mbirla, mselli, rmeda

ASSIGNMENT 1

Final Report

Introduction

In order to solve the problems proposed in this assignment, we have split the main file a1.cpp into two different files: a1_utils.h and a1.cpp. The first file contains all the functions developed and implemented to carry out each of the problems of the assignment. The second file contains the calls to each of the functions needed for every problem. In the main function of a1.cpp, the uncommented lines are there to solve problem 7. The calls to the rest of the problems remain commented.

Convolution

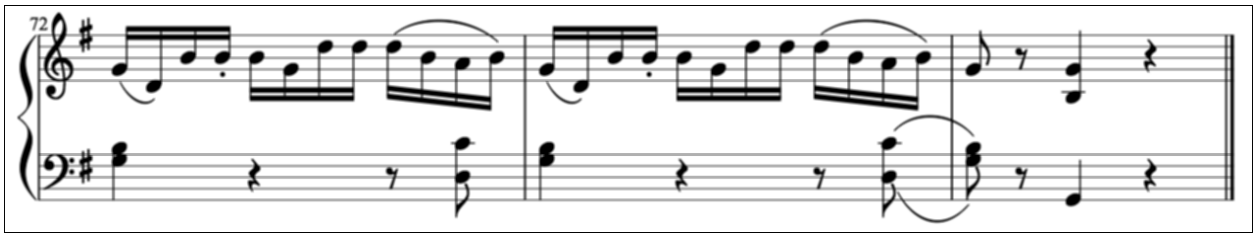
To apply convolution 2D in the images, we implemented a simple algorithm with two for loops that calculate the new value of the pixels given a specific template.

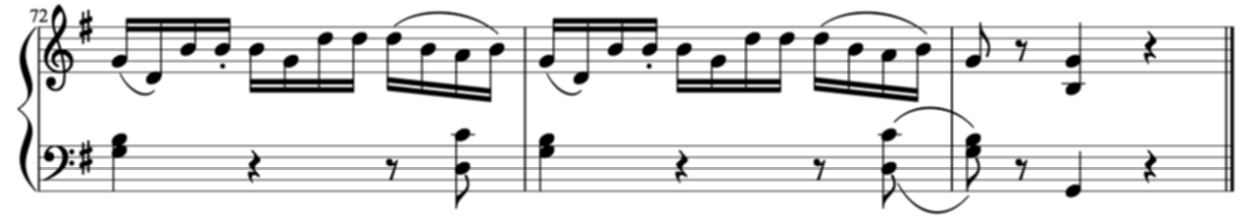
To apply convolution in 1D, we have to create two kernels Hx and Hy, where H = Hx*Hy. The procedure in this case is very similar to convolution 2D, but the matrix boundaries are restricted for one dimension at the time only.

In both cases, images have to be normalize after applying convolution.

Since the borders of these images are all white, we can do reflection on the edge for border correction to give good results. We implemented a function reflect, which will give the reflection of the points that are outside the border.

We can see the difference between the convolution operation before and after applying border correction.





mbirla, mselli, rmeda

ASSIGNMENT 1

Template Matching

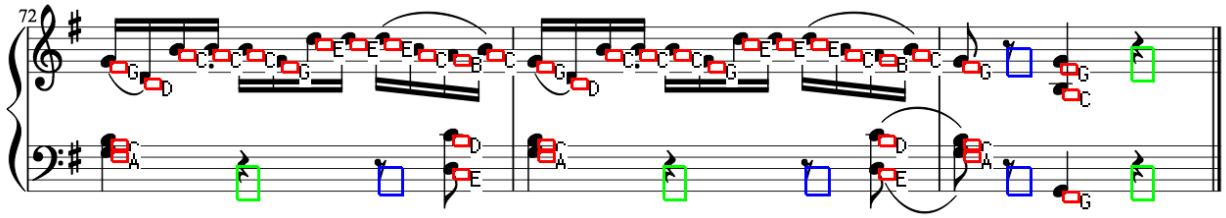
The aim of problem 4 is to automatically detect the location of a certain template on an image, if it exists. Specifically, the OMR system should be able to locate different musical symbols and estimate the pitch of the note heads, when necessary.

To solve this problem, given an image I and a template T, we calculate the distance between the two images and when the distance value in a pixel is below some threshold, it means that the template was found. Since there might be more than one value that satisfy the threshold condition, we have to apply non maximum suppression to keep only one value for each template. This task is carried out by function detectTemplate().

