

ParallelKmenasIgnageCompressor

Generated on Sun Sep 1 2024 12:45:01 for ParallelKmenasIgnageCompressor by Doxygen
1.9.8

Sun Sep 1 2024 12:45:01

1 Parallel Kmeans Images Compressor	1
1.1 Doxygen Documentation	1
1.2 Prerequisites	1
1.2.1 OpenCV C++ Library	1
1.2.2 Mpicc	1
1.2.3 OpenMP	1
1.3 Getting Started	2
1.4 What to expect	2
1.5 Project Structure	2
1.5.0.1 Folders	2
1.5.0.2 Files and Executables	3
1.6 How does it work?	3
1.7 Parallelization Techniques	4
1.8 Benchmarking	4
1.9 Authors	4
2 Namespace Index	5
2.1 Namespace List	5
3 Hierarchical Index	7
3.1 Class Hierarchy	7
4 Class Index	9
4.1 Class List	9
5 File Index	11
5.1 File List	11
6 Namespace Documentation	13
6.1 km Namespace Reference	13
6.1.1 Detailed Description	14
6.2 km::filesUtils Namespace Reference	14
6.2.1 Detailed Description	14
6.2.2 Function Documentation	14
6.2.2.1 createDecodingMenu()	14
6.2.2.2 createOutputDirectories()	15
6.2.2.3 isCorrectExtension()	15
6.2.2.4 readBinaryFile()	15
6.2.2.5 writeBinaryFile()	15
6.3 km::imageUtils Namespace Reference	17
6.3.1 Detailed Description	17
6.3.2 Function Documentation	17
6.3.2.1 defineKValue()	17
6.3.2.2 pointsFromImage()	18

6.3.2.3 preprocessing()	18
6.4 km::utilsCLI Namespace Reference	18
6.4.1 Detailed Description	19
6.4.2 Function Documentation	19
6.4.2.1 compressionChoices()	19
6.4.2.2 decoderHeader()	19
6.4.2.3 displayDecodingMenu()	19
6.4.2.4 mainMenuHeader()	20
6.4.2.5 mpiEncoderHeader()	20
6.4.2.6 ompEncoderHeader()	20
6.4.2.7 printCompressionInformations()	20
6.4.2.8 sequentialEncoderHeader()	21
6.4.2.9 workDone()	21
7 Class Documentation	23
7.1 km::ConfigReader Class Reference	23
7.1.1 Detailed Description	25
7.1.2 Constructor & Destructor Documentation	26
7.1.2.1 ConfigReader()	26
7.1.3 Member Function Documentation	26
7.1.3.1 checkVariableExists()	26
7.1.3.2 getColorChoice()	26
7.1.3.3 getCompressionChoice()	26
7.1.3.4 getFifthLevelCompressionColor()	26
7.1.3.5 getFirstLevelCompressionColor()	27
7.1.3.6 getFourthLevelCompressionColor()	27
7.1.3.7 getInputImageFilePath()	27
7.1.3.8 getResizingFactor()	27
7.1.3.9 getSecondLevelCompressionColor()	27
7.1.3.10 getThirdLevelCompressionColor()	28
7.1.3.11 readConfigFile()	28
7.1.4 Member Data Documentation	28
7.1.4.1 color_choice	28
7.1.4.2 compression_choice	28
7.1.4.3 fifth_level_compression_color	28
7.1.4.4 first_level_compression_color	28
7.1.4.5 fourth_level_compression_color	29
7.1.4.6 inputImageFilePath	29
7.1.4.7 pattern	29
7.1.4.8 requiredVariables	29
7.1.4.9 resizing_factor	29
7.1.4.10 second_level_compression_color	29

7.1.4.11 third_level_compression_color	29
7.2 km::KMeansBase Class Reference	30
7.2.1 Detailed Description	32
7.2.2 Constructor & Destructor Documentation	32
7.2.2.1 KMeansBase() [1/2]	32
7.2.2.2 KMeansBase() [2/2]	32
7.2.2.3 ~KMeansBase()	32
7.2.3 Member Function Documentation	33
7.2.3.1 getCentroids()	33
7.2.3.2 getIterations()	33
7.2.3.3 getPoints()	33
7.2.3.4 run()	33
7.2.4 Member Data Documentation	33
7.2.4.1 centroids	33
7.2.4.2 k	34
7.2.4.3 number_of_iterations	34
7.2.4.4 points	34
7.3 km::KMeansCUDA Class Reference	34
7.3.1 Detailed Description	35
7.3.2 Constructor & Destructor Documentation	35
7.3.2.1 KMeansCUDA()	35
7.3.3 Member Function Documentation	36
7.3.3.1 getCentroids()	36
7.3.3.2 getIterations()	36
7.3.3.3 getPoints()	36
7.3.3.4 plotClusters()	36
7.3.3.5 printClusters()	36
7.3.3.6 run()	37
7.3.4 Member Data Documentation	37
7.3.4.1 centroids	37
7.3.4.2 k	37
7.3.4.3 number_of_iterations	37
7.3.4.4 points	37
7.4 km::KMeansMPI Class Reference	38
7.4.1 Detailed Description	40
7.4.2 Constructor & Destructor Documentation	40
7.4.2.1 KMeansMPI() [1/2]	40
7.4.2.2 KMeansMPI() [2/2]	41
7.4.3 Member Function Documentation	41
7.4.3.1 run()	41
7.4.4 Member Data Documentation	41
7.4.4.1 local_points	41

7.5 km::KMeansOMP Class Reference	42
7.5.1 Detailed Description	44
7.5.2 Constructor & Destructor Documentation	44
7.5.2.1 KMeansOMP()	44
7.5.3 Member Function Documentation	45
7.5.3.1 run()	45
7.6 km::KMeansSequential Class Reference	45
7.6.1 Detailed Description	48
7.6.2 Constructor & Destructor Documentation	48
7.6.2.1 KMeansSequential()	48
7.6.3 Member Function Documentation	49
7.6.3.1 run()	49
7.7 km::Performance Class Reference	49
7.7.1 Detailed Description	50
7.7.2 Constructor & Destructor Documentation	50
7.7.2.1 Performance()	50
7.7.3 Member Function Documentation	51
7.7.3.1 appendToCSV()	51
7.7.3.2 createOrOpenCSV()	51
7.7.3.3 extractFileName()	51
7.7.3.4 fillPerformance()	52
7.7.3.5 writeCSV()	52
7.7.4 Member Data Documentation	52
7.7.4.1 choice	52
7.7.4.2 img	53
7.7.4.3 method	53
7.8 km::Point Class Reference	53
7.8.1 Detailed Description	54
7.8.2 Constructor & Destructor Documentation	54
7.8.2.1 Point() [1/2]	54
7.8.2.2 Point() [2/2]	54
7.8.3 Member Function Documentation	55
7.8.3.1 distance()	55
7.8.3.2 getFeature()	55
7.8.3.3 getFeature_int()	55
7.8.3.4 setFeature()	56
7.8.4 Member Data Documentation	56
7.8.4.1 b	56
7.8.4.2 clusterId	56
7.8.4.3 g	56
7.8.4.4 id	56
7.8.4.5 r	57

8 File Documentation	59
8.1 include/configReader.hpp File Reference	59
8.1.1 Detailed Description	60
8.2 configReader.hpp	60
8.3 include/filesUtils.hpp File Reference	61
8.3.1 Detailed Description	62
8.4 filesUtils.hpp	62
8.5 include/imagesUtils.hpp File Reference	63
8.5.1 Detailed Description	63
8.6 imagesUtils.hpp	64
8.7 include/kmDocs.hpp File Reference	64
8.7.1 Detailed Description	64
8.8 kmDocs.hpp	64
8.9 include/kMeansBase.hpp File Reference	65
8.9.1 Detailed Description	65
8.10 kMeansBase.hpp	66
8.11 include/kMeansCUDA.cuh File Reference	66
8.11.1 Detailed Description	67
8.11.2 Macro Definition Documentation	67
8.11.2.1 KMEANS_CUDA_HPP	67
8.12 include/kMeansMPI.hpp File Reference	67
8.12.1 Detailed Description	68
8.13 kMeansMPI.hpp	68
8.14 include/kMeansOMP.hpp File Reference	68
8.14.1 Detailed Description	69
8.15 kMeansOMP.hpp	69
8.16 include/kMeansSequential.hpp File Reference	70
8.16.1 Detailed Description	70
8.17 kMeansSequential.hpp	71
8.18 include/performanceEvaluation.hpp File Reference	71
8.18.1 Detailed Description	71
8.19 performanceEvaluation.hpp	72
8.20 include/point.hpp File Reference	72
8.20.1 Detailed Description	73
8.21 point.hpp	73
8.22 include/utlisCLI.hpp File Reference	74
8.22.1 Detailed Description	75
8.23 utlisCLI.hpp	75
8.24 README.md File Reference	75
Index	77

Chapter 1

Parallel Kmeans Images Compressor



This program compresses images by reducing the number of colors using k-means clustering. It offers enhanced performance through the implementation of several parallelization techniques. By clustering pixels into k color groups, the program reduces the image's color palette, thereby compressing the image while maintaining visual quality.

1.1 Doxygen Documentation

The documentation of the project can be found [here](#).

1.2 Prerequisites

In order to be able to compile and run the program, there are a few programs that need to be installed.

1.2.1 OpenCV C++ Library

A comprehensive library for computer vision and image processing tasks.

You can refer to the [official page](#) to download.

1.2.2 Mpicc

A C compiler wrapper for parallel programming with the MPI library.

1.2.3 OpenMP

A C++ API for parallel programming on shared-memory systems.

1.3 Getting Started

To compile the project, navigate to the project root directory in your terminal and run the following command:

```
make
```

Once you have compiled you can execute the main program by:

```
./exe
```

1.4 What to expect

Once the program is started, the following screen appears, through which it is possible to compress a new image or decompress an already compressed image.

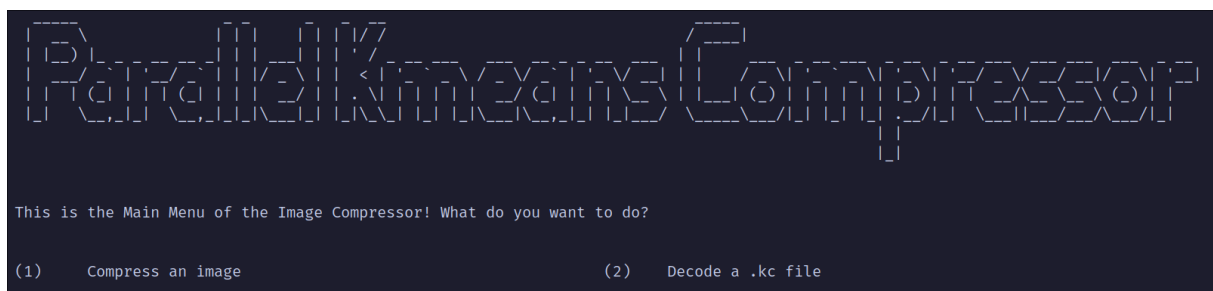


Figure 1.1 alt text

If you choose the "Compress an image" option you can select one type of compressor (sequential, MPI or OpenMP), the type of compression and the path of the original image.

