

# ImagesToLARModel, a tool for creation of three-dimensional models from a stack of images

Danilo Salvati

October 19, 2015

## **Abstract**

This is the abstract (we will use LAR [\[CL13\]](#))

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Exporting the library</b>	<b>3</b>
2.1	Installing the library . . . . .	23
<b>3</b>	<b>Conclusions</b>	<b>23</b>
3.1	Results . . . . .	23
3.2	Further improvements . . . . .	23
<b>A</b>	<b>Utility functions</b>	<b>24</b>
<b>B</b>	<b>Tests</b>	<b>24</b>

# 1 Introduction

## 2 Exporting the library

```
"src/ImagesToLARModel.jl" 3≡
module ImagesToLARModel
"""
Main module for the library. It starts conversion
taking configuration parameters
"""
require(string(Pkg.dir("ImagesToLARModel/src"), "/imagesConversion.jl"))

import JSON
import ImagesConversion

using Logging

export convertImagesToLARModel

function loadConfiguration(configurationFile)
"""
load parameters from JSON file

configurationFile: Path of the configuration file
"""

configuration = JSON.parse(configurationFile)

DEBUG_LEVELS = [DEBUG, INFO, WARNING, ERROR, CRITICAL]

return configuration["inputDirectory"], configuration["outputDirectory"], configuration["bestImage"],
configuration["nx"], configuration["ny"], configuration["nz"],
DEBUG_LEVELS[configuration["DEBUG_LEVEL"]]
end

function convertImagesToLARModel(configurationFile)
"""
Start conversion of a stack of images into a 3D model
loading parameters from a JSON configuration file

configurationFile: Path of the configuration file
"""

inputDirectory, outputDirectory, bestImage, nx, ny, nz, DEBUG_LEVEL = loadConfiguration(open(configurationFile))
convertImagesToLARModel(inputDirectory, outputDirectory, bestImage, nx, ny, nz, DEBUG_LEVEL)
end
```

```

function convertImagesToLARModel(inputDirectory, outputDirectory, bestImage,
                                nx, ny, nz, DEBUG_LEVEL = INFO)
    """
    Start conversion of a stack of images into a 3D model

    inputDirectory: Directory containing the stack of images
    outputDirectory: Directory containing the output
    bestImage: Image chosen for centroids computation
    nx, ny, nz: Border dimensions (Possibly the biggest power of two of images dimensions)
    DEBUG_LEVEL: Debug level for Julia logger. It can be one of the following:
        - DEBUG
        - INFO
        - WARNING
        - ERROR
        - CRITICAL
    """
    # Create output directory
    try
        mkpath(outputDirectory)
    catch
    end

    Logging.configure(level=DEBUG_LEVEL)
    ImagesConversion.images2LARModel(nx, ny, nz, bestImage, inputDirectory, outputDirectory)
end
end
◇

```

```

"src/imagesConversion.jl" 4≡
module ImagesConversion

    require(string(Pkg.dir("ImagesToLARModel/src"), "/generateBorderMatrix.jl"))
    require(string(Pkg.dir("ImagesToLARModel/src"), "/pngStack2Array3dJulia.jl"))
    require(string(Pkg.dir("ImagesToLARModel/src"), "/lar2Julia.jl"))
    require(string(Pkg.dir("ImagesToLARModel/src"), "/model2Obj.jl"))

    import GenerateBorderMatrix
    import PngStack2Array3dJulia
    import Lar2Julia
    import Model2Obj

    import JSON

