Project 1 Report

CMPEN 454
Professor Collins
Due: 10/7

a) Summarize in your own words what you think the project was about. What were the tasks you performed; what did you expect to achieve?

In this project, we need to generate both Gaussian and Laplacian pyramids for a given image, where Laplacian pyramids were derived by calculating the differences of two adjacent layers of the Gaussian pyramids (DoG). From there, we also need to blend two source images together with a mask indicating which part of the source images to blend into the output result.

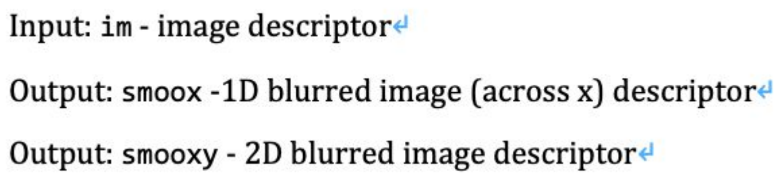
In order to do that, first we need to generate both Gaussian and Laplacian pyramids of two source images of the same size. Then we also need to generate a Gaussian pyramid for the binary mask. By completing those actions listed above, we can then generate a Gaussian pyramid and a Laplacian pyramid for the blended images. Therefore, we got a blended lowest resolution image at the end of the Gaussian pyramid. Moreover, we were also able to recover the blended image that is similar to the original image (located at the top of each pyramid) from the lowest resolution image (located at the bottom of each pyramid) by calculation along with the blended Gaussian pyramids and Laplacian pyramids.

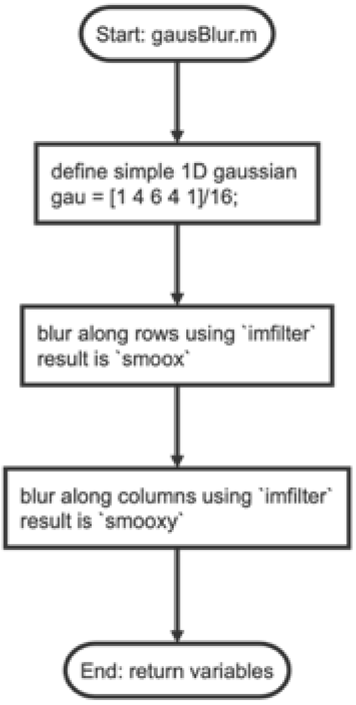
The basic outcome we expected to achieve is to blend the two original images into one image, and downsample it into low resolution image along with the pyramids. In addition, we were also expected to recover the blended images back to the original scale along with the pyramids that we generated previously.

b) Present an outline of the procedural approach along with a flowchart showing the flow of control and subroutine structure of your Matlab code. Explain any design decisions you had to make. For example, what data structure did you use to represent Gaussian and Laplacian pyramids? Did you make any modifications to the basic algorithms presented in the Crowley or the Burt and Adelson papers?

Our Mathlab code consists of four parts that are shown as below. In the main function (`project1.m`) called three subroutines, which are `buildPyra.m`, `gausBlur.m` and `recoverImg.m`.

`gausBlur.m: used for blur the images by Gaussian filters`


Input: im - image descriptor
Output: smoox -1D blurred image (across x) descriptor
Output: smooxy - 2D blurred image descriptor



The flowchart of `gausBlur.m`.