The result image will be created in the output folder and you can rerun `./exe` selecting the decoding function to decode it.

1.5 Project Structure

The project is organized as follows:

1.5.0.1 Folders

- `benchmarkImages`: This folder contains the images used for benchmarking the program. It can be used to test the program's performance.
- `outputs`: This folder contains the compressed images. After installing the program, you may notice that the outputs folder is not present. However, don't worry! It will be automatically created during the first execution of the program.
- `include`: This folder contains the header files of the project. These define the classes and functions that are used in the program.
- `src`: This folder contains the source files of the project. These files contain the implementation of the classes and functions defined in the header files.
- `build`: This folder contains the object files generated during the compilation process.

1.5.0.2 Files and Executables

- `exe`: This is the executable file generated after compiling the project. It is the main program that can be executed to compress or decompress images.
- `Makefile`: This file contains the instructions for compiling the project. It specifies the dependencies and the commands to compile the project.
- `.config`: This file contains the configuration of the program. It is used to store some hyperparameters that can be modified to change the behavior of the program.

1.6 How does it work?

The program compresses images by reducing the number of colors in the image. It does this by clustering the pixels into k color groups using the k -means clustering algorithm. The k -means algorithm is an unsupervised learning algorithm that partitions the data into k clusters based on the similarity of the data points. In the context of image compression, the data points are the pixels of the image, and the clusters are the colors that represent the image.

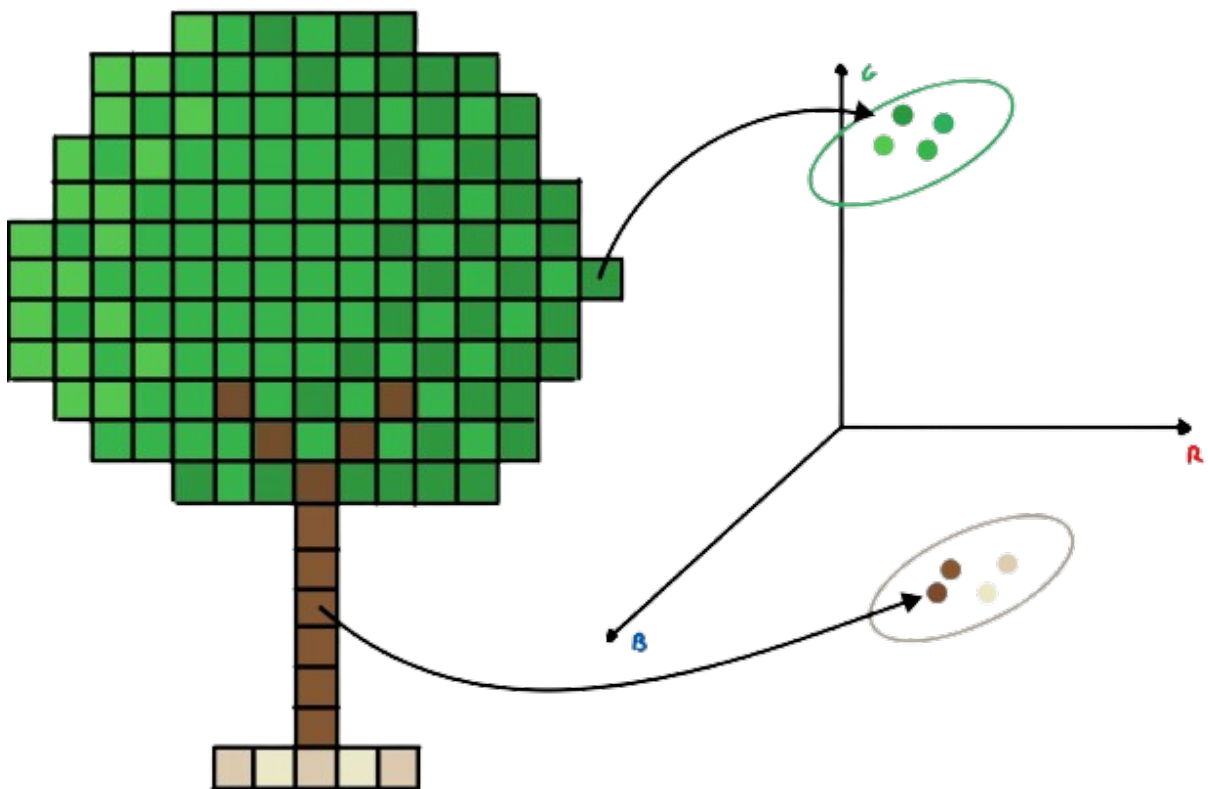


Figure 1.2 tree

The k -means algorithm works as follows:

1. Initialize k centroids randomly.
2. Assign each data point to the nearest centroid.
3. Recompute the centroids based on the data points assigned to them.
4. Repeat steps 2 and 3 until convergence.

The k -means algorithm is an iterative algorithm that converges to a local minimum. The quality of the compression depends on the value of k , the number of clusters. A higher value of k will result in a better representation of the image but will require more memory to store the centroids.

1.7 Parallelization Techniques

The program uses several parallelization techniques to enhance performance. These techniques include:

- **OpenMP:** OpenMP is an API for parallel programming on shared-memory systems. It allows the program to parallelize the computation of the k-means algorithm by distributing the work among multiple threads.
- **MPI:** MPI is a message-passing library for parallel programming on distributed-memory systems. It allows the program to parallelize the computation of the k-means algorithm by distributing the work among multiple processes running on different nodes.

1.8 Benchmarking

The program includes a benchmarking feature that allows you to test the performance of the program on different images. The benchmarking feature measures the time taken to compress an image using different compression techniques and different values of k. The benchmarking results are displayed in a table that shows the time taken to compress the image for each value of k and each compression technique.

1.9 Authors

- [Leonardo Ignazio Pagliochini](#)
- [Francesco Rosnati](#)

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

km	Main namespace for the project	13
km::filesUtils	Provides utility functions for file handling	14
km::imageUtils	Provides utility functions for image processing	17
km::utilsCLI	Provides utility functions for the command-line interface	18

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

km::ConfigReader	23
km::KMeansBase	30
km::KMeansMPI	38
km::KMeansOMP	42
km::KMeansSequential	45
km::KMeansCUDA	34
km::Performance	49
km::Point	53

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

km::ConfigReader	Reads and stores configuration values from a file	23
km::KMeansBase	Base class for K-means clustering algorithm	30
km::KMeansCUDA	Represents the K-means clustering algorithm using CUDA	34
km::KMeansMPI	Represents the K-means clustering algorithm using MPI	38
km::KMeansOMP	Represents the K-means clustering algorithm using OpenMP	42
km::KMeansSequential	Represents the K-means clustering algorithm	45
km::Performance	Represents the performance evaluation	49
km::Point	Represents a point in a feature space	53

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

include/ configReader.hpp	
ConfigReader class declaration	59
include/ filesUtils.hpp	
Utility functions for file handling	61
include/ imagesUtils.hpp	
Utility functions for image processing	63
include/ kmDocs.hpp	
Documentation for the <code>km</code> namespace	64
include/ kMeansBase.hpp	
Base class for K-means clustering algorithm	65
include/ kMeansCUDA.cuh	
Implementation of the K-means clustering algorithm using CUDA	66
include/ kMeansMPI.hpp	
Implementation of the K-means clustering algorithm using MPI	67
include/ kMeansOMP.hpp	
Implementation of the K-means clustering algorithm using OpenMP	68
include/ kMeansSequential.hpp	
Implementation of the K-means clustering algorithm	70
include/ performanceEvaluation.hpp	
Performance evaluation class	71
include/ point.hpp	
Point class representing a point in a feature space	72
include/ utilsCLI.hpp	
Utility functions for the command-line interface	74

Chapter 6

Namespace Documentation

6.1 km Namespace Reference

Main namespace for the project.

Namespaces

- namespace [filesUtils](#)
Provides utility functions for file handling.
- namespace [imageUtils](#)
Provides utility functions for image processing.
- namespace [utilsCLI](#)
Provides utility functions for the command-line interface.

Classes

- class [ConfigReader](#)
Reads and stores configuration values from a file.
- class [KMeansBase](#)
Base class for K-means clustering algorithm.
- class [KMeansCUDA](#)
Represents the K-means clustering algorithm using CUDA.
- class [KMeansMPI](#)
Represents the K-means clustering algorithm using MPI.
- class [KMeansOMP](#)
Represents the K-means clustering algorithm using OpenMP.
- class [KMeansSequential](#)
Represents the K-means clustering algorithm.
- class [Performance](#)
Represents the performance evaluation.
- class [Point](#)
Represents a point in a feature space.

6.1.1 Detailed Description

Main namespace for the project.

The `km` namespace encapsulates various functionalities related to data clustering, file manipulation, and image processing. It is designed to organize core utilities and algorithms used across different modules of the project.

6.2 `km::filesUtils` Namespace Reference

Provides utility functions for file handling.

Functions

- auto `createOutputDirectories` () -> void
Creates output directories.
- auto `writeBinaryFile` (std::string &outputPath, int &width, int &height, int &k, std::vector< `Point` > points, std::vector< `Point` > centroids) -> void
Writes data to a binary file.
- auto `isCorrectExtension` (const std::filesystem::path &filePath, const std::string &correctExtension) -> bool
Checks if a file has the correct extension.
- auto `createDecodingMenu` (std::filesystem::path &decodeDir, std::vector< std::filesystem::path > &imageNames) -> void
Creates a decoding menu.
- auto `readBinaryFile` (std::string &path, cv::Mat &imageCompressed) -> int
Reads a binary file and reconstructs the compressed image.

6.2.1 Detailed Description

Provides utility functions for file handling.

6.2.2 Function Documentation

6.2.2.1 `createDecodingMenu()`

```
auto km::filesUtils::createDecodingMenu (
    std::filesystem::path & decodeDir,
    std::vector< std::filesystem::path > & imageNames ) -> void
```

Creates a decoding menu.

Parameters

<code>decodeDir</code>	Directory for decoding
<code>imageNames</code>	Vector of image names

6.2.2.2 createOutputDirectories()

```
auto km::filesUtils::createOutputDirectories ( ) -> void
```

Creates output directories.

6.2.2.3 isCorrectExtension()

```
auto km::filesUtils::isCorrectExtension (
    const std::filesystem::path & filePath,
    const std::string & correctExtension ) -> bool
```

Checks if a file has the correct extension.

Parameters

<i>filePath</i>	Path of the file
<i>correctExtension</i>	Correct extension to check

Returns

True if the file has the correct extension, false otherwise

6.2.2.4 readBinaryFile()

```
auto km::filesUtils::readBinaryFile (
    std::string & path,
    cv::Mat & imageCompressed ) -> int
```

Reads a binary file and reconstructs the compressed image.