After we have the location of all the points founded in the image, we need to estimate the pitch of each note heads. Depending on the location of the note heads with respect to the staff lines, you can easily find the pitch for a specific note head. This is taken care by function findPitch().

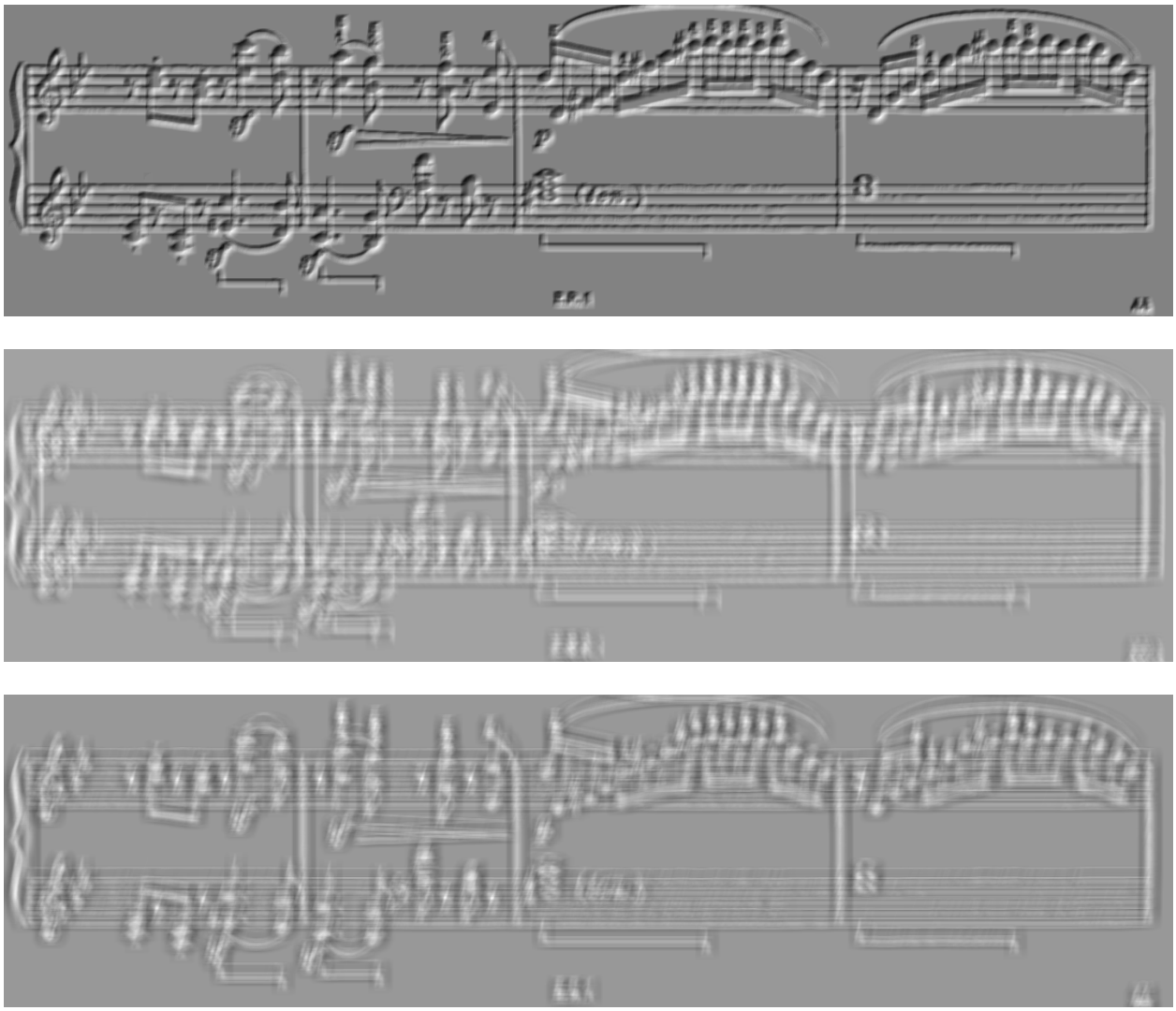
In the following images, you can see how the brightest points are remarking the founded symbols for every template.