```

```

using PyCall
@pyimport scipy.sparse as Pysparse

using Logging

export images2LARModel

"""
This is main module for converting a stack
of images into a 3d model
"""

function images2LARModel(nx, ny, nz, bestImage, inputDirectory, outputDirectory)
    """
    Convert a stack of images into a 3d model
    """

    info("Starting model creation")

    numberOfClusters = 2 # Number of clusters for
                        # images segmentation

    imageWidth, imageHeight = PngStack2Array3dJulia.getImageData(string(inputDirectory,bestImage),
    imageDepth = length(readdir(inputDirectory))

    # Computing border matrix
    info("Computing border matrix")
    try
        mkdir(string(outputDirectory, "BORDERS"))
    catch
    end
    borderFilename = GenerateBorderMatrix.getOriented3BorderPath(string(outputDirectory, "BORDERS"),

    # Starting images conversion and border computation
    info("Starting images conversion")
    startImageConversion(inputDirectory, bestImage, outputDirectory, borderFilename,
                        imageHeight, imageWidth, imageDepth,
                        nx, ny, nz,
                        numberOfClusters)

end

function startImageConversion(sliceDirectory, bestImage, outputDirectory, borderFilename,
                        imageHeight, imageWidth, imageDepth,

```

```

        imageDx, imageDy, imageDz,
        numberOfClusters)
"""
Support function for converting a stack of images into a model

sliceDirectory: directory containing the image stack
imageForCentroids: image chosen for centroid computation
"""

info("Moving images into temp directory")
try
    mkdir(string(outputDirectory, "TEMP"))
catch
end

tempDirectory = string(outputDirectory, "TEMP/")

newBestImage = PngStack2Array3dJulia.convertImages(sliceDirectory, tempDirectory, bestImage)

# Create clusters for image segmentation
info("Computing image centroids")
debug("Best image = ", bestImage)
centroidsCalc = PngStack2Array3dJulia.calculateClusterCentroids(tempDirectory, newBestImage,
debug(string("centroids = ", centroidsCalc))

try
    mkdir(string(outputDirectory, "BORDERS"))
catch
end
debug(string("Opening border file: ", "border_", imageDx, "-", imageDy, "-", imageDz, ".json"))
boundaryMat = getBorderMatrix(string(outputDirectory, "BORDERS/", "border_", imageDx, "-",
                                imageDy, "-", imageDz, ".json"))

beginImageStack = 0
endImage = beginImageStack

info("Converting images into a 3d model")
for zBlock in 0:(imageDepth / imageDz - 1)
    startImage = endImage
    endImage = startImage + imageDz
    info("StartImage = ", startImage)
    info("endImage = ", endImage)
    info(string("Start process conversion process ", zBlock))
    imageConversionProcess(tempDirectory, outputDirectory,
                            beginImageStack, startImage, endImage,
                            imageDx, imageDy, imageDz,
                            imageHeight, imageWidth,

```

```

                                centroidsCalc, boundaryMat)
end

# TODO: add something for waiting all processes
info("Merging obj models")
Model2Obj.mergeObj(string(outputDirectory,"MODELS"))

end

function imageConversionProcess(sliceDirectory, outputDirectory,
                                beginImageStack, startImage, endImage,
                                imageDx, imageDy, imageDz,
                                imageHeight, imageWidth,
                                centroids, boundaryMat)
    """
    Support function for converting a stack of image on a single
    independent process
    """

    info("Transforming png data into 3d array")
    theImage = PngStack2Array3dJulia.pngstack2array3d(sliceDirectory, startImage, endImage, centroids)

    centroidsSorted = sort(vec(reshape(centroids, 1, 2)))
    foreground = centroidsSorted[2]
    background = centroidsSorted[1]
    debug(string("background = ", background, " foreground = ", foreground))
    for xBlock in 0:(imageHeight / imageDx - 1)
        for yBlock in 0:(imageWidth / imageDy - 1)
            yStart = xBlock * imageDx
            xStart = yBlock * imageDy
            #xEnd = xStart + imageDx
            #yEnd = yStart + imageDy
            xEnd = xStart + imageDy
            yEnd = yStart + imageDx
            debug("*****")
            debug(string("xStart = ", xStart, " xEnd = ", xEnd))
            debug(string("yStart = ", yStart, " yEnd = ", yEnd))
            debug("theImage dimensions: ", size(theImage)[1], " ", size(theImage[1])[1], " ", size(theImage[1][1]))

            # Getting a slice of theImage array
            image = Array{UInt8, 2}(convert{Int32, length(theImage)}, convert{Int32, xEnd - xStart}, convert{Int32, yEnd - yStart})
            debug("image size: ", size(image))
            for z in 1:length(theImage)
                for x in 1 : (xEnd - xStart)
                    for y in 1 : (yEnd - yStart)
                        image[z, x, y] = theImage[z][x + xStart, y + yStart]
                    end
                end
            end
        end
    end
end