Parameters

<i>path</i>	Path of the binary file
<i>imageCompressed</i>	Compressed image matrix

Returns

Number of clusters

6.2.2.5 writeBinaryFile()

```
auto km::filesUtils::writeBinaryFile (
    std::string & outputPath,
    int & width,
    int & height,
    int & k,
```

```
std::vector< Point > points,  
std::vector< Point > centroids ) -> void
```

Writes data to a binary file.

Parameters

<i>outputPath</i>	Path of the output file
<i>width</i>	Width of the image
<i>height</i>	Height of the image
<i>k</i>	Number of clusters
<i>points</i>	Vector of points
<i>centroids</i>	Vector of centroids

6.3 km::imageUtils Namespace Reference

Provides utility functions for image processing.

Functions

- void [preprocessing](#) (cv::Mat &image, int &typeCompressionChoice)
Performs preprocessing on an image.
- void [defineKValue](#) (int &k, int levelsColorsChoice, std::set< std::vector< unsigned char > > &different_colors)
Defines the value of K based on the color levels choice.
- void [pointsFromImage](#) (cv::Mat &image, std::vector< [Point](#) > &points, std::set< std::vector< unsigned char > > &different_colors)
Extracts points from an image.

6.3.1 Detailed Description

Provides utility functions for image processing.

6.3.2 Function Documentation

6.3.2.1 defineKValue()

```
void km::imageUtils::defineKValue (
    int & k,
    int levelsColorsChoice,
    std::set< std::vector< unsigned char > > & different_colors )
```

Defines the value of K based on the color levels choice.

Parameters

<i>k</i>	Value of K
<i>levelsColorsChoice</i>	Levels of colors choice
<i>different_colors</i>	Set of different colors in the image

6.3.2.2 pointsFromImage()

```
void km::imageUtils::pointsFromImage (
    cv::Mat & image,
    std::vector< Point > & points,
    std::set< std::vector< unsigned char > > & different_colors )
```

Extracts points from an image.

Parameters

<i>image</i>	Input image
<i>points</i>	Vector of points
<i>different_colors</i>	Set of different colors in the image

6.3.2.3 preprocessing()

```
void km::imageUtils::preprocessing (
    cv::Mat & image,
    int & typeCompressionChoice )
```

Performs preprocessing on an image.

Parameters

<i>image</i>	Input image
<i>typeCompressionChoice</i>	Type of compression choice

6.4 km::utilsCLI Namespace Reference

Provides utility functions for the command-line interface.

Functions

- void [sequentialEncoderHeader](#) ()
Displays the header for the sequential encoder.
- void [mpiEncoderHeader](#) ()
Displays the header for the MPI encoder.
- void [ompEncoderHeader](#) ()
Displays the header for the OpenMP encoder.
- void [mainMenuHeader](#) ()
Displays the main menu header.
- void [decoderHeader](#) ()
Displays the decoder header.
- void [workDone](#) ()
Displays the work done message.

- void [compressionChoices](#) (int &levelsColorsChoice, int &typeCompressionChoice, std::string &outputPath, cv::Mat &image, int executionStandard)
Handles the compression choices.
- void [printCompressionInformations](#) (int &originalWidth, int &originalHeight, int &width, int &height, int &k, size_t &different_colors_size)
Prints the compression information.
- void [displayDecodingMenu](#) (std::string &path, std::vector< std::filesystem::path > &imageNames, std::filesystem::path &decodeDir)
Displays the decoding menu.

6.4.1 Detailed Description

Provides utility functions for the command-line interface.

6.4.2 Function Documentation

6.4.2.1 [compressionChoices\(\)](#)

```
void km::utilsCLI::compressionChoices (
    int & levelsColorsChoice,
    int & typeCompressionChoice,
    std::string & outputPath,
    cv::Mat & image,
    int executionStandard )
```

Handles the compression choices.

Parameters

<i>levelsColorsChoice</i>	Choice of color levels
<i>typeCompressionChoice</i>	Choice of compression type
<i>outputPath</i>	Output path
<i>image</i>	Input image
<i>executionStandard</i>	Execution standard

6.4.2.2 [decoderHeader\(\)](#)

```
void km::utilsCLI::decoderHeader ( )
```

Displays the decoder header.

6.4.2.3 [displayDecodingMenu\(\)](#)

```
void km::utilsCLI::displayDecodingMenu (
    std::string & path,
    std::vector< std::filesystem::path > & imageNames,
    std::filesystem::path & decodeDir )
```

Displays the decoding menu.

Parameters

<i>path</i>	Path of the directory containing the compressed images
<i>imageNames</i>	Vector of image names
<i>decodeDir</i>	Path of the decoding directory

6.4.2.4 mainMenuHeader()

```
void km::utilsCLI::mainMenuHeader ( )
```

Displays the main menu header.

6.4.2.5 mpiEncoderHeader()

```
void km::utilsCLI::mpiEncoderHeader ( )
```

Displays the header for the MPI encoder.

6.4.2.6 ompEncoderHeader()

```
void km::utilsCLI::ompEncoderHeader ( )
```

Displays the header for the OpenMP encoder.

6.4.2.7 printCompressionInformations()

```
void km::utilsCLI::printCompressionInformations (
    int & originalWidth,
    int & originalHeight,
    int & width,
    int & height,
    int & k,
    size_t & different_colors_size )
```

Prints the compression information.

Parameters

<i>originalWidth</i>	Original width of the image
<i>originalHeight</i>	Original height of the image
<i>width</i>	Width of the compressed image
<i>height</i>	Height of the compressed image
<i>k</i>	Number of clusters
<i>different_colors_size</i>	Number of different colors

6.4.2.8 sequentialEncoderHeader()

```
void km::utilsCLI::sequentialEncoderHeader ( )
```

Displays the header for the sequential encoder.

6.4.2.9 workDone()

```
void km::utilsCLI::workDone ( )
```

Displays the work done message.

Chapter 7

Class Documentation

7.1 km::ConfigReader Class Reference

Reads and stores configuration values from a file.

```
#include <configReader.hpp>
```

Collaboration diagram for km::ConfigReader:

km::ConfigReader
<ul style="list-style-type: none"> - double first_level_compression_color - double second_level_compression_color - double third_level_compression_color - double fourth_level_compression_color - double fifth_level_compression_color - double resizing_factor - int color_choice - int compression_choice - std::filesystem::path inputImagePath - std::regex pattern - std::unordered_set < std::string > requiredVariables
<ul style="list-style-type: none"> + auto getFirstLevelCompressionColor() const -> double + auto getSecondLevelCompressionColor() const -> double + auto getThirdLevelCompressionColor() const -> double + auto getFourthLevelCompressionColor() const -> double + auto getFifthLevelCompressionColor() const -> double + auto getColorChoice() const -> int + auto getCompressionChoice() const -> int + auto getInputImagePath() const -> std::filesystem::path + auto getResizingFactor() const -> double + auto readConfigFile() -> bool + ConfigReader() - auto checkVariableExists(const std::string &variableName) const -> bool

Public Member Functions

- auto [getFirstLevelCompressionColor](#) () const -> double
Gets the first level compression color value.
- auto [getSecondLevelCompressionColor](#) () const -> double
Gets the second level compression color value.
- auto [getThirdLevelCompressionColor](#) () const -> double

- Gets the third level compression color value.*
 - auto `getFourthLevelCompressionColor` () const -> double
- Gets the fourth level compression color value.*
 - auto `getFifthLevelCompressionColor` () const -> double
- Gets the fifth level compression color value.*
 - auto `getColorChoice` () const -> int
- Gets the color choice.*
 - auto `getCompressionChoice` () const -> int
- Gets the compression choice.*
 - auto `getInputImageFilePath` () const -> std::filesystem::path
- Gets the input image file path.*
 - auto `getResizingFactor` () const -> double
- Gets the resizing factor.*
 - auto `readConfigFile` () -> bool
- Reads the configuration file.*
 - `ConfigReader` ()

Private Member Functions

- auto `checkVariableExists` (const std::string &variableName) const -> bool

Private Attributes

- double `first_level_compression_color` = 0.
First level compression color value.
- double `second_level_compression_color` = 0.
Second level compression color value.
- double `third_level_compression_color` = 0.
Third level compression color value.
- double `fourth_level_compression_color` = 0.
Fourth level compression color value.
- double `fifth_level_compression_color` = 0.
Fifth level compression color value.
- double `resizing_factor` = 0.
Resizing factor.
- int `color_choice` = 0
Color choice.
- int `compression_choice` = 0
Compression choice.
- std::filesystem::path `inputImageFilePath`
Input image file path.
- std::regex `pattern`
Regular expression pattern.
- std::unordered_set< std::string > `requiredVariables` = {}
Set of required variables.

7.1.1 Detailed Description

Reads and stores configuration values from a file.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ConfigReader()

```
km::ConfigReader::ConfigReader ( )
```

7.1.3 Member Function Documentation

7.1.3.1 checkVariableExists()

```
auto km::ConfigReader::checkVariableExists (
    const std::string & variableName ) const -> bool [private]
```

7.1.3.2 getColorChoice()

```
auto km::ConfigReader::getColorChoice ( ) const -> int
```

Gets the color choice.

Returns

Color choice

7.1.3.3 getCompressionChoice()

```
auto km::ConfigReader::getCompressionChoice ( ) const -> int
```

Gets the compression choice.

Returns

Compression choice

7.1.3.4 getFifthLevelCompressionColor()

```
auto km::ConfigReader::getFifthLevelCompressionColor ( ) const -> double
```

Gets the fifth level compression color value.

Returns

Fifth level compression color value

7.1.3.5 getFirstLevelCompressionColor()

```
auto km::ConfigReader::getFirstLevelCompressionColor ( ) const -> double
```

Gets the first level compression color value.

Returns

First level compression color value

7.1.3.6 getFourthLevelCompressionColor()

```
auto km::ConfigReader::getFourthLevelCompressionColor ( ) const -> double
```

Gets the fourth level compression color value.

Returns

Fourth level compression color value

7.1.3.7 getInputImageFilePath()

```
auto km::ConfigReader::getInputImageFilePath ( ) const -> std::filesystem::path
```

Gets the input image file path.

Returns

Input image file path

7.1.3.8 getResizingFactor()

```
auto km::ConfigReader::getResizingFactor ( ) const -> double
```

Gets the resizing factor.

Returns

Resizing factor

7.1.3.9 getSecondLevelCompressionColor()

```
auto km::ConfigReader::getSecondLevelCompressionColor ( ) const -> double
```

Gets the second level compression color value.

Returns

Second level compression color value

7.1.3.10 getThirdLevelCompressionColor()

```
auto km::ConfigReader::getThirdLevelCompressionColor ( ) const -> double
```

Gets the third level compression color value.

Returns

Third level compression color value

7.1.3.11 readConfigFile()

```
auto km::ConfigReader::readConfigFile ( ) -> bool
```

Reads the configuration file.

Returns

True if the configuration file is read successfully, false otherwise

7.1.4 Member Data Documentation

7.1.4.1 color_choice

```
int km::ConfigReader::color_choice = 0 [private]
```

Color choice.