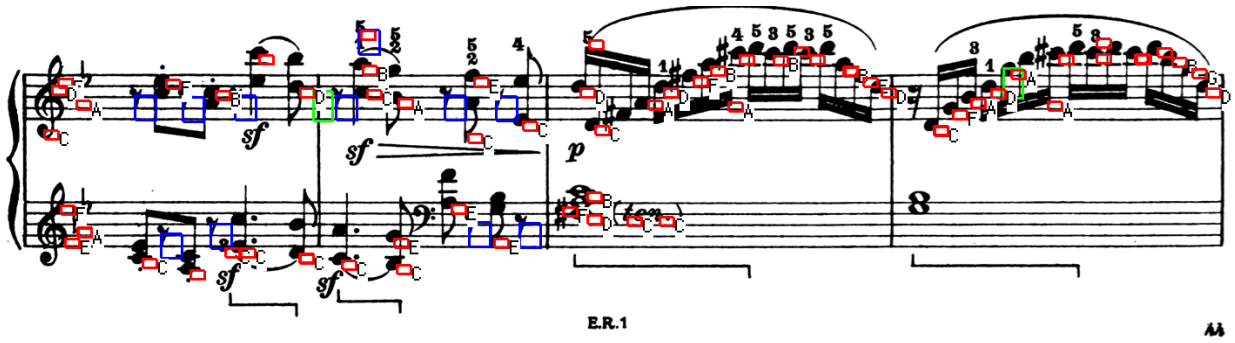
Results for music1.png



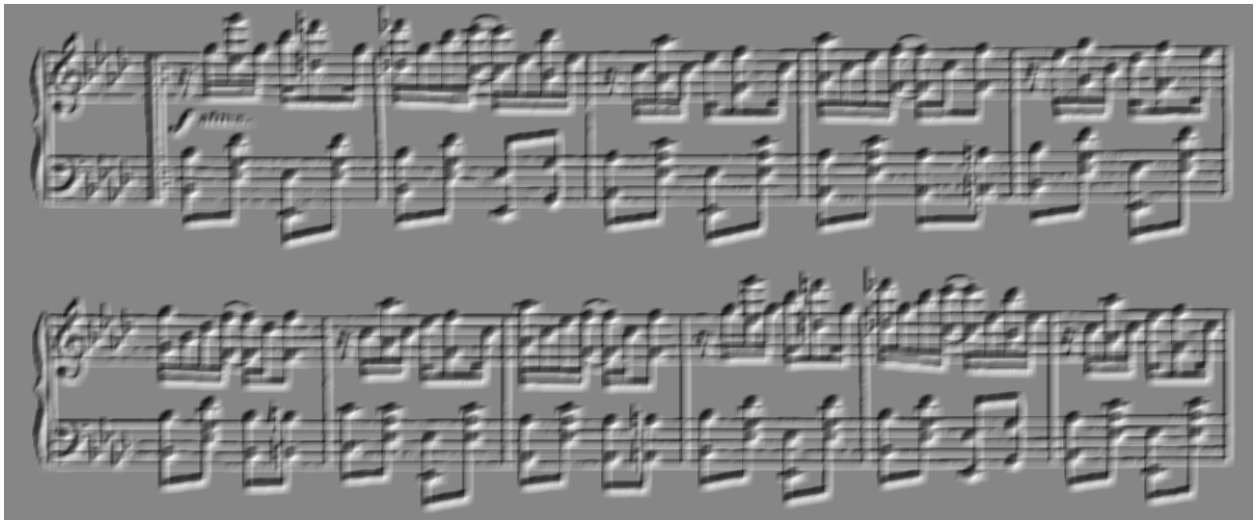
As you could see, the results are very clear and almost the same as the original image. But for some cases this process does not work in the same way. This is mainly because of the size of the templates and the definition of the image. If the image is blurry, for example, then it will be more likely to match a template with a blurry zone of the image.