```

```

        end
    end
end

nx, ny, nz = size(image)
chains3D = Array{UInt8, 0}
zStart = startImage - beginImageStack
for y in 0:(ny - 1)
    for x in 0:(nx - 1)
        for z in 0:(nz - 1)
            if(image[z + 1, x + 1, y + 1] == foreground)
                push!(chains3D, y + ny * (x + nx * z))
            end
        end
    end
end

if(length(chains3D) != 0)
    # Computing boundary chain
    debug("chains3d = ", chains3D)
    debug("Computing boundary chain")
    objectBoundaryChain = Lar2Julia.larBoundaryChain(boundaryMat, chains3D)
    debug("Converting models into obj")
    try
        mkdir(string(outputDirectory, "MODELS"))
    catch
    end
    # IMPORTANT: inverting xStart and yStart for obtaining correct rotation of the model
    outputFilename = string(outputDirectory, "MODELS/model-", xBlock, "-", yBlock, "_output.obj")
    Model2Obj.writeToObj(imageDx, imageDy, imageDz, yStart, xStart, zStart, objectBoundaryChain, outputFilename)
else
    debug("Model is empty")
end
end
end
end

function getBorderMatrix(borderFilename)
    """
    TO REMOVE WHEN PORTING OF LARCC IN JULIA IS COMPLETED

    Get the border matrix from json file and convert it in
    CSC format
    """
    # Loading borderMatrix from json file
    borderData = JSON.parsefile(borderFilename)
end

```



```

row = Array{Int64, length(borderData["ROW"])}
col = Array{Int64, length(borderData["COL"])}
data = Array{Int64, length(borderData["DATA"])}

for i in 1: length(borderData["ROW"])
    row[i] = borderData["ROW"][i]
end

for i in 1: length(borderData["COL"])
    col[i] = borderData["COL"][i]
end

for i in 1: length(borderData["DATA"])
    data[i] = borderData["DATA"][i]
end

# Converting csr matrix to csc
csrBorderMatrix = Pysparse.csr_matrix((data,col,row), shape=(borderData["ROWCOUNT"],borderData["COLCOUNT"]))
denseMatrix = pycall(csrBorderMatrix["toarray"],PyAny)

cscBoundaryMat = sparse(denseMatrix)

return cscBoundaryMat

end

end
◇

```

```

"src/generateBorderMatrix.jl" 9≡
module GenerateBorderMatrix
"""
Module for generation of the boundary matrix
"""

type MatrixObject
    ROWCOUNT
    COLCOUNT
    ROW
    COL
    DATA
end

```

```

export computeOriented3Border, writeBorder, getOriented3BorderPath

require(string(Pkg.dir("ImagesToLARModel/src"), "/larUtils.jl"))

import LARUtils
using PyCall

import JSON

@pyimport sys
unshift!(PyVector(pyimport("sys")["path"]), "") # Search for python modules in folder
# Search for python modules in package folder
unshift!(PyVector(pyimport("sys")["path"]), Pkg.dir("ImagesToLARModel/src"))
@pyimport larcc # Importing larcc from local folder

# Compute the 3-border operator
function computeOriented3Border(nx, ny, nz)
    """
    Compute the 3-border matrix using a modified
    version of larcc
    """
    V, bases = LARUtils.getBases(nx, ny, nz)
    boundaryMat = larcc.signedCellularBoundary(V, bases)
    return boundaryMat
end

function writeBorder(boundaryMatrix, outputFile)
    """
    Write 3-border matrix on json file

    boundaryMatrix: matrix to write on file
    outputFile: path of the outputFile
    """

    rowcount = boundaryMatrix[:shape][1]
    colcount = boundaryMatrix[:shape][2]

    row = boundaryMatrix[:indptr]
    col = boundaryMatrix[:indices]
    data = boundaryMatrix[:data]

    # Writing informations on file
    outfile = open(outputFile, "w")