7.1.4.2 compression_choice

```
int km::ConfigReader::compression_choice = 0 [private]
```

Compression choice.

7.1.4.3 fifth_level_compression_color

```
double km::ConfigReader::fifth_level_compression_color = 0. [private]
```

Fifth level compression color value.

7.1.4.4 first_level_compression_color

```
double km::ConfigReader::first_level_compression_color = 0. [private]
```

First level compression color value.

7.1.4.5 fourth_level_compression_color

```
double km::ConfigReader::fourth_level_compression_color = 0. [private]
```

Fourth level compression color value.

7.1.4.6 inputImagePath

```
std::filesystem::path km::ConfigReader::inputImagePath [private]
```

Input image file path.

7.1.4.7 pattern

```
std::regex km::ConfigReader::pattern [private]
```

Regular expression pattern.

7.1.4.8 requiredVariables

```
std::unordered_set<std::string> km::ConfigReader::requiredVariables = {} [private]
```

Set of required variables.

7.1.4.9 resizing_factor

```
double km::ConfigReader::resizing_factor = 0. [private]
```

Resizing factor.

7.1.4.10 second_level_compression_color

```
double km::ConfigReader::second_level_compression_color = 0. [private]
```

Second level compression color value.

7.1.4.11 third_level_compression_color

```
double km::ConfigReader::third_level_compression_color = 0. [private]
```

Third level compression color value.

The documentation for this class was generated from the following file:

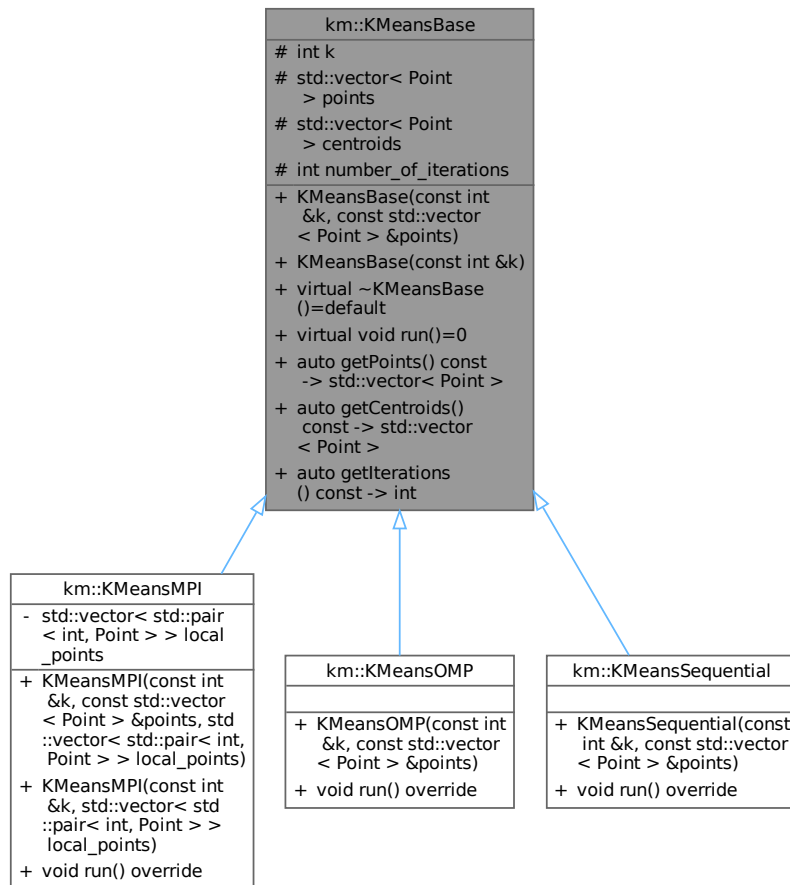
- include/[configReader.hpp](#)

7.2 km::KMeansBase Class Reference

Base class for K-means clustering algorithm.

```
#include <kMeansBase.hpp>
```

Inheritance diagram for km::KMeansBase:



Collaboration diagram for km::KMeansBase:

km::KMeansBase
<pre># int k # std::vector< Point > points # std::vector< Point > centroids # int number_of_iterations</pre>
<pre>+ KMeansBase(const int &k, const std::vector < Point > &points) + KMeansBase(const int &k) + virtual ~KMeansBase ()=default + virtual void run()=0 + auto getPoints() const -> std::vector< Point > + auto getCentroids() const -> std::vector < Point > + auto getIterations () const -> int</pre>

Public Member Functions

- [KMeansBase](#) (const int &k, const std::vector< [Point](#) > &points)
Constructor for [KMeansBase](#).
- [KMeansBase](#) (const int &k)
Constructs a [KMeansBase](#) object only with the specified number of clusters.
- virtual [~KMeansBase](#) ()=default
Virtual destructor for [KMeansBase](#).
- virtual void [run](#) ()=0
Runs the K-means clustering algorithm.
- auto [getPoints](#) () const -> std::vector< [Point](#) >
Gets the poinots.
- auto [getCentroids](#) () const -> std::vector< [Point](#) >
Gets the centroids.
- auto [getIterations](#) () const -> int
Gets the number of iterations.

Protected Attributes

- int `k`
Number of clusters.
- `std::vector< Point > points`
Vector of points.
- `std::vector< Point > centroids`
Vector of centroids.
- int `number_of_iterations`
Number of iterations.

7.2.1 Detailed Description

Base class for K-means clustering algorithm.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 KMeansBase() [1/2]

```
km::KMeansBase::KMeansBase (
    const int & k,
    const std::vector< Point > & points )
```

Constructor for `KMeansBase`.

Parameters

<code>k</code>	Number of clusters
<code>points</code>	Vector of points

7.2.2.2 KMeansBase() [2/2]

```
km::KMeansBase::KMeansBase (
    const int & k )
```

Constructs a `KMeansBase` object only with the specified number of clusters.

Parameters

<code>k</code>	The number of clusters.
----------------	-------------------------

7.2.2.3 ~KMeansBase()

```
virtual km::KMeansBase::~~KMeansBase ( ) [virtual], [default]
```

Virtual destructor for `KMeansBase`.

7.2.3 Member Function Documentation

7.2.3.1 getCentroids()

```
auto km::KMeansBase::getCentroids ( ) const -> std::vector< Point >
```

Gets the centroids.

Returns

Vector of centroids

7.2.3.2 getIterations()

```
auto km::KMeansBase::getIterations ( ) const -> int
```

Gets the number of iterations.

Returns

Number of iterations

7.2.3.3 getPoints()

```
auto km::KMeansBase::getPoints ( ) const -> std::vector< Point >
```

Gets the poinots.

Returns

Vector of points

7.2.3.4 run()

```
virtual void km::KMeansBase::run ( ) [pure virtual]
```

Runs the K-means clustering algorithm.

Implemented in [km::KMeansMPI](#), [km::KMeansOMP](#), and [km::KMeansSequential](#).

7.2.4 Member Data Documentation

7.2.4.1 centroids

```
std::vector<Point> km::KMeansBase::centroids [protected]
```

Vector of centroids.

7.2.4.2 k

```
int km::KMeansBase::k [protected]
```

Number of clusters.

7.2.4.3 number_of_iterations

```
int km::KMeansBase::number_of_iterations [protected]
```

Number of iterations.

7.2.4.4 points

```
std::vector<Point> km::KMeansBase::points [protected]
```

Vector of points.

The documentation for this class was generated from the following file:

- [include/kMeansBase.hpp](#)

7.3 km::KMeansCUDA Class Reference

Represents the K-means clustering algorithm using CUDA.

Collaboration diagram for km::KMeansCUDA:

km::KMeansCUDA
<ul style="list-style-type: none"> - int k - std::vector< Point > points - std::vector< Point > centroids - int number_of_iterations
<ul style="list-style-type: none"> + KMeansCUDA(const int &k, const std::vector< Point > &points) + void run() + void printClusters() const + void plotClusters() + auto getPoints() -> std::vector< Point > + auto getCentroids() -> std::vector< Point > + auto getIterations() -> int

Public Member Functions

- **KMeansCUDA** (const int &*k*, const std::vector< [Point](#) > &*points*)
Constructor for KMeans.
- void **run** ()
Runs the K-means clustering algorithm using CUDA.
- void **printClusters** () const
Prints the clusters.
- void **plotClusters** ()
Plots the clusters.
- auto **getPoints** () -> std::vector< [Point](#) >
Gets the points.
- auto **getCentroids** () -> std::vector< [Point](#) >
Gets the centroids.
- auto **getIterations** () -> int
Gets the number of iterations.

Private Attributes

- int **k**
Number of clusters.
- std::vector< [Point](#) > **points**
Vector of points.
- std::vector< [Point](#) > **centroids**
Vector of centroids.
- int **number_of_iterations**
Number of iterations.

7.3.1 Detailed Description

Represents the K-means clustering algorithm using CUDA.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 KMeansCUDA()

```
km::KMeansCUDA::KMeansCUDA (
    const int & k,
    const std::vector< Point > & points )
```

Constructor for KMeans.

Parameters

<i>k</i>	Number of clusters
<i>points</i>	Vector of points

7.3.3 Member Function Documentation

7.3.3.1 getCentroids()

```
auto km::KMeansCUDA::getCentroids ( ) -> std::vector< Point >
```

Gets the centroids.

Returns

Vector of centroids

7.3.3.2 getIterations()

```
auto km::KMeansCUDA::getIterations ( ) -> int
```

Gets the number of iterations.

Returns

Number of iterations

7.3.3.3 getPoints()

```
auto km::KMeansCUDA::getPoints ( ) -> std::vector< Point >
```

Gets the points.

Returns

Vector of points

7.3.3.4 plotClusters()

```
void km::KMeansCUDA::plotClusters ( )
```

Plots the clusters.

7.3.3.5 printClusters()

```
void km::KMeansCUDA::printClusters ( ) const
```

Prints the clusters.

7.3.3.6 run()

```
void km::KMeansCUDA::run ( )
```

Runs the K-means clustering algorithm using CUDA.

7.3.4 Member Data Documentation

7.3.4.1 centroids

```
std::vector<Point> km::KMeansCUDA::centroids [private]
```

Vector of centroids.

7.3.4.2 k

```
int km::KMeansCUDA::k [private]
```

Number of clusters.

7.3.4.3 number_of_iterations

```
int km::KMeansCUDA::number_of_iterations [private]
```

Number of iterations.

7.3.4.4 points

```
std::vector<Point> km::KMeansCUDA::points [private]
```

Vector of points.