`buildPyra.m: used for generating both Gaussian and Laplacian pyramid.`

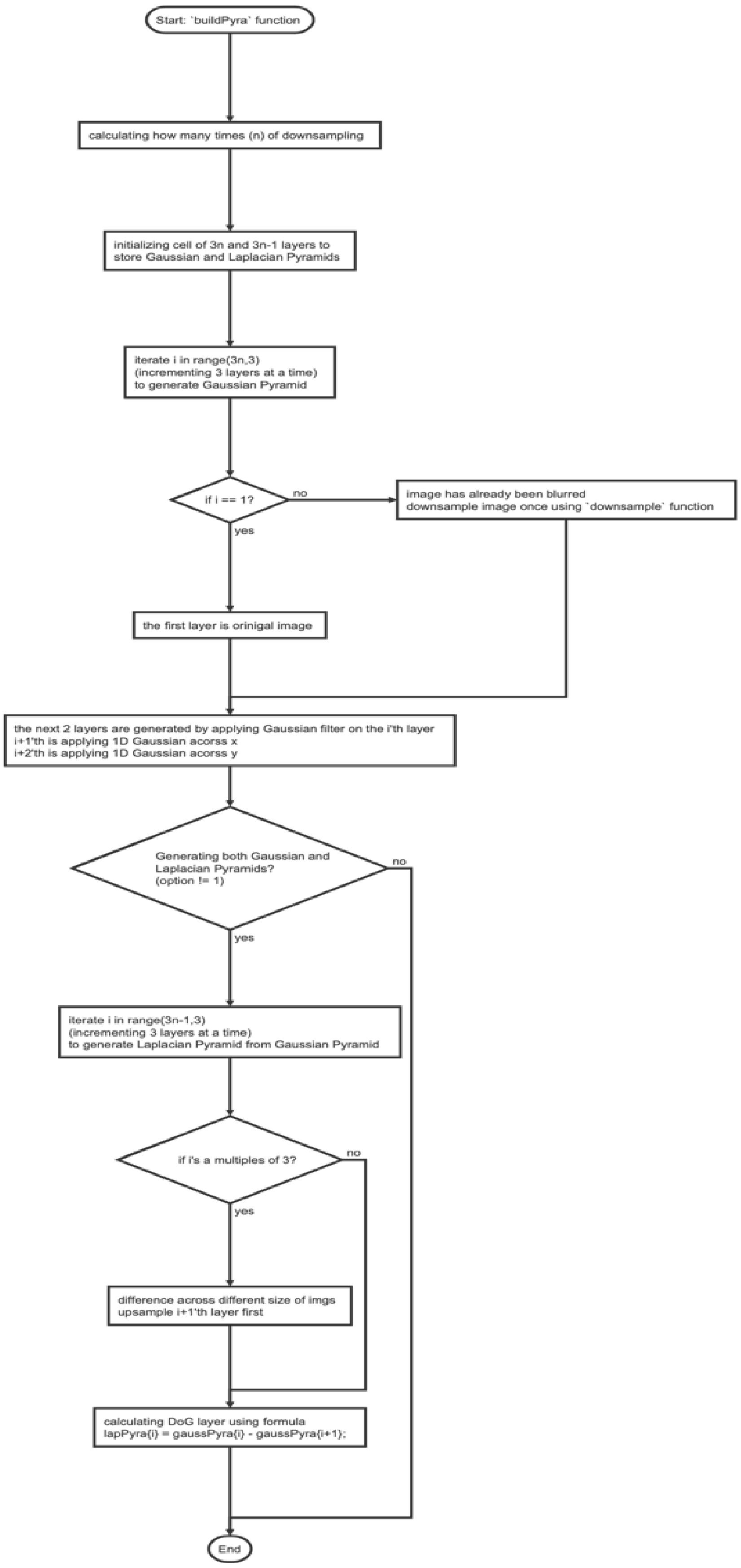
Input: `im` - image descriptor

Input: `k` - threshold pixel to stop downsampling

Input: `option = 1` for generating Gaussian Pyramid only

Output: `gaussPyra` - cell array for Gaussian Pyramid

Output: `lapPyra` - cell array for Laplacian Pyramid


Start: `buildPyra` function
calculating how many times (n) of downsampling
initializing cell of 3n and 3n-1 layers to store Gaussian and Laplacian Pyramids
iterate i in range(3n,3) (incrementing 3 layers at a time) to generate Gaussian Pyramid
if i == 1?
no
image has already been blurred downsample image once using `downsample` function
yes
the first layer is orinigal image
the next 2 layers are generated by applying Gaussian filter on the i'th layer i+1'th is applying 1D Gaussian acorss x i+2'th is applying 1D Gaussian acorss y
Generating both Gaussian and Laplacian Pyramids? (option != 1)
no
yes
iterate i in range(3n-1,3) (incrementing 3 layers at a time) to generate Laplacian Pyramid from Gaussian Pyramid
if i's a multiples of 3?
no
yes
difference across different size of imgs upsample i+1'th layer first
calculating DoG layer using formula lapPyra{i} = gaussPyra{i} - gaussPyra{i+1};
End

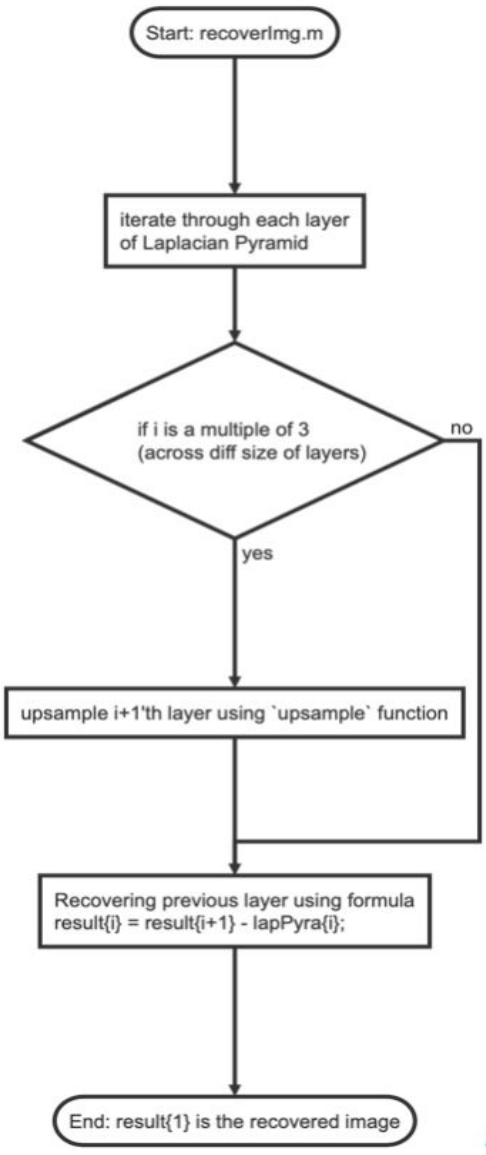
The flowchart of `buildPyra.m`.

recoverImg.m: given the Laplacian pyramid and low resolution image, the recoverImg.m was used to recover the image bock to the original image descriptor.