```

```

    matrixObj = MatrixObject(rowcount, colcount, row, col, data)
    JSON.print(outfile, matrixObj)
    close(outfile)

end

function getOriented3BorderPath(borderPath, nx, ny, nz)
    """
    Try reading 3-border matrix from file. If it fails matrix
    is computed and saved on disk in JSON format

    borderPath: path of border directory
    nx, ny, nz: image dimensions
    """

    filename = string(borderPath, "/border_", nx, "-", ny, "-", nz, ".json")
    if !isfile(filename)
        border = computeOriented3Border(nx, ny, nz)
        writeBorder(border, filename)
    end
    return filename
end
end
◇

```

```

"src/lar2Julia.jl" 11≡
module Lar2Julia
    """
    larcc functions for Julia
    """
    export larBoundaryChain, cscChainToCellList

    import JSON

    using Logging

    function larBoundaryChain(cscBoundaryMat, brcCellList)
        """
        Compute boundary chains
        """

        # Computing boundary chains
    end
end

```

```

n = size(cscBoundaryMat)[1]
m = size(cscBoundaryMat)[2]

debug("Boundary matrix size: ", n, "\t", m)

data = ones(Int64, length(brcCellList))

i = Array{Int64, length(brcCellList)}
for k in 1:length(brcCellList)
    i[k] = brcCellList[k] + 1
end

j = ones{Int64, length(brcCellList)}

debug("cscChain rows length: ", length(i))
debug("cscChain columns length: ", length(j))
debug("cscChain data length: ", length(brcCellList))

debug("rows ", i)
debug("columns ", j)
debug("data ", data)

cscChain = sparse(i, j, data, m, 1)
cscmat = cscBoundaryMat * cscChain
out = cscBinFilter(cscmat)
return out
end

function cscBinFilter(CSCm)
    k = 1
    data = nonzeros(CSCm)
    sgArray = copysign(1, data)

    while k <= nnz(CSCm)
        if data[k] % 2 == 1 || data[k] % 2 == -1
            data[k] = 1 * sgArray[k]
        else
            data[k] = 0
        end
        k += 1
    end

    return CSCm
end

function cscChainToCellList(CSCm)

```

```

"""
Get a csc containing a chain and returns
the cell list of the "+1" oriented faces
"""

data = nonzeros(CSCm)
# Now I need to remove zero element (problem with Julia nonzeros)
nonzeroData = Array{Int64, 0}
for n in data
    if n != 0
        push!(nonzeroData, n)
    end
end

cellList = Array{Int64,0}
for (k, theRow) in enumerate(findn(CSCm)[1])
    if nonzeroData[k] == 1
        push!(cellList, theRow)
    end
end
return cellList
end
end
◇

```

```

"src/larUtils.jl" 13≡
module LARUtils
"""
Utility functions for extracting 3d models from images
"""
export ind, invertIndex, getBases

function ind(x, y, z, nx, ny)
    """
    Transform coordinates into linearized matrix indexes
    """
    return x + (nx+1) * (y + (ny+1) * (z))
end

function invertIndex(nx,ny,nz)
    """
    Invert indexes
    """

```

```

nx, ny, nz = nx + 1, ny + 1, nz + 1
function invertIndex0(offset)
    a0, b0 = trunc(offset / nx), offset % nx
    a1, b1 = trunc(a0 / ny), a0 % ny
    a2, b2 = trunc(a1 / nz), a1 % nz
    return b0, b1, b2
end
return invertIndex0
end

function getBases(nx, ny, nz)
    """
    Compute all LAR relations
    """

    function the3Dcell(coords)
        x,y,z = coords
        return [ind(x,y,z,nx,ny),ind(x+1,y,z,nx,ny),ind(x,y+1,z,nx,ny),ind(x,y,z+1,nx,ny),ind(x+1,y,z+1,nx,ny),ind(x,y+1,z+1,nx,ny),ind(x+1,y+1,z+1,nx,ny)]
    end

    # Calculating vertex coordinates (nx * ny * nz)
    V = Array{Int64}[]
    for z in 0:nz
        for y in 0:ny
            for x in 0:nx
                push!(V,[x,y,z])
            end
        end
    end

    # Building CV relationship
    CV = Array{Int64}[]
    for z in 0:nz-1
        for y in 0:ny-1
            for x in 0:nx-1
                push!(CV,the3Dcell([x,y,z]))
            end
        end
    end

    # Building FV relationship
    FV = Array{Int64}[]
    v2coords = invertIndex(nx,ny,nz)