The documentation for this class was generated from the following file:

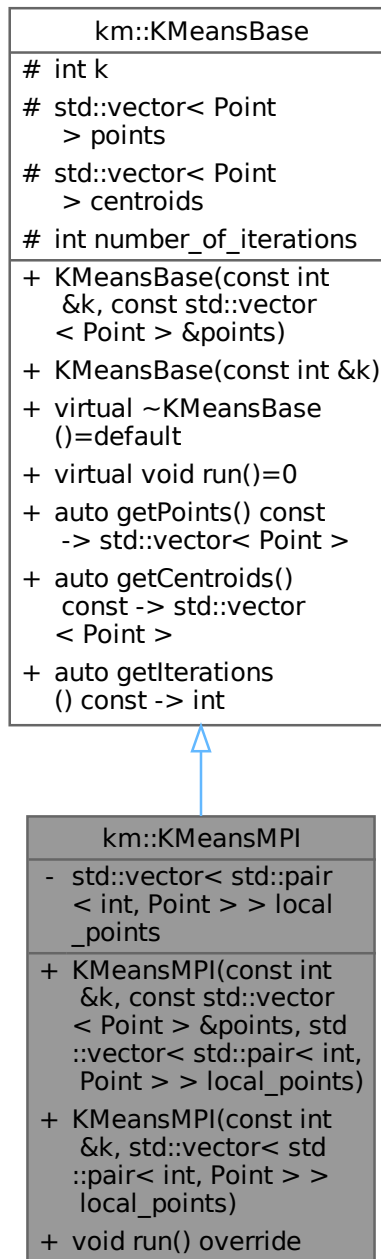
- include/[kMeansCUDA.cuh](#)

7.4 km::KMeansMPI Class Reference

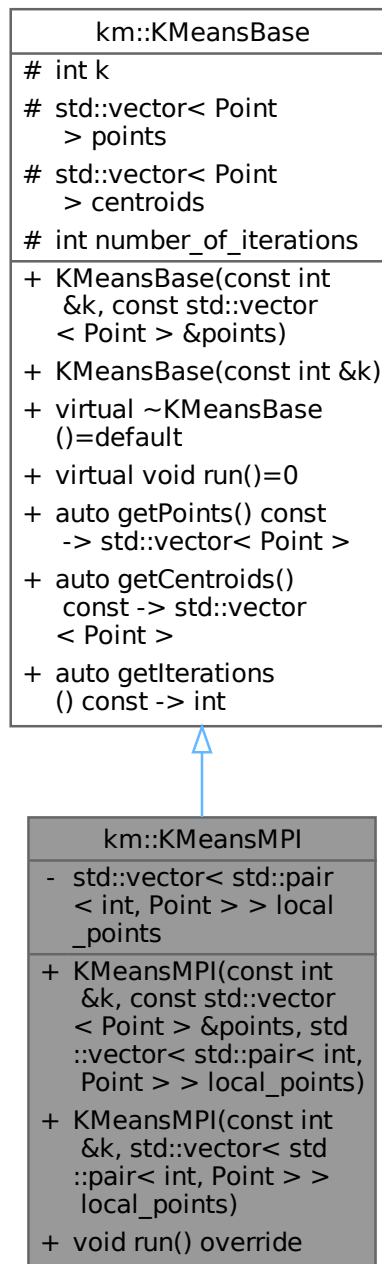
Represents the K-means clustering algorithm using MPI.

```
#include <kMeansMPI.hpp>
```

Inheritance diagram for km::KMeansMPI:



Collaboration diagram for km::KMeansMPI:



Public Member Functions

- `KMeansMPI` (const int &k, const std::vector< [Point](#) > &points, std::vector< std::pair< int, [Point](#) > > local_points)
Constructor for [KMeansMPI](#).
- `KMeansMPI` (const int &k, std::vector< std::pair< int, [Point](#) > > local_points)
Constructor for [KMeansMPI](#).

- void `run()` override
Runs the K-means clustering algorithm using MPI.

Public Member Functions inherited from `km::KMeansBase`

- `KMeansBase` (const int &`k`, const std::vector< `Point` > &`points`)
Constructor for `KMeansBase`.
- `KMeansBase` (const int &`k`)
Constructs a `KMeansBase` object only with the specified number of clusters.
- virtual `~KMeansBase` ()=default
Virtual destructor for `KMeansBase`.
- auto `getPoints` () const -> std::vector< `Point` >
Gets the points.
- auto `getCentroids` () const -> std::vector< `Point` >
Gets the centroids.
- auto `getIterations` () const -> int
Gets the number of iterations.

Private Attributes

- std::vector< std::pair< int, `Point` > > `local_points`
Vector of local points.

Additional Inherited Members

Protected Attributes inherited from `km::KMeansBase`

- int `k`
Number of clusters.
- std::vector< `Point` > `points`
Vector of points.
- std::vector< `Point` > `centroids`
Vector of centroids.
- int `number_of_iterations`
Number of iterations.

7.4.1 Detailed Description

Represents the K-means clustering algorithm using MPI.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 `KMeansMPI()` [1/2]

```
km::KMeansMPI::KMeansMPI (
    const int & k,
    const std::vector< Point > & points,
    std::vector< std::pair< int, Point > > local_points )
```

Constructor for `KMeansMPI`.

Parameters

<i>k</i>	Number of clusters
<i>points</i>	Vector of points

7.4.2.2 KMeansMPI() [2/2]

```
km::KMeansMPI::KMeansMPI (
    const int & k,
    std::vector< std::pair< int, Point > > local_points )
```

Constructor for [KMeansMPI](#).

Parameters

<i>k</i>	Number of clusters
----------	--------------------

7.4.3 Member Function Documentation

7.4.3.1 run()

```
void km::KMeansMPI::run ( ) [override], [virtual]
```

Runs the K-means clustering algorithm using MPI.

Implements [km::KMeansBase](#).

7.4.4 Member Data Documentation

7.4.4.1 local_points

```
std::vector<std::pair<int, Point> > km::KMeansMPI::local_points [private]
```

Vector of local points.

The documentation for this class was generated from the following file:

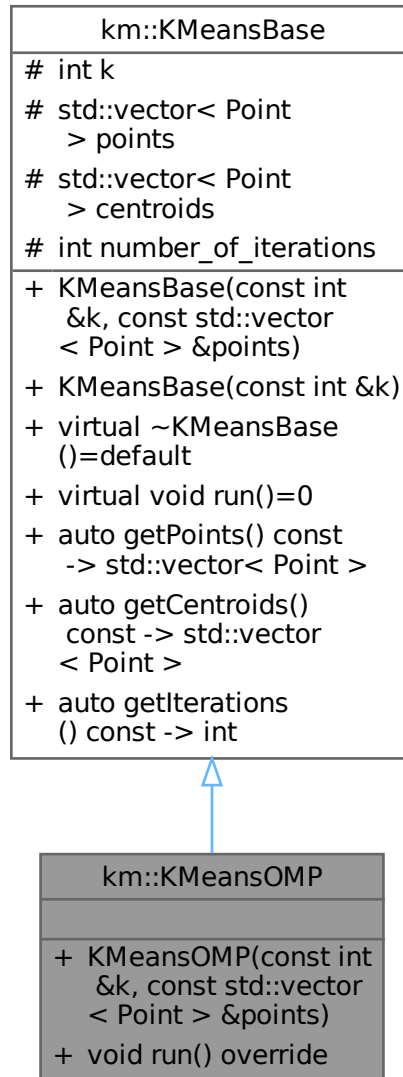
- [include/kMeansMPI.hpp](#)

7.5 km::KMeansOMP Class Reference

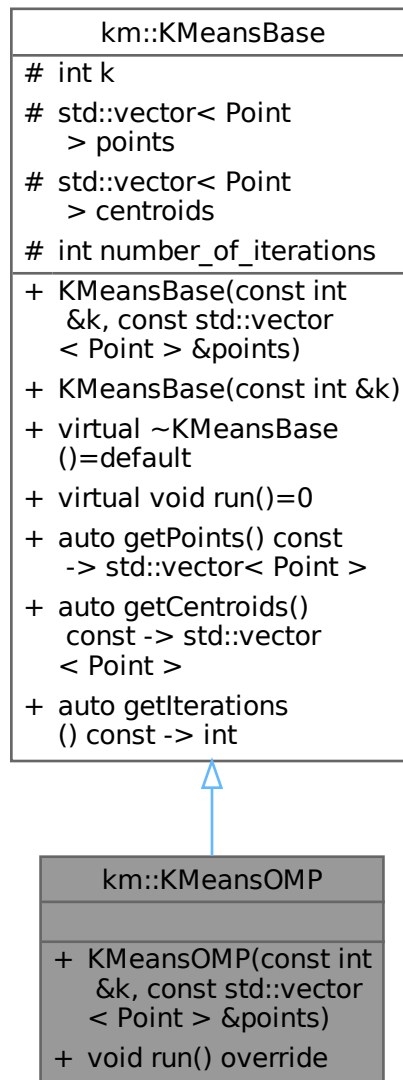
Represents the K-means clustering algorithm using OpenMP.

```
#include <kMeansOMP.hpp>
```

Inheritance diagram for km::KMeansOMP:



Collaboration diagram for km::KMeansOMP:



Public Member Functions

- [KMeansOMP](#) (const int &k, const std::vector< [Point](#) > &points)
Constructor for [KMeansOMP](#).
- void [run](#) () override
Runs the K-means clustering algorithm using OpenMP.

Public Member Functions inherited from [km::KMeansBase](#)

- [KMeansBase](#) (const int &k, const std::vector< [Point](#) > &points)

- Constructor for *KMeansBase*.

 - *KMeansBase* (const int &k)
Constructs a *KMeansBase* object only with the specified number of clusters.
- virtual *~KMeansBase* ()=default
Virtual destructor for *KMeansBase*.
- auto *getPoints* () const -> std::vector< *Point* >
Gets the poinots.
- auto *getCentroids* () const -> std::vector< *Point* >
Gets the centroids.
- auto *getIterations* () const -> int
Gets the number of iterations.

Additional Inherited Members

Protected Attributes inherited from *km::KMeansBase*

- int *k*
Number of clusters.
- std::vector< *Point* > *points*
Vector of points.
- std::vector< *Point* > *centroids*
Vector of centroids.
- int *number_of_iterations*
Number of iterations.

7.5.1 Detailed Description

Represents the K-means clustering algorithm using OpenMP.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 KMeansOMP()

```
km::KMeansOMP::KMeansOMP (
    const int & k,
    const std::vector< Point > & points )
```

Constructor for *KMeansOMP*.

Parameters

<i>k</i>	Number of clusters
<i>points</i>	Vector of points

7.5.3 Member Function Documentation

7.5.3.1 run()

```
void km::KMeansOMP::run ( ) [override], [virtual]
```

Runs the K-means clustering algorithm using OpenMP.

Implements [km::KMeansBase](#).