Input: `gausLstLyr` - the smallest image / layer of the Gaussian Pyramid

Input: `lapPyra` - generated Laplacian Pyramid

Output: `img` - recovered original image descriptor matrix



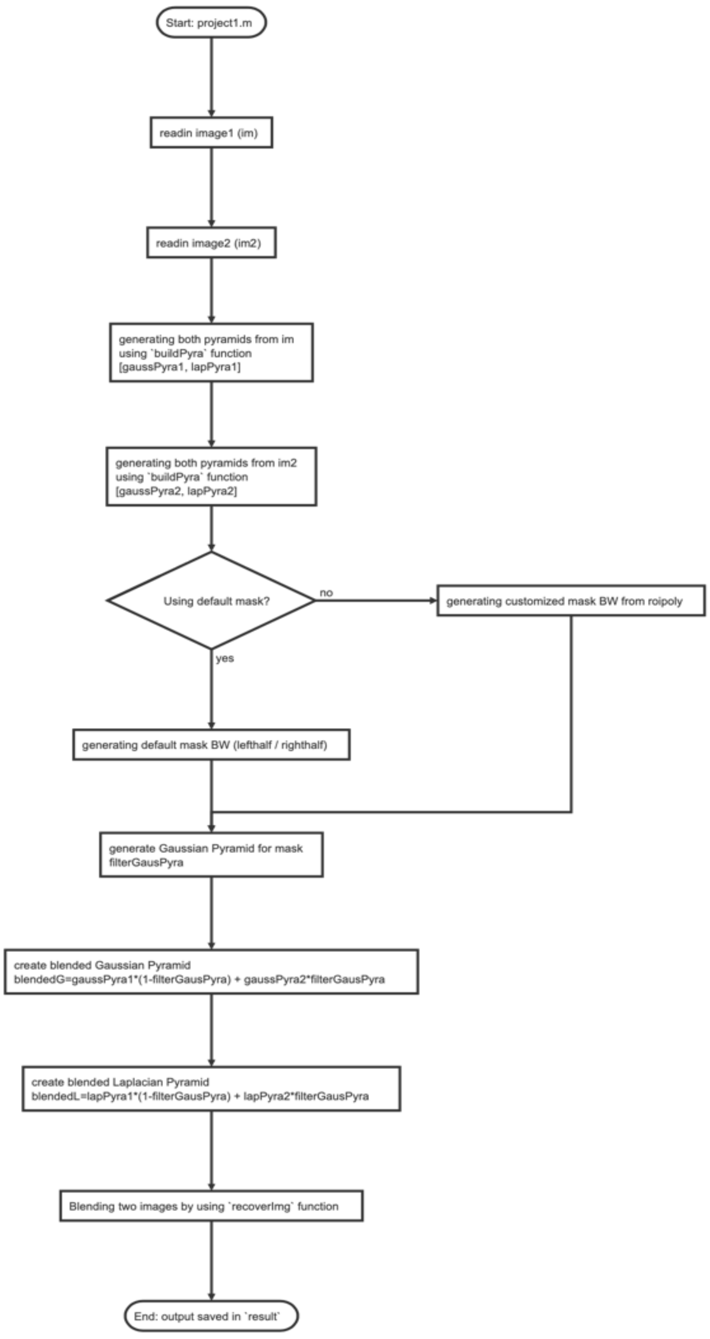
Start: recoverImg.m
iterate through each layer of Laplacian Pyramid
if i is a multiple of 3 (across diff size of layers)
no
yes
upsample i+1'th layer using `upsample` function
Recovering previous layer using formula result{i} = result{i+1} - lapPyra{i};
End: result{1} is the recovered image

The flowchart of `recoverImg.m`.

project1.m: the main function of the code, called three functions to blend the two images

Input: N/A. Specific path to two images in the program.

Output: Blended image matrix `result`



Start: project1.m
readin image1 (im)
readin image2 (im2)
generating both pyramids from im using `buildPyra` function [gaussPyra1, lapPyra1]
generating both pyramids from im2 using `buildPyra` function [gaussPyra2, lapPyra2]
Using default mask?
no
generating customized mask BW from roipoly
yes
generating default mask BW (lefthalf / righthalf)
generate Gaussian Pyramid for mask filterGausPyra
create blended Gaussian Pyramid blendedG=gaussPyra1*(1-filterGausPyra) + gaussPyra2*filterGausPyra
create blended Laplacian Pyramid blendedL=lapPyra1*(1-filterGausPyra) + lapPyra2*filterGausPyra
Blending two images by using `recoverImg` function
End: output saved in `result`

The flowchart of `project1.m`.

Within the Matlab code, the data structure we used to represent the Gaussian and Laplacian pyramids is `cell`, which holds an array of matrices and each element corresponds to a layer of the pyramid.

We modified the algorithm for the pyramid by adding the calculation between every layer, including those of different sizes, through upsampling. By doing that, we got the more completed Gaussian and Laplacian pyramids.

c) Image coding milestone. What do you observe about the behavior of your code for pyramid decomposition and image regeneration? Does it seem to work the way you think it should? For a sample image, can you regenerate the original image after decomposing it into a low-res Gaussian and a Laplacian pyramid? Show some intermediate results of the process, such as images from different levels of the Gaussian pyramid and the Laplacian pyramid.

During this project, we initially encountered difficulty of not able to successfully upsample the image. The solution turns out to be the incompatibility of `imagesc` with image matrix of `double` type and it was solved by manually converting the matrix to type of `uint8`.

From there, a bigger problem was encountered, where after we attempted to regenerate the original image back from the pyramids, the result was drastically different from our source image. After sometime of debugging, we identified the problem to be the loss of accuracy from `double` to `uint8` during the process. So we further changed the problem to use `double` throughout all the calculations and only convert it to `uint8` in the last step before function returns. Our program works smoothly and is giving out our expected results after these problems were fixed.

Below are some images showing the intermediate results while generating the pyramids.

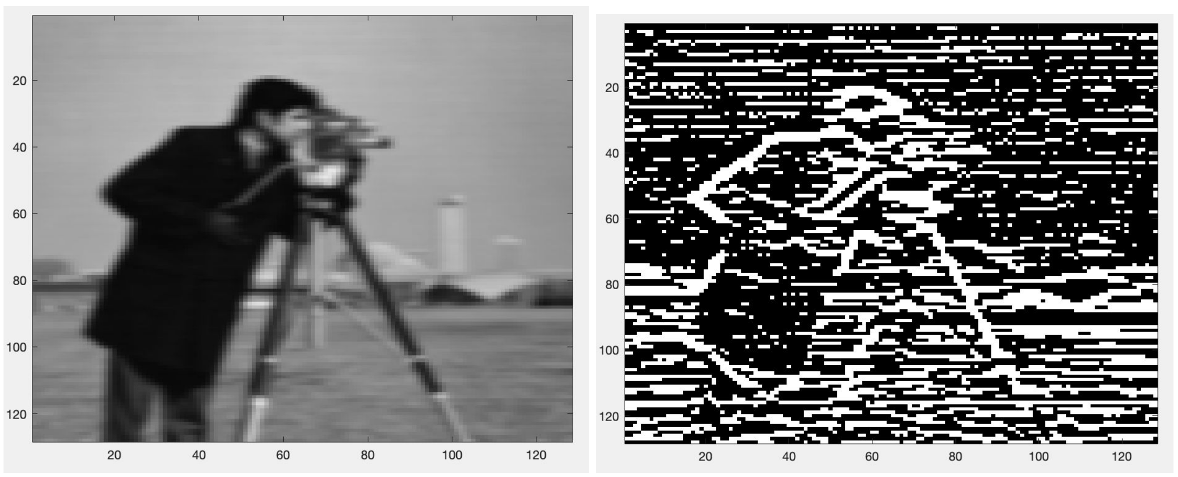


Image c1 - Gaussian and Laplacian result at layer #5 (of 15)