```

```

for h in 0:(length(V)-1)
  x,y,z = v2coords(h)

  if (x < nx) && (y < ny)
    push!(FV, [h,ind(x+1,y,z,nx,ny),ind(x,y+1,z,nx,ny),ind(x+1,y+1,z,nx,ny)])
  end

  if (x < nx) && (z < nz)
    push!(FV, [h,ind(x+1,y,z,nx,ny),ind(x,y,z+1,nx,ny),ind(x+1,y,z+1,nx,ny)])
  end

  if (y < ny) && (z < nz)
    push!(FV, [h,ind(x,y+1,z,nx,ny),ind(x,y,z+1,nx,ny),ind(x,y+1,z+1,nx,ny)])
  end

end

# Building VV relationship
VV = map((x)->[x], 0:length(V)-1)

# Building EV relationship
EV = Array{Int64}[]
for h in 0:length(V)-1
  x,y,z = v2coords(h)
  if (x < nx)
    push!(EV, [h,ind(x+1,y,z,nx,ny)])
  end
  if (y < ny)
    push!(EV, [h,ind(x,y+1,z,nx,ny)])
  end
  if (z < nz)
    push!(EV, [h,ind(x,y,z+1,nx,ny)])
  end
end

# return all basis
return V, (VV, EV, FV, CV)
end
end
◇

```

"src/model20bj.jl" 15≡

```

module Model2Obj
"""
Module that takes a 3d model and write it on
obj files
"""

require(string(Pkg.dir("ImagesToLARModel/src"), "/larUtils.jl"))

import LARUtils

using Logging

export writeToObj, mergeObj

function writeToObj(imageDx, imageDy, imageDz,
                    xStart, yStart, zStart,
                    objectBoundaryChain, outputFilename)
    """
    Takes the boundary chain of a part of the model
    and writes it on stl files
    """
    V, bases = LARUtils.getBases(imageDx, imageDy, imageDz)
    FV = bases[3]

    outputVtx = string(outputFilename, "_vtx.stl")
    outputFaces = string(outputFilename, "_faces.stl")

    fileVertex = open(outputVtx, "w")
    fileFaces = open(outputFaces, "w")

    vertex_count = 1
    count = 0

    #b2cells = Lar2Julia.cscChainToCellList(objectBoundaryChain)
    # Get all cells (independently from orientation)
    b2cells = findn(objectBoundaryChain)[1]

    debug("b2cells = ", b2cells)

    for f in b2cells
        old_vertex_count = vertex_count
        for vtx in FV[f]
            write(fileVertex, "v ")
            write(fileVertex, string(convert{Int64, V[vtx + 1][1] + xStart}))
            write(fileVertex, " ")
            write(fileVertex, string(convert{Int64, V[vtx + 1][2] + yStart}))