The documentation for this class was generated from the following file:

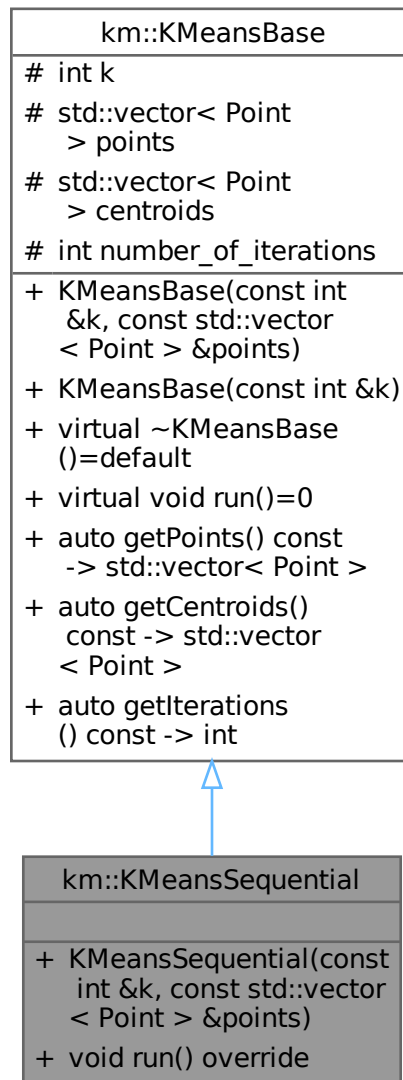
- [include/kMeansOMP.hpp](#)

7.6 km::KMeansSequential Class Reference

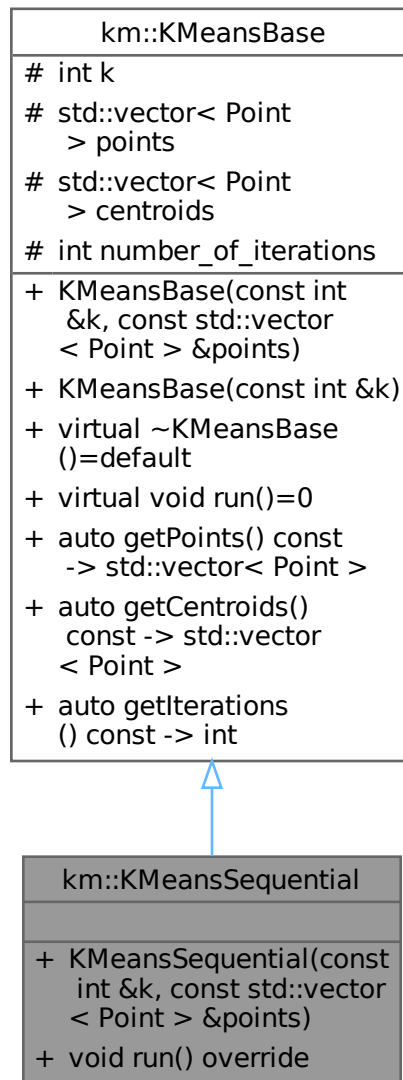
Represents the K-means clustering algorithm.

```
#include <kMeansSequential.hpp>
```

Inheritance diagram for km::KMeansSequential:



Collaboration diagram for km::KMeansSequential:



Public Member Functions

- [KMeansSequential](#) (const int &k, const std::vector< [Point](#) > &points)
Constructor for [KMeansSequential](#).
- void [run](#) () override
Runs the K-means clustering algorithm.

Public Member Functions inherited from [km::KMeansBase](#)

- [KMeansBase](#) (const int &k, const std::vector< [Point](#) > &points)

- Constructor for *KMeansBase*.

 - *KMeansBase* (const int &k)
Constructs a *KMeansBase* object only with the specified number of clusters.
 - virtual *~KMeansBase* ()=default
Virtual destructor for *KMeansBase*.
 - auto *getPoints* () const -> std::vector< *Point* >
Gets the poinots.
 - auto *getCentroids* () const -> std::vector< *Point* >
Gets the centroids.
 - auto *getIterations* () const -> int
Gets the number of iterations.

Additional Inherited Members

Protected Attributes inherited from *km::KMeansBase*

- int *k*
Number of clusters.
- std::vector< *Point* > *points*
Vector of points.
- std::vector< *Point* > *centroids*
Vector of centroids.
- int *number_of_iterations*
Number of iterations.

7.6.1 Detailed Description

Represents the K-means clustering algorithm.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 *KMeansSequential*()

```
km::KMeansSequential::KMeansSequential (
    const int & k,
    const std::vector< Point > & points )
```

Constructor for *KMeansSequential*.

Parameters

<i>k</i>	Number of clusters
<i>points</i>	Vector of points

7.6.3 Member Function Documentation

7.6.3.1 run()

```
void km::KMeansSequential::run ( ) [override], [virtual]
```

Runs the K-means clustering algorithm.

Implements [km::KMeansBase](#).

The documentation for this class was generated from the following file:

- [include/kMeansSequential.hpp](#)

7.7 km::Performance Class Reference

Represents the performance evaluation.

```
#include <performanceEvaluation.hpp>
```

Collaboration diagram for km::Performance:

km::Performance
<ul style="list-style-type: none"> - std::string img - int choice - std::string method
<ul style="list-style-type: none"> + Performance() + auto writeCSV(int different_colors_size, int k, int n_points, double elapsedKmeans, int number_of_iterations, int num_processes=0) -> void + auto fillPerformance(int choice, const std::string &img, const std::string &method) -> void + static auto extractFileName(const std::string &outputPath) -> std::string - auto createOrOpenCSV(const std::string &filename) -> void - auto appendToCSV(const std::string &filename, int n_diff_colors, int k, int n_colors, const std::string &compType, double time, int num_processes, int number_of_iteratios) -> void

Public Member Functions

- [Performance](#) ()
Default constructor.
- auto [writeCSV](#) (int different_colors_size, int k, int n_points, double elapsedKmeans, int number_of_iterations, int num_processes=0) -> void
Writes performance data to a CSV file.
- auto [fillPerformance](#) (int [choice](#), const std::string &[img](#), const std::string &[method](#)) -> void
Fills the performance data.

Static Public Member Functions

- static auto [extractFileName](#) (const std::string &outputPath) -> std::string
Extracts the file name from the output path.

Private Member Functions

- auto [createOrOpenCSV](#) (const std::string &filename) -> void
Creates or opens a CSV file.
- auto [appendToCSV](#) (const std::string &filename, int n_diff_colors, int k, int n_colors, const std::string &comp←
Type, double time, int num_processes, int number_of_iteratios) -> void
Appends performance data to the CSV file.

Private Attributes

- std::string [img](#)
Image name.
- int [choice](#) {}
Choice of performance evaluation.
- std::string [method](#)
Method used.

7.7.1 Detailed Description

Represents the performance evaluation.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Performance()

```
km::Performance::Performance ( )
```

Default constructor.

7.7.3 Member Function Documentation

7.7.3.1 appendToCSV()

```
auto km::Performance::appendToCSV (
    const std::string & filename,
    int n_diff_colors,
    int k,
    int n_colors,
    const std::string & compType,
    double time,
    int num_processes,
    int number_of_iteratios ) -> void [private]
```

Appends performance data to the CSV file.

Parameters

<i>filename</i>	Name of the CSV file
<i>n_diff_colors</i>	Number of different colors
<i>k</i>	Number of clusters
<i>n_colors</i>	Number of colors
<i>compType</i>	Compression type
<i>time</i>	Elapsed time
<i>num_processes</i>	Number of processes

7.7.3.2 createOrOpenCSV()

```
auto km::Performance::createOrOpenCSV (
    const std::string & filename ) -> void [private]
```

Creates or opens a CSV file.

Parameters

<i>filename</i>	Name of the CSV file
-----------------	----------------------

7.7.3.3 extractFileName()

```
static auto km::Performance::extractFileName (
    const std::string & outputPath ) -> std::string [static]
```

Extracts the file name from the output path.

Parameters

<i>outputPath</i>	Output path
-------------------	-------------

Returns

Extracted file name

7.7.3.4 fillPerformance()

```
auto km::Performance::fillPerformance (
    int choice,
    const std::string & img,
    const std::string & method ) -> void
```

Fills the performance data.

Parameters

<i>choice</i>	Choice of performance evaluation
<i>img</i>	Image name
<i>method</i>	Method used

7.7.3.5 writeCSV()

```
auto km::Performance::writeCSV (
    int different_colors_size,
    int k,
    int n_points,
    double elapsedKmeans,
    int number_of_iterations,
    int num_processes = 0 ) -> void
```

Writes performance data to a CSV file.

Parameters

<i>different_colors_size</i>	Number of different colors
<i>k</i>	Number of clusters
<i>n_points</i>	Number of points
<i>elapsedKmeans</i>	Elapsed time for K-means clustering
<i>num_processes</i>	Number of processes (optional, default=0)

7.7.4 Member Data Documentation

7.7.4.1 choice

```
int km::Performance::choice {} [private]
```

Choice of performance evaluation.

7.7.4.2 img

```
std::string km::Performance::img [private]
```

Image name.

7.7.4.3 method

```
std::string km::Performance::method [private]
```

Method used.

The documentation for this class was generated from the following file:

- include/[performanceEvaluation.hpp](#)

7.8 km::Point Class Reference

Represents a point in a feature space.

```
#include <point.hpp>
```

Collaboration diagram for km::Point:

km::Point
<ul style="list-style-type: none">+ int id+ unsigned char r+ unsigned char g+ unsigned char b+ int clusterId
<ul style="list-style-type: none">+ Point()+ Point(const int &id, const std::vector< int > &coordinates)+ auto distance(const Point &p) const -> double+ auto getFeature(int index) -> unsigned char &+ auto getFeature_int(int index) const -> int+ auto setFeature(int index, int x) -> void

Public Member Functions

- [Point](#) ()
Constructor for [Point](#).
- [Point](#) (const int &[id](#), const std::vector< int > &coordinates)
Constructor for [Point](#).
- auto [distance](#) (const [Point](#) &p) const -> double
Calculates the distance between this point and another point.
- auto [getFeature](#) (int index) -> unsigned char &
Gets a feature value at the specified index.
- auto [getFeature_int](#) (int index) const -> int
Gets a feature value as an integer at the specified index.
- auto [setFeature](#) (int index, int x) -> void
Sets a feature value at the specified index.

Public Attributes

- int [id](#) {0}
ID of the point.
- unsigned char [r](#) {0}
Red component.
- unsigned char [g](#) {0}
Green component.
- unsigned char [b](#) {0}
Blue component.
- int [clusterId](#) {-1}
ID of the cluster the point belongs to.

7.8.1 Detailed Description

Represents a point in a feature space.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 [Point](#)() [1/2]

```
km::Point::Point ( )
```

Constructor for [Point](#).

Parameters

<i>features_size</i>	Number of features
----------------------	--------------------

7.8.2.2 [Point](#)() [2/2]

```
km::Point::Point (
```

```
const int & id,  
const std::vector< int > & coordinates )
```

Constructor for [Point](#).

Parameters

<i>id</i>	ID of the point
<i>coordinates</i>	Coordinates of the point

7.8.3 Member Function Documentation

7.8.3.1 distance()

```
auto km::Point::distance (  
    const Point & p ) const -> double
```

Calculates the distance between this point and another point.

Parameters

<i>p</i>	Other point
----------	-------------

Returns

Distance between the points

7.8.3.2 getFeature()

```
auto km::Point::getFeature (  
    int index ) -> unsigned char &
```

Gets a feature value at the specified index.