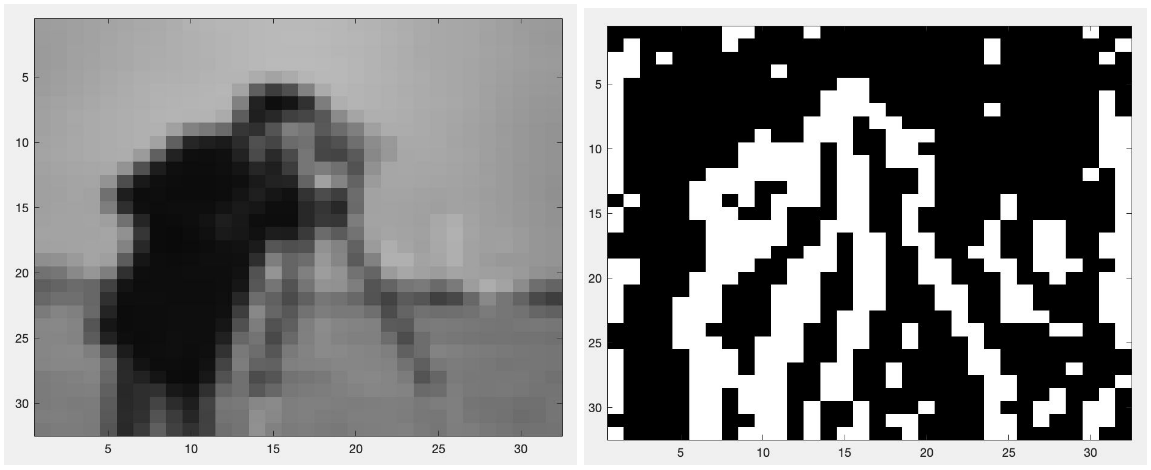


Image c2 - Gaussian and Laplacian result at layer #10 (of 15)

d) Experimental results. Show examples output by your blending application – the images chosen, the binary region mask, and the resulting blended image. Do some examples work better than others? How did you choose images to use for the blending application? How did you generate a binary mask image? Did you develop any rules of thumb by trial and error to tell you what to look for when choosing images that would be good to blend?

We chose the cameraman image from class as our first source image. We flipped the image horizontally as our second image. Image below shows a preview of our two source images using `imagesc`.

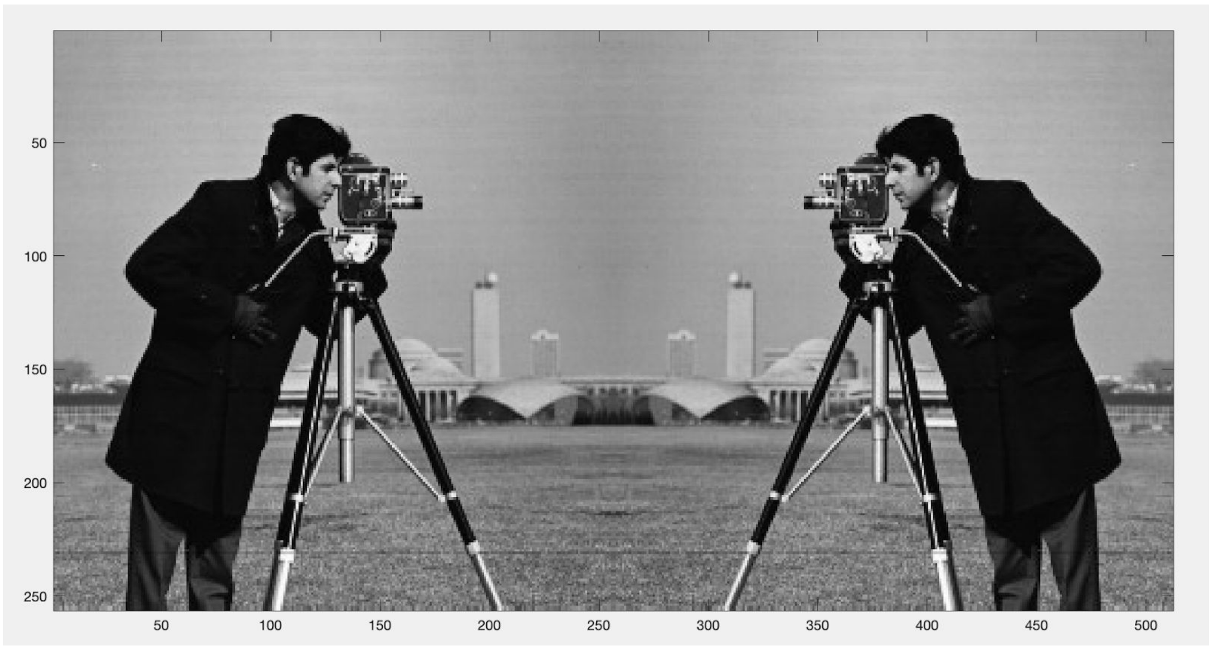


Image d1 - Source Images `im` and `im2`

Our default binary mask takes left half of the first source image and the right half of the second image. In our program we also provides an option where users can generate their own mask using `roipoly` function. We generated the default mask by first getting the width and height of the source image(s) `xsize` and `ysize`, then we separately generate two matrices of ones and zeros width of `xsize/2` and height of `ysize` and then combine these two matrices to form the final mask matrix. Image 2 below shows a preview of our default mask.