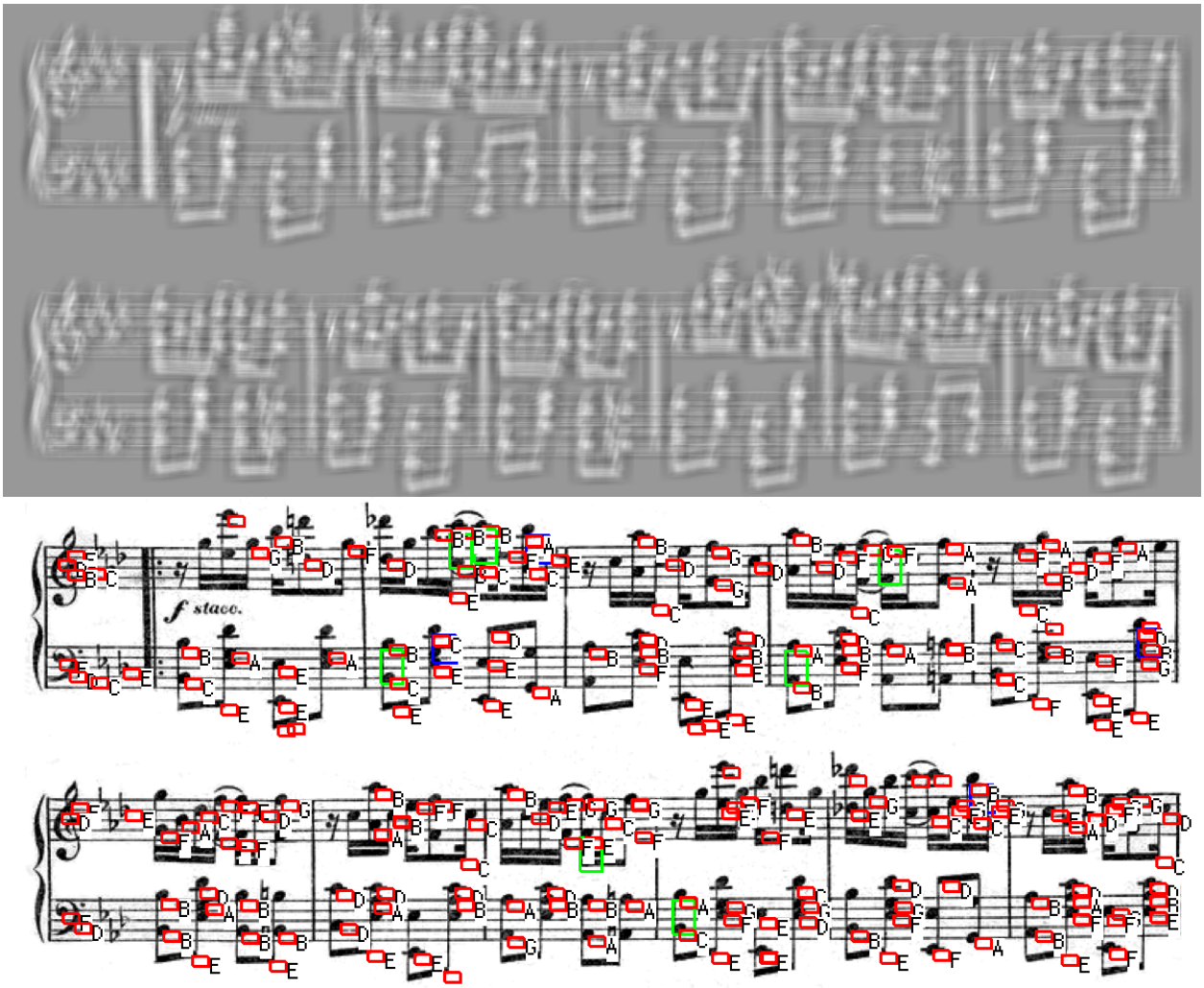
Results for music2.png



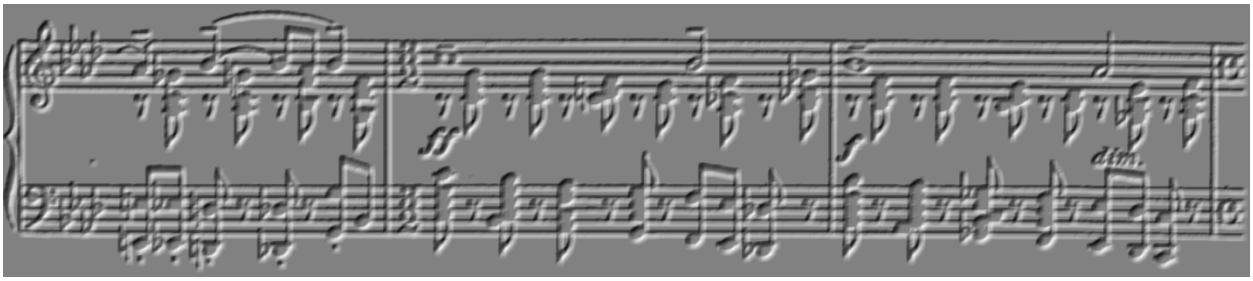


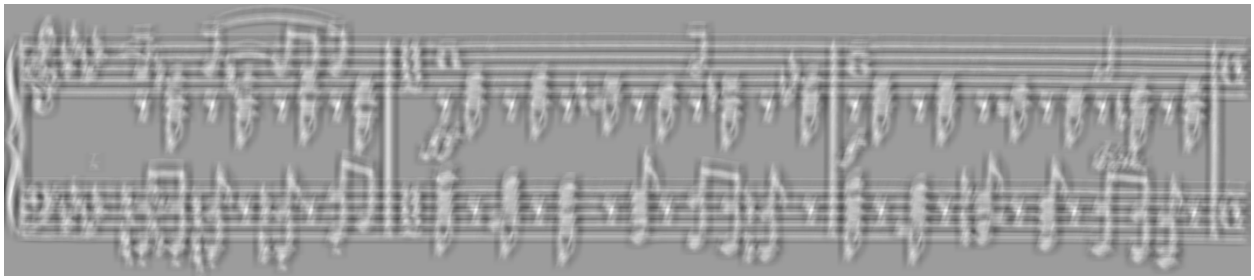
Results for music3.png

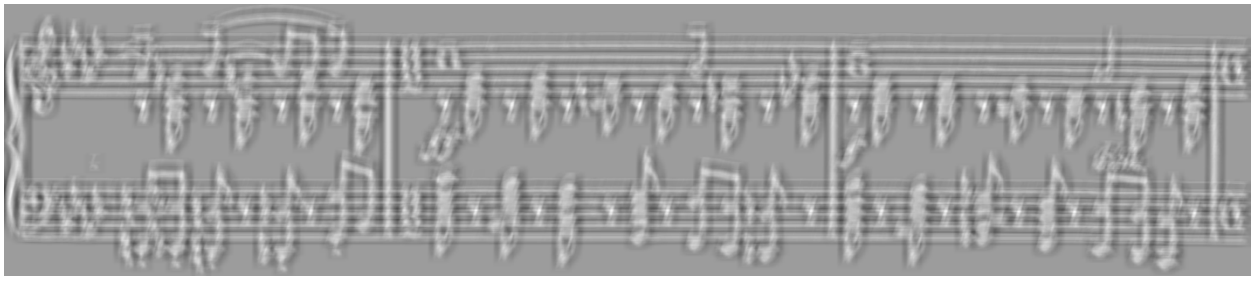


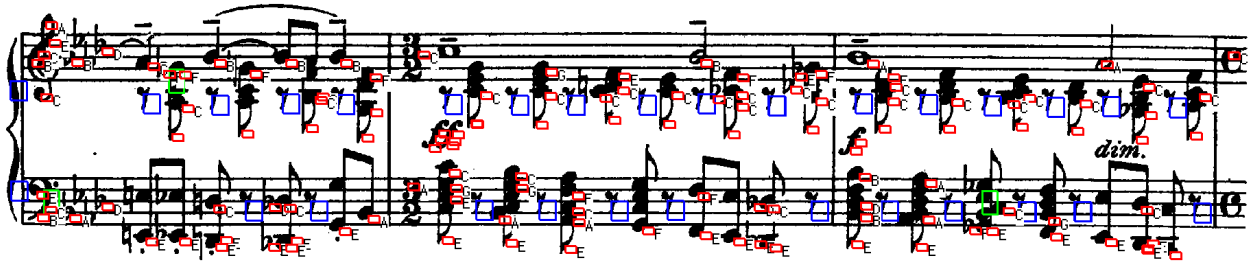


Results for music4.png









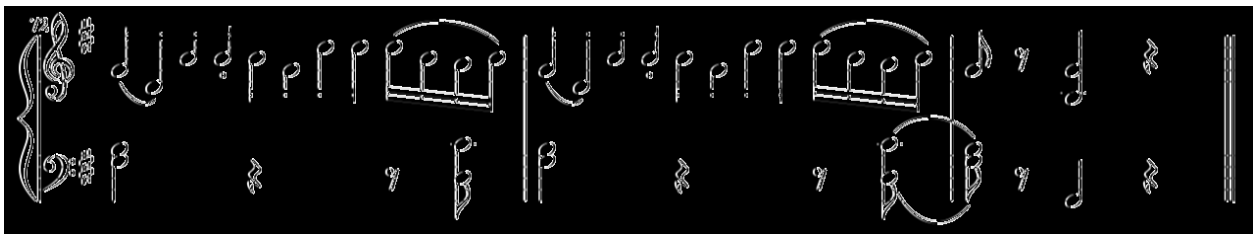
Sobel Edge Detection

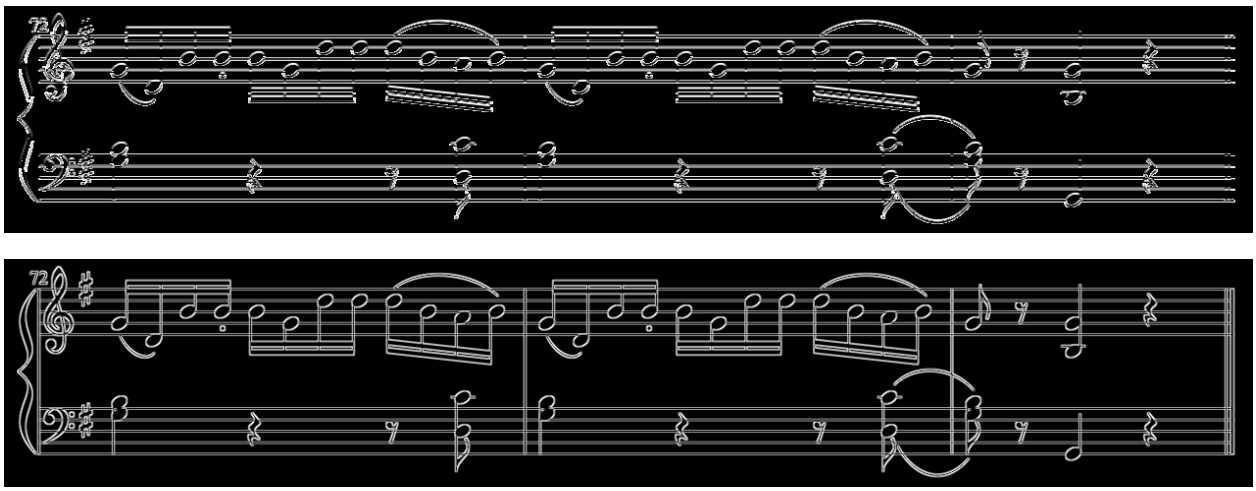