```



```

        write(fileVertex, " ")
        write(fileVertex, string(convert(Int64, V[vtx + 1][3] + zStart)))
        write(fileVertex, "\n")
        vertex_count += 1
    end

    write(fileFaces, "f ")
    write(fileFaces, string(old_vertex_count))
    write(fileFaces, " ")
    write(fileFaces, string(old_vertex_count + 1))
    write(fileFaces, " ")
    write(fileFaces, string(old_vertex_count + 3))
    write(fileFaces, "\n")

    write(fileFaces, "f ")
    write(fileFaces, string(old_vertex_count))
    write(fileFaces, " ")
    write(fileFaces, string(old_vertex_count + 3))
    write(fileFaces, " ")
    write(fileFaces, string(old_vertex_count + 2))
    write(fileFaces, "\n")

end

close(fileVertex)
close(fileFaces)

end

function mergeObj(modelDirectory)
    """
    Merge stl files in a single obj file

    modelDirectory: directory containing models
    """

    files = readdir(modelDirectory)
    vertices_files = files[find(s -> contains(s,string("_vtx.stl")), files)]
    faces_files = files[find(s -> contains(s,string("_faces.stl")), files)]
    obj_file = open(string(modelDirectory, "/", "model.obj"), "w") # Output file

    vertices_counts = Array{Int64, length(vertices_files)}
    number_of_vertices = 0
    for i in 1:length(vertices_files)
        vtx_file = vertices_files[i]
        f = open(string(modelDirectory, "/", vtx_file))
    end
end

```

```

    # Writing vertices on the obj file
    for ln in eachline(f)
        write(obj_file, ln)
        number_of_vertices += 1
    end
    # Saving number of vertices
    vertices_counts[i] = number_of_vertices
    close(f)
end

for i in 1 : length(faces_files)
    faces_file = faces_files[i]
    f = open(string(modelDirectory, "/", faces_file))

    for ln in eachline(f)
        splitted = split(ln)
        write(obj_file, "f ")
        if i > 1
            write(obj_file, string(parse(splitted[2]) + vertices_counts[i - 1], " "))
            write(obj_file, string(parse(splitted[3]) + vertices_counts[i - 1], " "))
            write(obj_file, string(parse(splitted[4]) + vertices_counts[i - 1]))
        else
            write(obj_file, string(splitted[2], " "))
            write(obj_file, string(splitted[3], " "))
            write(obj_file, splitted[4])
        end
        write(obj_file, "\n")
    end
    close(f)
end
close(obj_file)

# Removing all tmp files
for vtx_file in vertices_files
    rm(string(modelDirectory, "/", vtx_file))
end

for fcs_file in faces_files
    rm(string(modelDirectory, "/", fcs_file))
end

end
end
◇

```

```

"src/pngStack2Array3dJulia.jl" 19≡
module PngStack2Array3dJulia

    """
    This module loads a stack of png files returning
    an array of pixel values divided into segments
    """

    export calculateClusterCentroids, pngstack2array3d, getImageData, convertImages

    using Images # For loading png images
    using Colors # For grayscale images
    using PyCall # For including python clustering
    using Logging
    @pyimport scipy.ndimage as ndimage
    @pyimport scipy.cluster.vq as cluster

    NOISE_SHAPE_DETECT=10

    function getImageData(imageFile)
        """
        Get width and heigth from a png image
        """

        input = open(imageFile, "r")
        data = readbytes(input, 24)

        if (data[2:4] != [80, 78, 71] && data[13:16] != [73, 72, 68, 82])
            error("This is not a png image")
        end

        w = data[17:20]
        h = data[21:24]

        width = reinterpret{Int32, reverse(w)}[1]
        height = reinterpret{Int32, reverse(h)}[1]

        close(input)

        return width, height
    end

    function calculateClusterCentroids(path, image, numberOfClusters = 2)
        """
        Loads an image and calculate cluster centroids for segmentation
        """
    end
end

```

```

path: Path of the image folder
image: name of the image
numberOfClusters: number of desired clusters
"""
imageFilename = string(path, image)

img = imread(imageFilename) # Open png image with Julia Package

rgb_img = convert(Image{ColorTypes.RGB}, img)
gray_img = convert(Image{ColorTypes.Gray}, rgb_img)
imArray = raw(gray_img)

imageWidth = size(imArray)[1]
imageHeight = size(imArray)[2]

# Getting pixel values and saving them with another shape
image3d = Array{Array{UInt8,2}, 0}

# Inserting page on another list and reshaping
push!(image3d, imArray)
pixel = reshape(image3d[1], (imageWidth * imageHeight), 1)

# Segmenting image using kmeans
# https://en.wikipedia.org/wiki/Image_segmentation#Clustering_methods

centroids,_ = cluster.kmeans(pixel, numberOfClusters)

return centroids
end

function pngstack2array3d(path, minSlice, maxSlice, centroids)
    """
    Import a stack of PNG images into a 3d array

    path: path of images directory
    minSlice and maxSlice: number of first and last slice
    centroids: centroids for image segmentation
    """

    # image3d contains all images values
    image3d = Array{Array{UInt8,2}, 0}

    debug("maxSlice = ", maxSlice, " minSlice = ", minSlice)
    files = readdir(path)