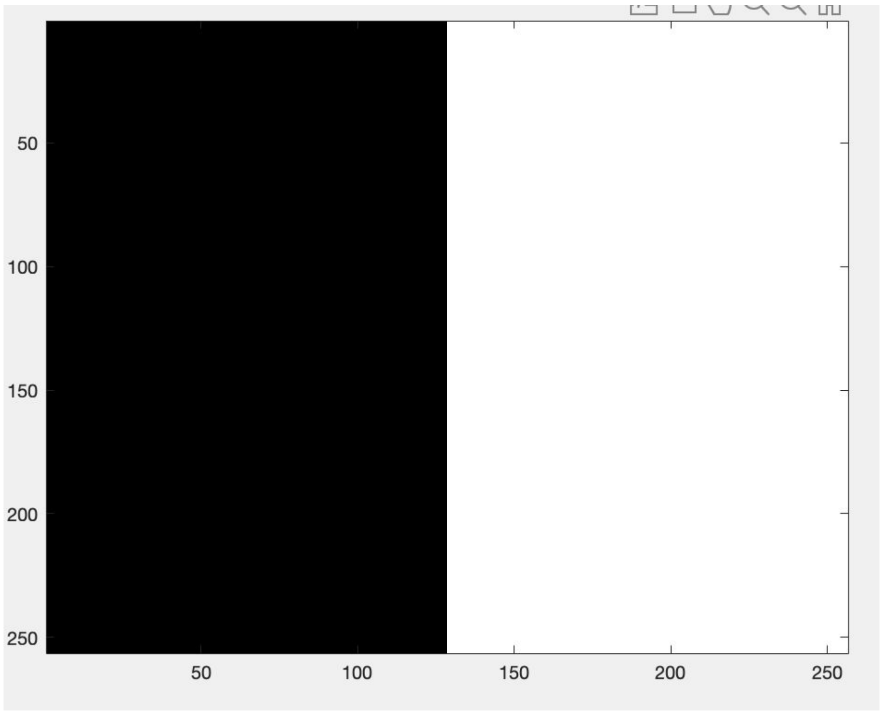


Image d2 - Binary mask `BW`

As we experiment through this project, we found it difficult to deal with source images of different sizes. So in the final design of our program, we assume both source images to be of the same size. We encourage our users to pre-process the source images and use our program solely as a blender. Image 3 below shows the final output, or the blended image, of our experiment.