Parameters

<i>index</i>	Index of the feature
--------------	----------------------

Returns

Feature value

7.8.3.3 getFeature_int()

```
auto km::Point::getFeature_int (  
    int index ) const -> int
```

Gets a feature value as an integer at the specified index.

Parameters

<i>index</i>	Index of the feature
--------------	----------------------

Returns

Feature value as an integer

7.8.3.4 setFeature()

```
auto km::Point::setFeature (
    int index,
    int x ) -> void
```

Sets a feature value at the specified index.

Parameters

<i>index</i>	Index of the feature
<i>x</i>	Feature value

7.8.4 Member Data Documentation**7.8.4.1 b**

```
unsigned char km::Point::b {0}
```

Blue component.

7.8.4.2 clusterId

```
int km::Point::clusterId {-1}
```

ID of the cluster the point belongs to.

7.8.4.3 g

```
unsigned char km::Point::g {0}
```

Green component.

7.8.4.4 id

```
int km::Point::id {0}
```

ID of the point.

7.8.4.5 r

```
unsigned char km::Point::r {0}
```

Red component.

The documentation for this class was generated from the following file:

- include/[point.hpp](#)

Chapter 8

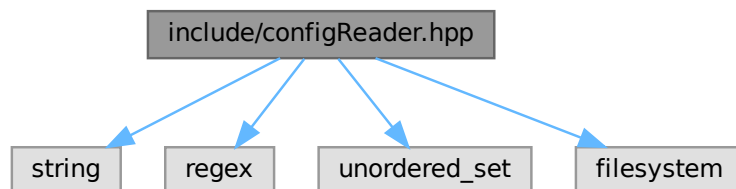
File Documentation

8.1 include/configReader.hpp File Reference

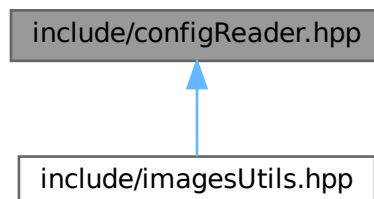
ConfigReader class declaration.

```
#include <string>
#include <regex>
#include <unordered_set>
#include <filesystem>
```

Include dependency graph for configReader.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [km::ConfigReader](#)
Reads and stores configuration values from a file.

Namespaces

- namespace [km](#)
Main namespace for the project.

8.1.1 Detailed Description

ConfigReader class declaration.

8.2 configReader.hpp

[Go to the documentation of this file.](#)

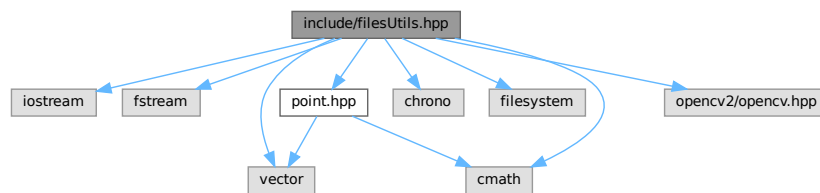
```
00001
00006 #ifndef CONFIG_READER_HPP
00007 #define CONFIG_READER_HPP
00008
00009 #include <string>
00010 #include <regex>
00011 #include <unordered_set>
00012 #include <filesystem>
00013
00014 namespace km
00015 {
00021     class ConfigReader
00022     {
00023     private:
00024         double first_level_compression_color = 0.;
00025         double second_level_compression_color = 0.;
00026         double third_level_compression_color = 0.;
00027         double fourth_level_compression_color = 0.;
00028         double fifth_level_compression_color = 0.;
00029         double resizing_factor = 0.;
00030         int color_choice = 0;
00031         int compression_choice = 0;
00032         std::filesystem::path inputImagePath;
00033         std::regex pattern;
00034         std::unordered_set<std::string> requiredVariables = {};
00035
00036         [[nodiscard]] auto checkVariableExists(const std::string &variableName) const -> bool;
00037
00043     public:
00044         [[nodiscard]] auto getFirstLevelCompressionColor() const -> double;
00045
00050         [[nodiscard]] auto getSecondLevelCompressionColor() const -> double;
00051
00056         [[nodiscard]] auto getThirdLevelCompressionColor() const -> double;
00057
00062         [[nodiscard]] auto getFourthLevelCompressionColor() const -> double;
00063
00068         [[nodiscard]] auto getFifthLevelCompressionColor() const -> double;
00069
00074         [[nodiscard]] auto getColorChoice() const -> int;
00075
00080         [[nodiscard]] auto getCompressionChoice() const -> int;
00081
00086         [[nodiscard]] auto getInputImagePath() const -> std::filesystem::path;
00087
00092         [[nodiscard]] auto getResizingFactor() const -> double;
00093
00098         [[nodiscard]] auto readConfigFile() -> bool;
00099
00100         ConfigReader();
00101     };
00102 } // namespace km
00103
00104 #endif // CONFIG_READER_HPP
```

8.3 include/filesUtils.hpp File Reference

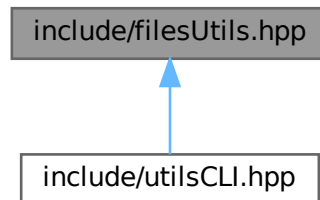
Utility functions for file handling.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <chrono>
#include <filesystem>
#include <point.hpp>
#include <opencv2/opencv.hpp>
```

Include dependency graph for filesUtils.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `km::filesUtils`
Provides utility functions for file handling.
- namespace `km`
Main namespace for the project.

Functions

- auto [km::filesUtils::createOutputDirectories](#) () -> void
Creates output directories.
- auto [km::filesUtils::writeBinaryFile](#) (std::string &outputPath, int &width, int &height, int &k, std::vector< [Point](#) > points, std::vector< [Point](#) > centroids) -> void
Writes data to a binary file.
- auto [km::filesUtils::isCorrectExtension](#) (const std::filesystem::path &filePath, const std::string &correctExtension) -> bool
Checks if a file has the correct extension.
- auto [km::filesUtils::createDecodingMenu](#) (std::filesystem::path &decodeDir, std::vector< std::filesystem::path > &imageNames) -> void
Creates a decoding menu.
- auto [km::filesUtils::readBinaryFile](#) (std::string &path, cv::Mat &imageCompressed) -> int
Reads a binary file and reconstructs the compressed image.

8.3.1 Detailed Description

Utility functions for file handling.

8.4 filesUtils.hpp

[Go to the documentation of this file.](#)

```

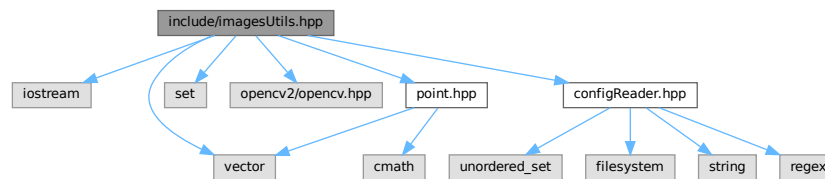
00001
00006 #ifndef FILESUTILS_HPP
00007 #define FILESUTILS_HPP
00008
00009 #include <iostream>
00010 #include <fstream>
00011 #include <vector>
00012 #include <cmath>
00013 #include <chrono>
00014 #include <filesystem>
00015 #include <point.hpp>
00016 #include <opencv2/opencv.hpp>
00017
00023 namespace km
00024 {
00025     namespace filesUtils
00026     {
00030         auto createOutputDirectories() -> void;
00031
00041         auto writeBinaryFile(std::string &outputPath, int &width, int &height, int &k,
std::vector<Point> points, std::vector<Point> centroids) -> void;
00042
00049         auto isCorrectExtension(const std::filesystem::path &filePath, const std::string
&correctExtension) -> bool;
00050
00056         auto createDecodingMenu(std::filesystem::path &decodeDir, std::vector<std::filesystem::path>
&imageNames) -> void;
00057
00064         auto readBinaryFile(std::string &path, cv::Mat &imageCompressed) -> int;
00065     };
00066 }
00067
00068 #endif // FILESUTILS_HPP

```

8.5 include/imagesUtils.hpp File Reference

Utility functions for image processing.

```
#include <iostream>
#include <vector>
#include <set>
#include <opencv2/opencv.hpp>
#include <configReader.hpp>
#include <point.hpp>
Include dependency graph for imagesUtils.hpp:
```



Namespaces

- namespace [km::imageUtils](#)
Provides utility functions for image processing.
- namespace [km](#)
Main namespace for the project.

Functions

- void [km::imageUtils::preprocessing](#) (cv::Mat &image, int &typeCompressionChoice)
Performs preprocessing on an image.
- void [km::imageUtils::defineKValue](#) (int &k, int levelsColorsChoice, std::set< std::vector< unsigned char > > &different_colors)
Defines the value of K based on the color levels choice.
- void [km::imageUtils::pointsFromImage](#) (cv::Mat &image, std::vector< [Point](#) > &points, std::set< std::vector< unsigned char > > &different_colors)
Extracts points from an image.

8.5.1 Detailed Description

Utility functions for image processing.

8.6 imagesUtils.hpp

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef IMAGEUTILS_HPP
00007 #define IMAGEUTILS_HPP
00008
00009 #include <iostream>
00010 #include <vector>
00011 #include <set>
00012 #include <opencv2/opencv.hpp>
00013 #include <configReader.hpp>
00014 #include <point.hpp>
00015
00021 namespace km
00022 {
00023     namespace imageUtils
00024     {
00030         void preprocessing(cv::Mat& image, int& typeCompressionChoice);
00031
00038         void defineKValue(int& k, int levelsColorsChoice, std::set<std::vector<unsigned char>&
different_colors);
00039
00046         void pointsFromImage(cv::Mat& image, std::vector<Point>& points,
std::set<std::vector<unsigned char>& different_colors);
00047     };
00048
00049 }
00050
00051 #endif // IMAGEUTILS_HPP
```

8.7 include/kmDocs.hpp File Reference

Documentation for the `km` namespace.

Namespaces

- namespace `km`
Main namespace for the project.

8.7.1 Detailed Description

Documentation for the `km` namespace.

This file provides comprehensive documentation for the `km` namespace, which includes utilities for clustering algorithms, file handling, image processing, etc. The `km` namespace serves as the main container for core functionalities and tools used throughout the project.