```

```

for slice in minSlice : (maxSlice - 1)
    debug("slice = ", slice)
    imageFilename = string(path, files[slice + 1])
    debug("image name: ", imageFilename)
    img = imread(imageFilename) # Open png image with Julia Package

    # Converting image in grayscale
    rgb_img = convert(Image{ColorTypes.RGB}, img)
    gray_img = convert(Image{ColorTypes.Gray}, rgb_img)
    imArray = raw(gray_img) # Putting pixel values into RAW 3d array
    debug("imArray size: ", size(imArray))

    # Inserting page on another list and reshaping
    push!(image3d, imArray)

end

# Removing noise using a median filter and quantization
for page in 1:length(image3d)

    # Denoising
    image3d[page] = ndimage.median_filter(image3d[page], NOISE_SHAPE_DETECT)

    # Image Quantization
    debug("page = ", page)
    debug("image3d[page] dimensions: ", size(image3d[page])[1], "\t", size(image3d[page])[2])
    pixel = reshape(image3d[page], size(image3d[page])[1] * size(image3d[page])[2] , 1)
    qnt,_ = cluster.vq(pixel,centroids)

    # Reshaping quantization result
    centers_idx = reshape(qnt, size(image3d[page],1), size(image3d[page],2))
    #centers_idx = reshape(qnt, size(image3d[page]))

    # Inserting quantized values into 3d image array
    tmp = Array{UInt8, size(image3d[page],1), size(image3d[page],2)}

    for j in 1:size(image3d[1],2)
        for i in 1:size(image3d[1],1)
            tmp[i,j] = centroids[centers_idx[i,j] + 1]
        end
    end

    image3d[page] = tmp

end

```

```

    return image3d
end

function convertImages(inputPath, outputPath, bestImage)
    """
    Get all images contained in inputPath directory
    saving them in outputPath directory in png format.
    If images have one of two odd dimensions, they will be resized
    and if folder contains an odd number of images another one will be
    added

    inputPath: Directory containing input images
    outputPath: Temporary directory containing png images
    bestImage: Image chosen for centroids computation

    Returns the new name for the best image
    """

    imageFiles = readdir(inputPath)
    numberOfImages = length(imageFiles)
    outputPrefix = ""
    for i in 1: length(string(numberOfImages)) - 1
        outputPrefix = string(outputPrefix,"0")
    end

    newBestImage = ""
    imageNumber = 0
    for imageFile in imageFiles
        img = imread(string(inputPath, imageFile))

        # resizing images if they do not have even dimensions
        dim = size(img)
        if(dim[1] % 2 != 0)
            debug("Image has odd x; resizing")
            xrange = 1: dim[1] - 1
        else
            xrange = 1: dim[1]
        end

        if(dim[2] % 2 != 0)
            debug("Image has odd y; resizing")
            yrange = 1: dim[2] - 1
        else
            yrange = 1: dim[2]
        end
    end
end

```

```

img = subim(img, xrange, yrange)

outputFilename = string(outputPath, outputPrefix[length(string(imageNumber)):end], imageNum
imwrite(img, outputFilename)

# Searching the best image
if(imageFile == bestImage)
    newBestImage = string(outputPrefix[length(string(imageNumber)):end], imageNumber, ".png")
end

imageNumber += 1
end

# Adding another image if they are odd
if(numberOfImages % 2 != 0)
    debug("Odd images, adding one")
    bestImage = imread(string(outputPath, "/", newBestImage))
    imArray = zeros(UInt8, size(bestImage))
    img = grayim(imArray)
    outputFilename = string(outputPath, "/", outputPrefix[length(string(imageNumber)):end], im
    imwrite(img, outputFilename)
end

return newBestImage
end

end
◇

```

## 2.1 Installing the library

## 3 Conclusions

### 3.1 Results

### 3.2 Further improvements

## References

- [CL13] CVD-Lab, *Linear algebraic representation*, Tech. Report 13-00, Roma Tre University, October 2013.