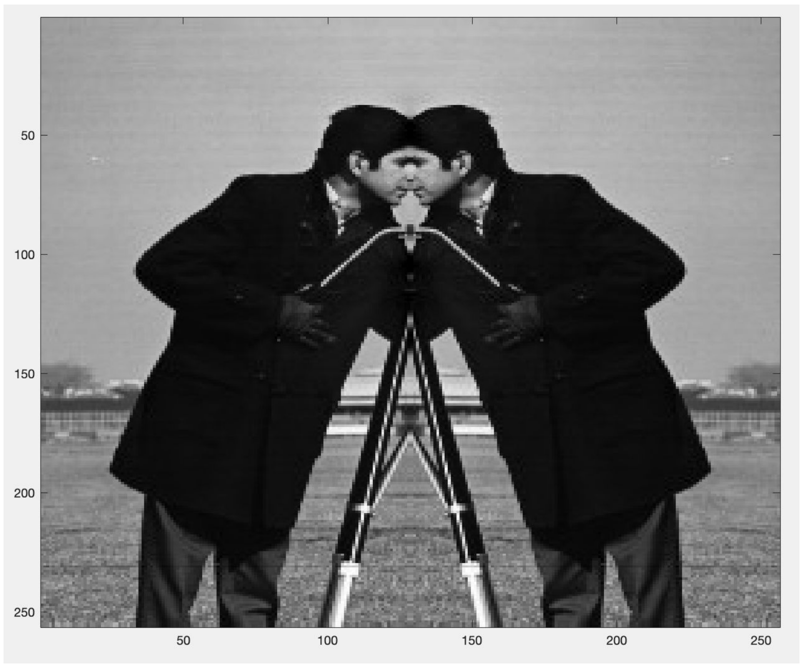


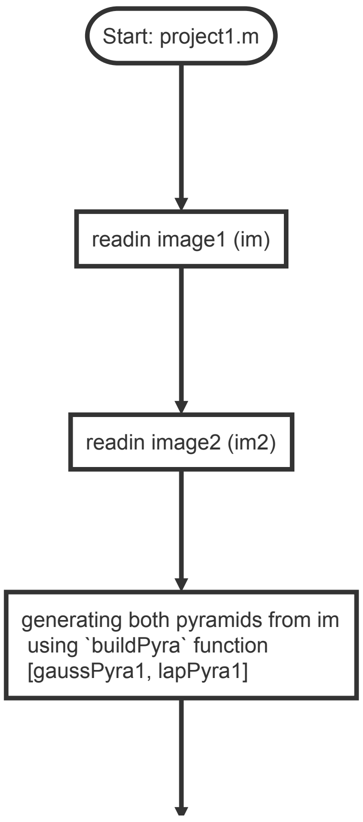
Image d3 - Preview of output result

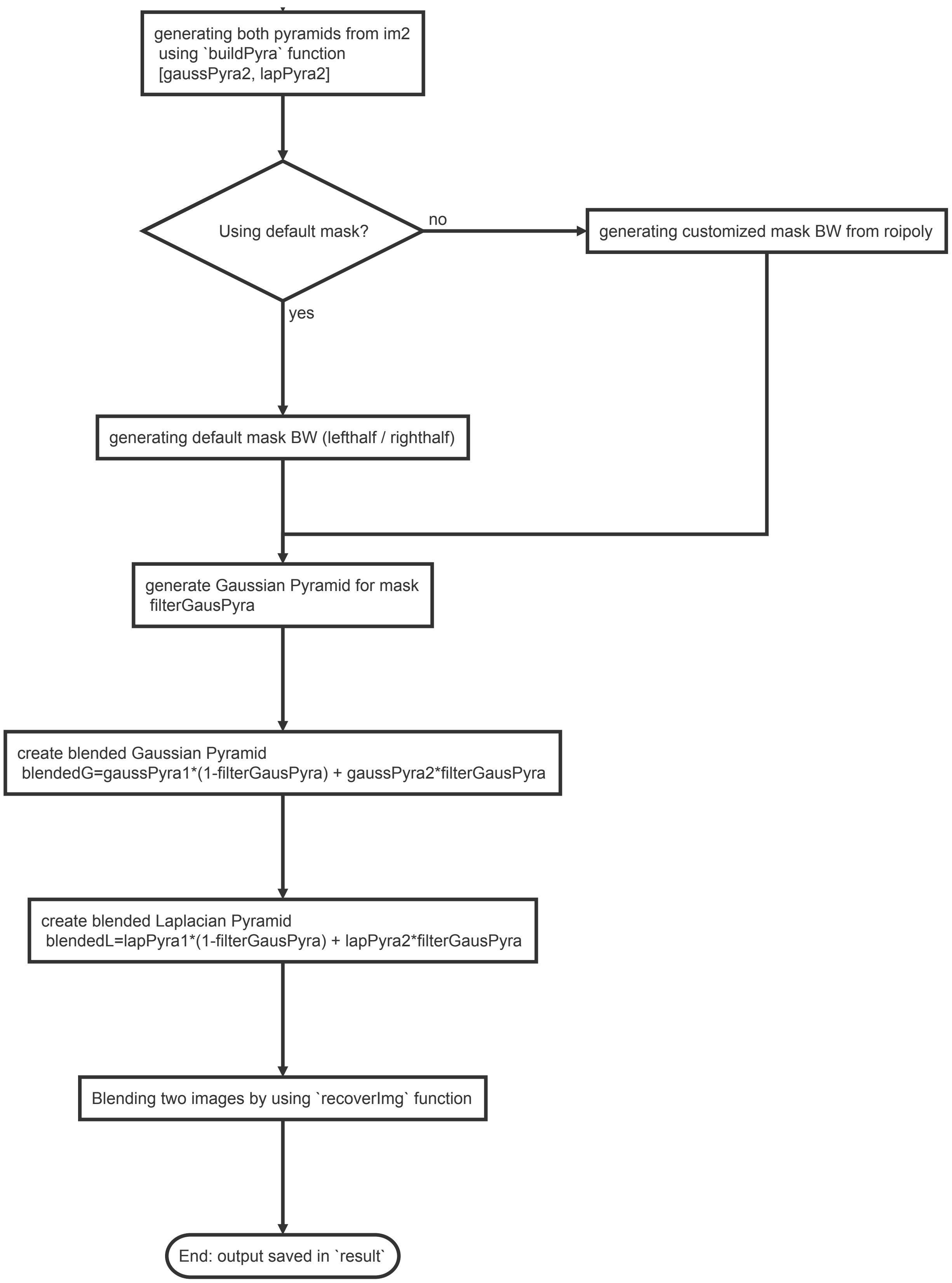
Appendix: Flow Chart and Function Specifications

main function - `project1.m`

Input: N/A. Specific path to two images in the program.

