator:

In the initial step where we apply a general convolution on the image we handle the boundary conditions by flipping the image. Initially we decided upon using the kernel mentioned below as the Sobel operator.

$[\ [-1\ 0\ 1\ ]\,[-2\ 0\ 2]\,[-1\ 0\ 1]\ ]$

Then we decided to convert it into a set of linearly separable kernels, as:
$[\ [1]\ [2]\ [1]\ ]\ [-1\ 0\ \ +1]$

Later on, we decided to use an improved version of the kernel:

$[\ [-3\ 0\ 3\ ]\,[-10\ 0\ 10]\,[-3\ 0\ 3]\ ]$

Since it did not provide us with the desired results we reverted back to the original separable kernel.

Canny Edge Detection:

Prerequisite to the canny edge detection is to filter out any noise in the image. This step is performed using a Gaussian filter which is convolved over the image. We use a Sobel operator to detect the edge strength. Following this we find the edge direction. Once the edge direction is known we use that to suppress non maximum pixels.

Hough Transform:

Here we consider the parameters in terms of 'r' and 'θ'.On detection of horizontal lines we group the lines which are very close to each other by taking the average. This was tricky because we got many lines in the output. We had to write a post processing step to combine all the lines if they were close enough.

Angle limitation in Hough Transform:

Since our objective was to detect only the horizontal lines we limit the angle to a very small threshold. In our case we have set the threshold between 0 and 0.001. This gives a significant speed improvement.

Staff Height Estimation:

Through Hough Transform we have detected the staff lines of the given image. Now, in order to know the staff group (trebel / bass), we do the following:

- Find successive difference of the intercepts. Pick the least 5 distances, and average them. This gives the staff height.
- Iterate over all the intercepts in a sorted fashion. Put each intercept in a different group if the corresponding difference is more than 3 times the staff height.

The above steps give us the approximate staff height, and the groups of the staffs.

Template Rescaling

The above step gives us the approximate staff height. The templates provided to us have a staff height of 11 px. We can now scale proportionately by picking the proper image from the library generated by the below process.

We used ImageMagick's command line utility to quickly generate us around 250 rescaled templates. We scripted it out. :-)

```
$ seq 50 200; while read line; do convert -resize “$line%”
templates/template-”$line”.png; done
```

We then dynamically load the templates as per the required scale.

However, we found out that for an image that requires 109% scale, “template1-100.png” performs better than “template1-109.png”. So we reverted to the nearest multiple of 10.

Template Matching using Hamming Distance:

Initially we iterate the kernel which in our case is the template over the image thereby performing convolution. Next we invert the image and the kernel and iterate the inverted kernel over the inverted image. Finally we add the result obtained from both. Based on the template to be detected we set the threshold. We have normalised the image (ie. divide every pixel by maximum pixel in the image), and then set a threshold of 240.

However, we saw that some of the templates perform better with threshold = 240, and some give false positives. We therefore had to implement per-template threshold.

Non Maximum Suppression

We saw through the process of implementation that some of the non-maximum (non local maxima) were getting detected because they were above the threshold. We had to implement a simple non-maximum suppression algorithm to overcome this problem.

Template Matching using Edge Map

We have implemented a naive O(n^4) approach. The downside of this implementation is that it takes a lot of time. This can be optimised by using dynamic programming. Nearest distance to each pixel can be represented as:

D[x][y] = min(min(1 + distance-neighbor(i)), min(sqrt(2) + distance-neighbor(j)))

where i is the index into a list of adjacent neighbors, and j is the index into the list of corner neighbors.

Accuracy:

The program runs well for music1.png and music4.png where all symbols are detected but the result for music2.png and music3.png is not as accurate as its preceding images.

When it works well:

Our code works exceptionally well for ideal images having very little noise and the staff lines being aligned horizontally. In this case the template symbol gets perfectly detected wherever it is present on the given image. Our code compiles and generates the images in comparatively less time.

Drawbacks:

Some of the drawbacks we encountered are as follows:

1. On scaling the templates the image quality is reduced and thereby during matching sometimes a few instances of the template image go undetected or a few times some of the non template symbols also get detected as an instance of the template under consideration.

2. When we obtain the edge map after Canny edge detection few of the edges of undetected and thereby when we try to detect the the templates in the image a few of the template images go undetected in the primary image in spite of the image in the template being present on the primary image.

3. There was an instance in ‘music3.png’ where the staff lines were not horizontally aligned during which we faced some problems detecting the hough lines. Thereby resulting in failure to detect the some notes when present in the image.

4. When the image provided is noisy even after filtering there were instances where symbols apart from the template symbols were detected. This sometimes even included numbers being perceived as the template symbol.

Further improvements:

Although our code works exceptionally well on ideal images there is scope for improvement. Some of the areas where we can improve our code are as follows:

1. Optimise the code such that it takes much less time to compile and generate the images especially in the edge based template detection.

2. Make it work well even on noisy images and where certain staff lines are not horizontally aligned.

3. Improve the accuracy when the templates are scaled.