First we apply separable convolutions using two sets of filters, one for calculating horizontal gradient Gx, and another calculating vertical gradient Gy. We use these following filters respectively.

- Vertical component of Sobel operator - col vector = {1, 2, 1}, row vector= {-1, 0, 1};
- Horizontal component of Sobel operator - col vector= {-1, 0, 1}, row vector = {1, 2, 1};

Then we calculate magnitude at each point G = abs(Gx) + abs(Gy) to get a Sobel edge image. We normalized the image dividing the magnitude of each pixel with the maximum value in the image and multiplying it with 255.

These are the outputs for music1.ong we got after implementing the edge detection. Gx, Gy, Sobel edge image in the order.





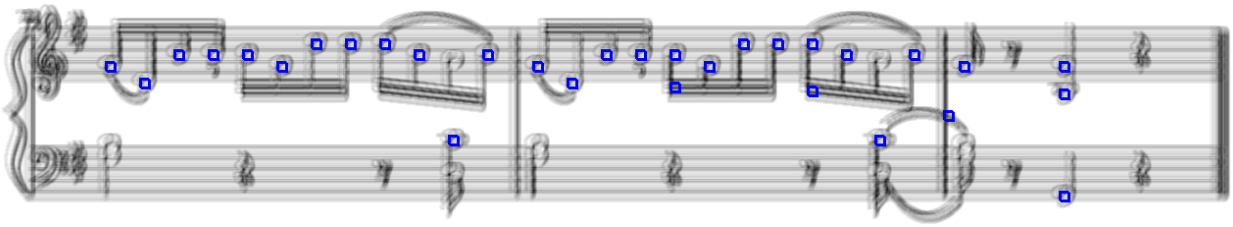
Improvements:
Using non maximum suppression we can reduce the Edge Thresholding.