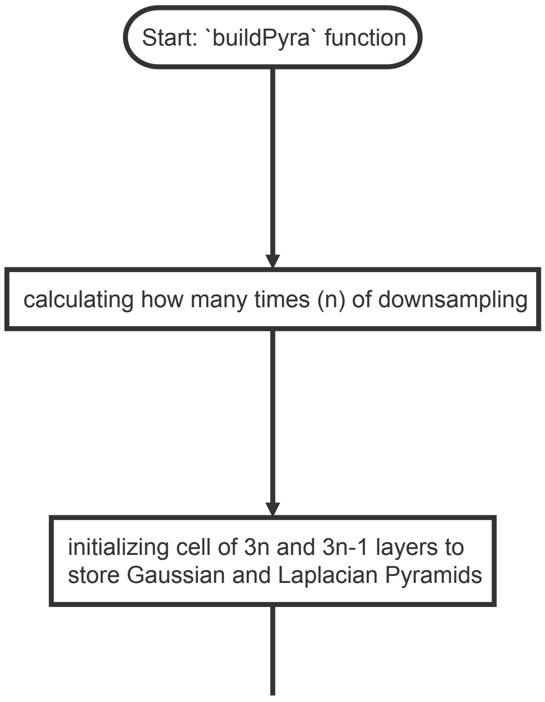
Output: Blended image matrix `result`

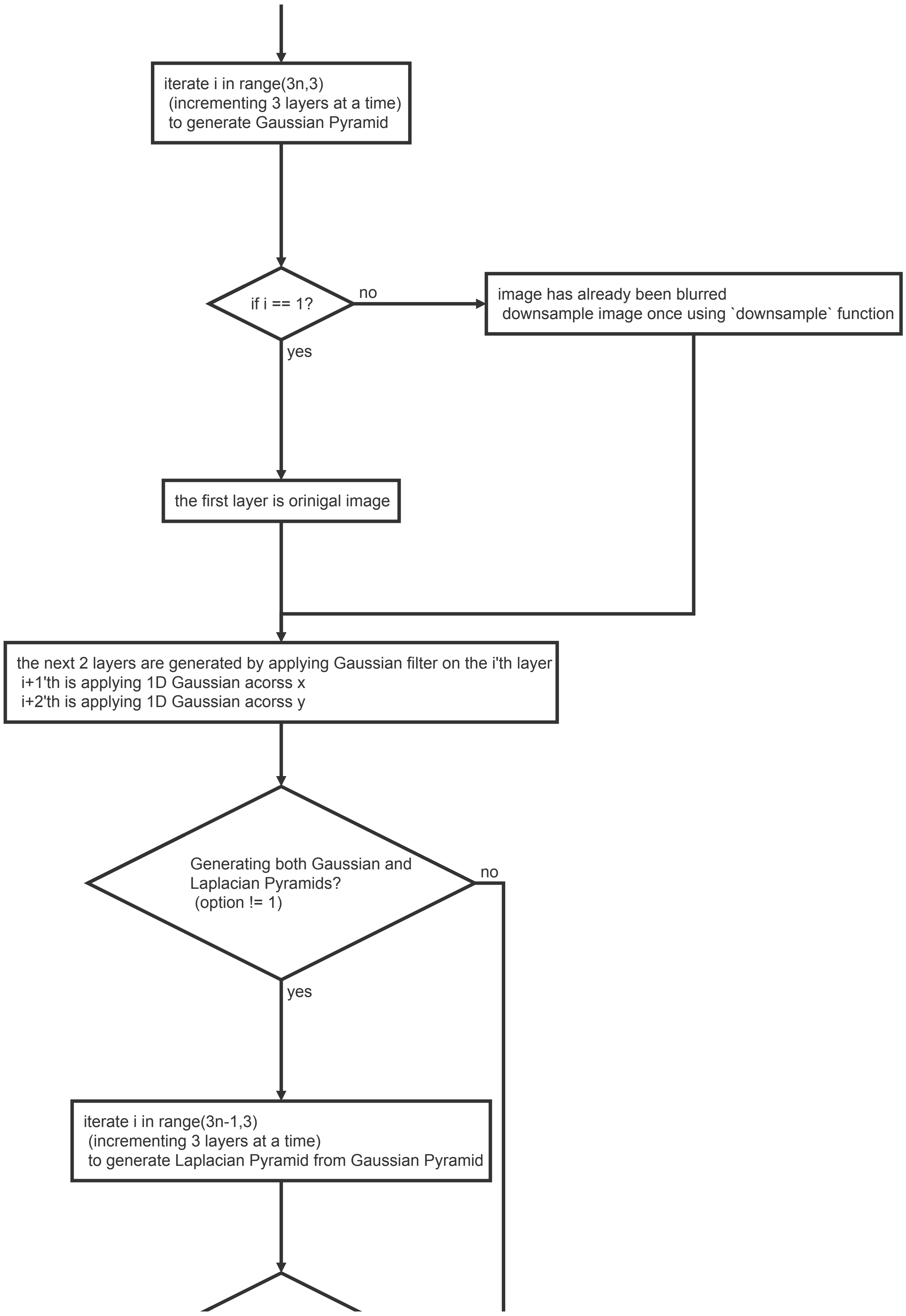

Start: project1.m
readin image1 (im)
readin image2 (im2)
generating both pyramids from im using `buildPyra` function [gaussPyra1, lapPyra1]

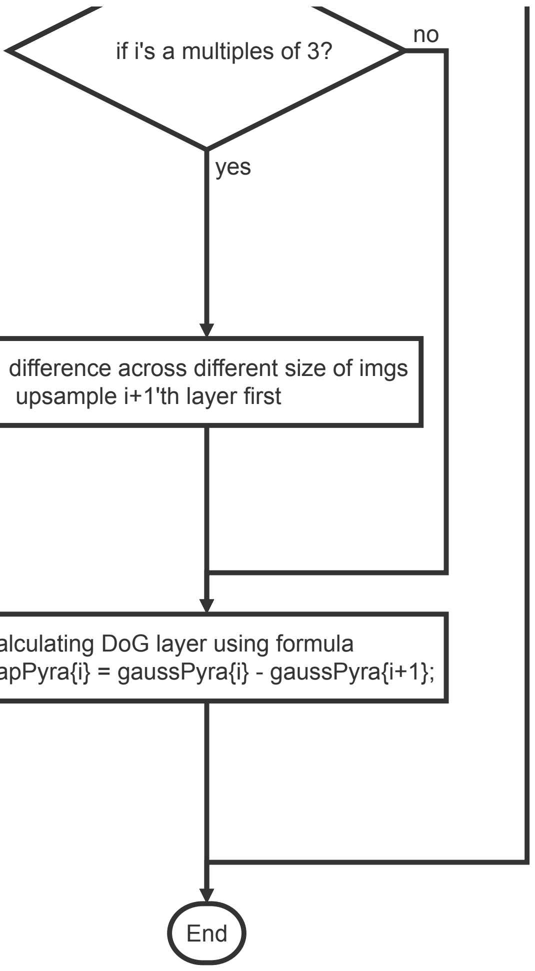

generating both pyramids from im2 using `buildPyra` function [gaussPyra2, lapPyra2]
Using default mask?
no
generating customized mask BW from roipoly
yes
generating default mask BW (lefthalf / righthalf)
generate Gaussian Pyramid for mask filterGausPyra
create blended Gaussian Pyramid blendedG=gaussPyra1*(1-filterGausPyra) + gaussPyra2*filterGausPyra
create blended Laplacian Pyramid blendedL=lapPyra1*(1-filterGausPyra) + lapPyra2*filterGausPyra
Blending two images by using `recoverImg` function
End: output saved in `result`

Building Pyramid Function - `buildPyra.m`

Input: `im` - image descriptor Input: `k` - threshold pixel to stop downsampling Input: `option = 1` for generating Gaussian Pyramid only Output: `gaussPyra` - cell array for Gaussian Pyramid Output: `lapPyra` - cell array for Laplacian Pyramid