## B Tests

```

"test/generateBorderMatrix.jl" 24==
push!(LOAD_PATH, "../..")
import GenerateBorderMatrix
import JSON
using Base.Test

function testComputeOriented3Border()
    """
    Test function for computeOriented3Border
    """

    boundaryMatrix = GenerateBorderMatrix.computeOriented3Border(2,2,2)

    rowcount = boundaryMatrix[:shape][1]
    @test rowcount == 36
    colcount = boundaryMatrix[:shape][2]
    @test colcount == 8
    row = boundaryMatrix[:indptr]
    @test row == [0,1,2,3,4,5,7,8,9,11,12,13,15,17,18,19,20,22,23,24,26,27,29,30,32,34,35,37,39,
    col = boundaryMatrix[:indices]
    @test col == [0,0,0,1,1,0,1,1,2,0,2,2,3,1,3,2,3,3,2,3,0,4,4,4,1,5,5,4,5,5,2,6,4,6,6,3,7,5,7,
    data = boundaryMatrix[:data]
    @test data == [-1,1,-1,-1,1,1,-1,1,-1,-1,1,-1,-1,-1,1,-1,-1,-1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,1,-1,1,1,
end

function testWriteBorder()
    """
    Test for writeBorder
    """

    boundaryMatrix = GenerateBorderMatrix.computeOriented3Border(2,2,2)
    filename = "borderFile"

    GenerateBorderMatrix.writeBorder(boundaryMatrix, filename)
    @test isfile(filename)

    # Loading borderMatrix from json file
    borderData = JSON.parsefile(filename)
    row = Array{Int64, length(borderData["ROW"])}
    col = Array{Int64, length(borderData["COL"])}
    data = Array{Int64, length(borderData["DATA"])}

```





```

    image = 0
    centroids = PngStack2Array3dJulia.calculateClusterCentroids(path, image, 2)

    expected = [0, 253]
    centroids = vec(reshape(centroids, 1, 2))

    @test sort(centroids) == expected
end

function testPngstack2array3d()
    """
    Test function for pngstack2array3d
    """
    path = "images/"
    minSlice = 0
    maxSlice = 4
    centroids = PngStack2Array3dJulia.calculateClusterCentroids(path, 0, 2)
    image3d = PngStack2Array3dJulia.pngstack2array3d(path, minSlice, maxSlice, centroids)

    @test size(image3d)[1] == 5
    @test size(image3d[1])[1] == 50
    @test size(image3d[1])[2] == 200

end

function executeAllTests()
    @time testCalculateClusterCentroids()
    @time testPngstack2array3d()
    @time testGetImageData()
    println("Tests completed.")
end

executeAllTests()

◇

```

## Test for LAR utilities

```

"test/LARUtils.jl" 27≡
    push!(LOAD_PATH, "../..")
    import LARUtils
    using Base.Test

```

```

function testInd()
    """
    Test function for ind
    """

    nx = 2
    ny = 2

    @test LARUtils.ind(0, 0, 0, nx, ny) == 0
    @test LARUtils.ind(1, 1, 1, nx, ny) == 13
    @test LARUtils.ind(2, 5, 4, nx, ny) == 53
    @test LARUtils.ind(1, 1, 1, nx, ny) == 13
    @test LARUtils.ind(2, 7, 1, nx, ny) == 32
    @test LARUtils.ind(1, 0, 3, nx, ny) == 28
end

function executeAllTests()
    @time testInd()
    println("Tests completed.")
end

executeAllTests()

◇

```