Distance Transform Function

First we generate the edge maps of image and templates using Sobel edge detection are then these edge maps are converted into binary images containing only 0 and 255 values.
We applied naïve method to calculate the Euclidian distance transform. We loop on each point in the image and calculate distance between this point and all the points in the image that have a value of 255. We keep track of the minimum of these distances and assign its value to the point.

Template matching using Distance Transform

We use the cross correlation operation on the image we got from distance transform function with the binary template images we generated earlier. Then we use template matching function to detect all the nodes and generate the image with the detected nodes.
Here is an example of detecting the template1.png in music1.png image.



mbirla, mselli, rmeda

ASSIGNMENT 1

Hough Transform

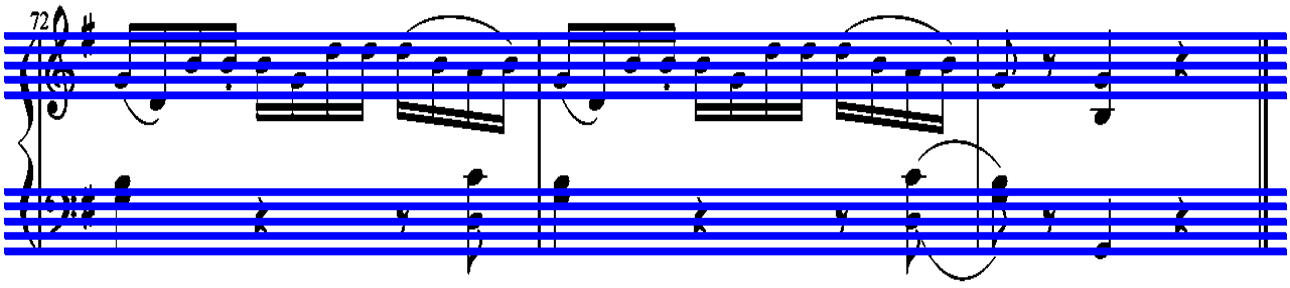
As professor mentioned in the assignment that rather than using standard accumulator, you can use the accumulator where row signifies the row coordinate of first line of staff and column signifies the distance between the staff. The final Image is houghLines.png
So, as stated above in our code follows the following steps with some modifications.

1. It processes each pixel and it assumes itself as the first staff line and it votes for the 4 staff lines it has below it.
2. So now when code is looping over all the rows below it find the first white pixel and then finding the distance between this pixel and the white pixel to get the distance which becomes the column value of our accumulator and incrementing the vote.

Now in the above steps, we observed that the lines which were not the first staff lines were getting high votes as compared to the below line. So, for that we flipped the image vertically and ran the above process again. Lastly we combine these two images to get the final output

Function Name and their Usage:

1. getFinalAccumulatorOutput :- This function creates accumulator array.
2. hough_transform :- Performs the hough transform on both images
3. FlipImageVertically:- Flips the images vertically

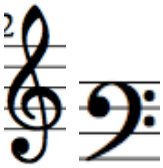


OMR System

For problem 7, we used the detection from step 4 with step 6 as the input to it which is a Hough transform that gives us the starting staff line and the distance between the lines of the staff. We used the methodology of problem 4 because the output was more accurate than with the algorithms used in problem 5 overall.
Is worth mention that some images don't have the same distribution of staff lines. Ones only have one pair of staff lines and other might have two pairs of them. Also, in some cases, the pair of staff lines are not representing by a Treble and a Bass pair, they could have two consecutive Trebles, like in music2.png for example. The problem with this, is that the pitch of the notes changes according these symbols. You can see this situation in the following images.

In order to tackle this situation, for each staff lines detected by Hough Transform in problem 6, we try to see if there is a Treble symbol or a Bass symbol using the function detectBass() that uses the method implemented in problem 4.
For this, we have defined a treble template (template_treble.png) and if the system can find it (in the same way templates 1, 2 and 3 were found before), then the pitch of the notes within that staff lines will be different as if there were a Bass symbol.
Since the template is not being modified for every image (the same template is used for all the images), in some cases the detection does not perform very well because these symbols are not exactly the same for all the images, as you could see in the following images.

music1.png



music3.png

music4.png

The code for this part is commented (lines 282 to 292 in a1.cpp file) and another solution (with some assumptions made) was added instead. We are assuming that for every image, the first staff lines are for Treble, the second for Bass, the third (is it exists) for Treble, and so on, and the output pitch for every note head is set based on this assumption.