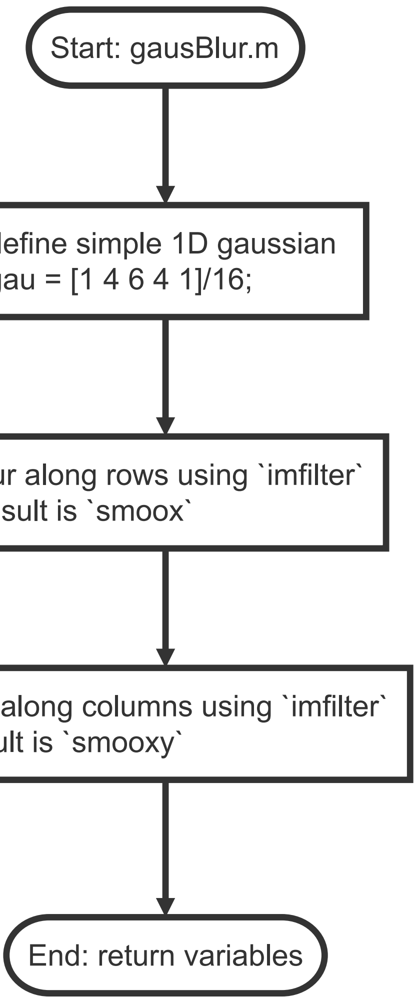

Start: `buildPyra` function
calculating how many times (n) of downsampling
initializing cell of 3n and 3n-1 layers to store Gaussian and Laplacian Pyramids


iterate i in range(3n,3)
(incrementing 3 layers at a time)
to generate Gaussian Pyramid
if i == 1?
no
image has already been blurred
downsample image once using `downsample` function
yes
the first layer is orinigal image
the next 2 layers are generated by applying Gaussian filter on the i'th layer
i+1'th is applying 1D Gaussian acorss x
i+2'th is applying 1D Gaussian acorss y
Generating both Gaussian and
Laplacian Pyramids?
(option != 1)
no
yes
iterate i in range(3n-1,3)
(incrementing 3 layers at a time)
to generate Laplacian Pyramid from Gaussian Pyramid


if i's a multiples of 3?
no
yes
difference across different size of imgs
upsample i+1'th layer first
calculating DoG layer using formula
lapPyra{i} = gaussPyra{i} - gaussPyra{i+1};
End

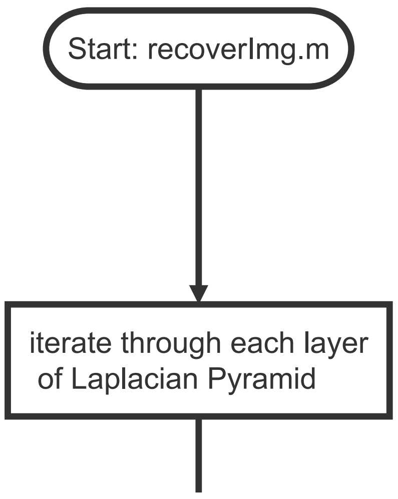
Gaussian Blur Filter Function - `gausBlur.m`

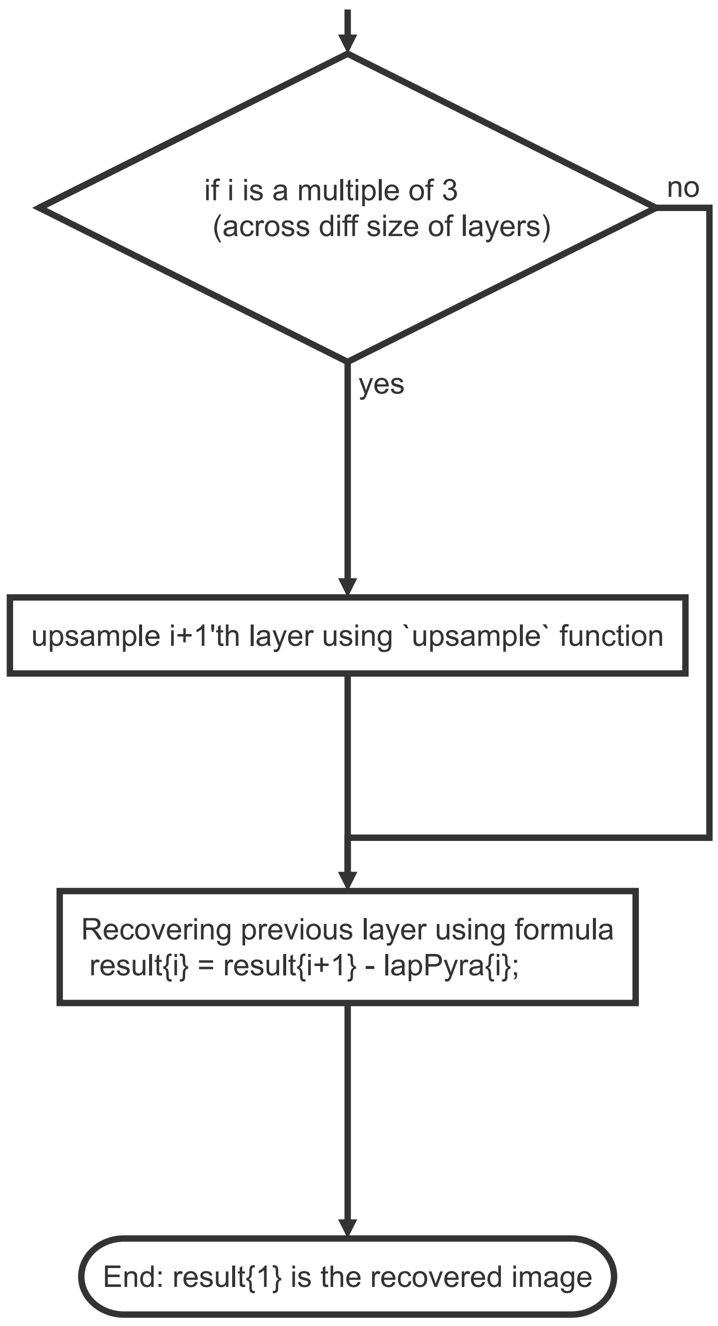
Input: `im` - image descriptor Output: `smoox` -1D blurred image (across x) descriptor Output: `smooxy` - 2D blurred image descriptor



Gaussian Blur Filter Function - `recoverImg.m`

Input: `gausLstLyr` - the smallest image / layer of the Gaussian Pyramid Input: `lapPyra` - generated Laplacian Pyramid Output: `img` - recovered original image descriptor matrix




if i is a multiple of 3
(across diff size of layers)
no
yes
upsample i+1'th layer using `upsample` function
Recovering previous layer using formula
result{i} = result{i+1} - lapPyra{i};
End: result{1} is the recovered image