8.8 kmDocs.hpp

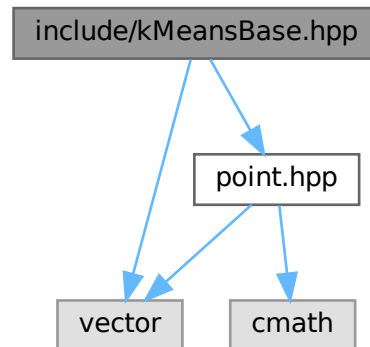
[Go to the documentation of this file.](#)

```
00001
00017 namespace km {
00018     // This file is for Doxygen documentation purposes only.
00019 }
```

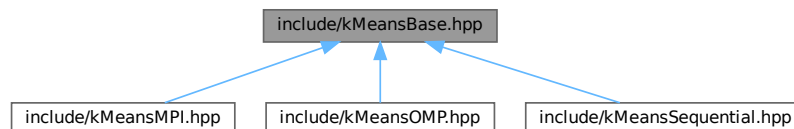

8.9 include/kMeansBase.hpp File Reference

Base class for K-means clustering algorithm.

```
#include <vector>
#include "point.hpp"
Include dependency graph for kMeansBase.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [km::KMeansBase](#)
Base class for K-means clustering algorithm.

Namespaces

- namespace [km](#)
Main namespace for the project.

8.9.1 Detailed Description

Base class for K-means clustering algorithm.

8.10 kMeansBase.hpp

[Go to the documentation of this file.](#)

```

00001
00006 #ifndef KMEANS_BASE_HPP
00007 #define KMEANS_BASE_HPP
00008
00009 #include <vector>
00010 #include "point.hpp"
00011
00012 namespace km
00013 {
00019     class KMeansBase
00020     {
00021     public:
00022         KMeansBase(const int &k, const std::vector<Point> &points);
00028
00029         KMeansBase(const int &k);
00035
00036         virtual ~KMeansBase() = default;
00040
00041         virtual void run() = 0;
00045
00046         [[nodiscard]] auto getPoints() const -> std::vector<Point>;
00051
00052         [[nodiscard]] auto getCentroids() const -> std::vector<Point>;
00057
00058         [[nodiscard]] auto getIterations() const -> int;
00063
00064     protected:
00065         int k;
00066         std::vector<Point> points;
00067         std::vector<Point> centroids;
00068         int number_of_iterations;
00069     };
00070 }
00071 // namespace k
00072
00073 #endif // KMEANS_BASE_HPP

```

8.11 include/kMeansCUDA.cuh File Reference

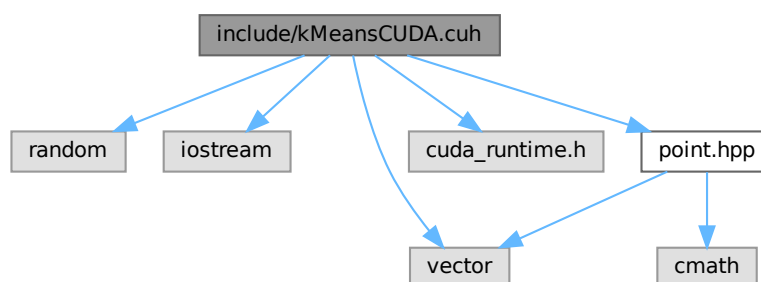
Implementation of the K-means clustering algorithm using CUDA.

```

#include <random>
#include <iostream>
#include <vector>
#include <cuda_runtime.h>
#include <point.hpp>

```

Include dependency graph for kMeansCUDA.cuh:



Classes

- class [km::KMeansCUDA](#)
Represents the K-means clustering algorithm using CUDA.

Namespaces

- namespace [km](#)
Main namespace for the project.

Macros

- #define [KMEANS_CUDA_HPP](#)

8.11.1 Detailed Description

Implementation of the K-means clustering algorithm using CUDA.

8.11.2 Macro Definition Documentation

8.11.2.1 KMEANS_CUDA_HPP

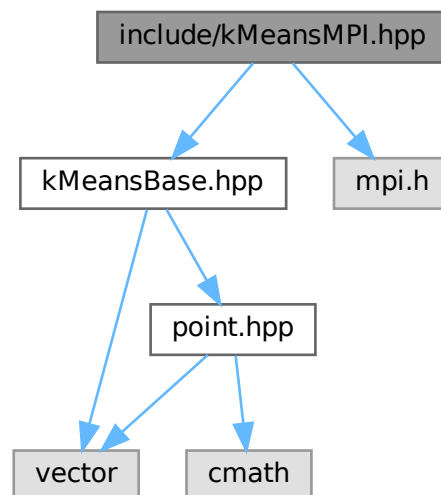
```
#define KMEANS_CUDA_HPP
```

8.12 include/kMeansMPI.hpp File Reference

Implementation of the K-means clustering algorithm using MPI.

```
#include "kMeansBase.hpp"  
#include <mpi.h>
```

Include dependency graph for kMeansMPI.hpp:



Classes

- class [km::KMeansMPI](#)

Represents the K-means clustering algorithm using MPI.

Namespaces

- namespace [km](#)

Main namespace for the project.

8.12.1 Detailed Description

Implementation of the K-means clustering algorithm using MPI.

8.13 kMeansMPI.hpp

[Go to the documentation of this file.](#)

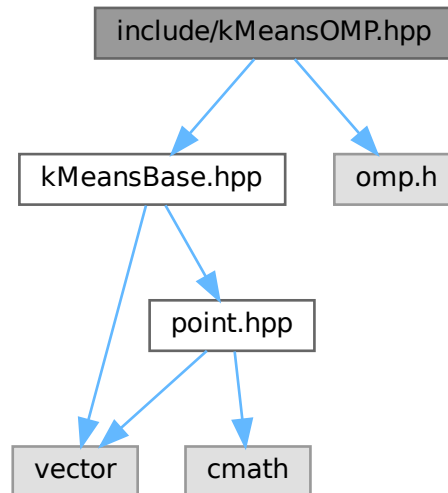
```
00001
00006 #ifndef KMEANS_MPI_HPP
00007 #define KMEANS_MPI_HPP
00008
00009 #include "kMeansBase.hpp"
00010 #include <mpi.h>
00011
00012 namespace km
00013 {
00019     class KMeansMPI : public KMeansBase
00020     {
00021     public:
00027         KMeansMPI(const int &k, const std::vector<Point> &points, std::vector<std::pair<int, Point>
local_points);
00028
00033         KMeansMPI(const int &k, std::vector<std::pair<int, Point> local_points);
00034
00038         void run() override;
00039
00040     private:
00041         std::vector<std::pair<int, Point> local_points;
00042     };
00043 } // namespace k
00044
00045 #endif // KMEANS_MPI_HPP
```

8.14 include/kMeansOMP.hpp File Reference

Implementation of the K-means clustering algorithm using OpenMP.

```
#include "kMeansBase.hpp"
#include <omp.h>
```

Include dependency graph for kMeansOMP.hpp:



Classes

- class [km::KMeansOMP](#)
Represents the K-means clustering algorithm using OpenMP.

Namespaces

- namespace [km](#)
Main namespace for the project.

8.14.1 Detailed Description

Implementation of the K-means clustering algorithm using OpenMP.

8.15 kMeansOMP.hpp

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef KMEANS_OMP_HPP
00007 #define KMEANS_OMP_HPP
00008
00009 #include "kMeansBase.hpp"
00010 #include <omp.h>
00011
00012 namespace km
00013 {
```

```

00019     class KMeansOMP : public KMeansBase
00020     {
00021     public:
00027         KMeansOMP(const int &k, const std::vector<Point> &points);
00028
00032         void run() override;
00033     };
00034 } // namespace km
00035
00036 #endif // KMEANS_OMP_HPP

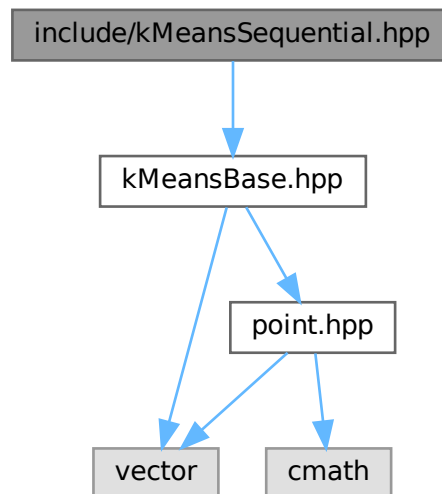
```

8.16 include/kMeansSequential.hpp File Reference

Implementation of the K-means clustering algorithm.

```
#include "kMeansBase.hpp"
```

Include dependency graph for kMeansSequential.hpp:



Classes

- class `km::KMeansSequential`
Represents the K-means clustering algorithm.

Namespaces

- namespace `km`
Main namespace for the project.

8.16.1 Detailed Description

Implementation of the K-means clustering algorithm.

8.17 kMeansSequential.hpp

[Go to the documentation of this file.](#)

```

00001
00006 #ifndef KMEANS_SEQUENTIAL_HPP
00007 #define KMEANS_SEQUENTIAL_HPP
00008
00009 #include "kMeansBase.hpp"
00010
00011 namespace km
00012 {
00018     class KMeansSequential : public KMeansBase
00019     {
00020     public:
00026         KMeansSequential(const int &k, const std::vector<Point> &points);
00027
00031         void run() override;
00032     };
00033 }
00034
00035 #endif // KMEANS_SEQUENTIAL_HPP

```

8.18 include/performanceEvaluation.hpp File Reference

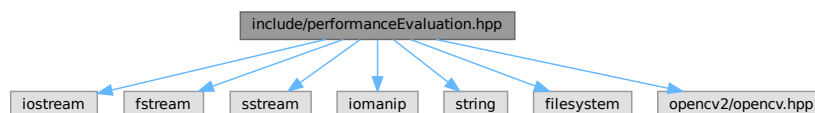
Performance evaluation class.

```

#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <string>
#include <filesystem>
#include <opencv2/opencv.hpp>

```

Include dependency graph for performanceEvaluation.hpp:



Classes

- class [km::Performance](#)
Represents the performance evaluation.

Namespaces

- namespace [km](#)
Main namespace for the project.

8.18.1 Detailed Description

Performance evaluation class.

8.19 performanceEvaluation.hpp

[Go to the documentation of this file.](#)

```

00001
00006 #ifndef PERFORMANCE_HPP
00007 #define PERFORMANCE_HPP
00008
00009 #include <iostream>
00010 #include <fstream>
00011 #include <sstream>
00012 #include <iomanip>
00013 #include <string>
00014 #include <filesystem>
00015 #include <opencv2/opencv.hpp>
00016
00017 namespace km
00018 {
00024     class Performance
00025     {
00026     public:
00030         Performance();
00031
00040         auto writeCSV(int different_colors_size, int k, int n_points, double elapsedKmeans, int
number_of_iterations, int num_processes = 0) -> void;
00041
00047         static auto extractFileName(const std::string &outputPath) -> std::string;
00048
00055         auto fillPerformance(int choice, const std::string &img, const std::string &method) -> void;
00056
00057     private:
00062         auto createOrOpenCSV(const std::string &filename) -> void;
00063
00074         auto appendToCSV(const std::string &filename, int n_diff_colors, int k, int n_colors, const
std::string &compType, double time, int num_processes, int number_of_iterations) -> void;
00075
00076         std::string img;
00077         int choice{};
00078         std::string method;
00079     };
00080 }
00081
00082 #endif // PERFORMANCE_HPP

```

8.20 include/point.hpp File Reference

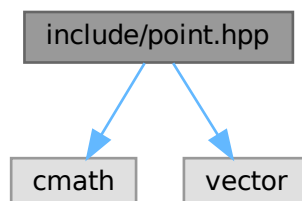
Point class representing a point in a feature space.

```

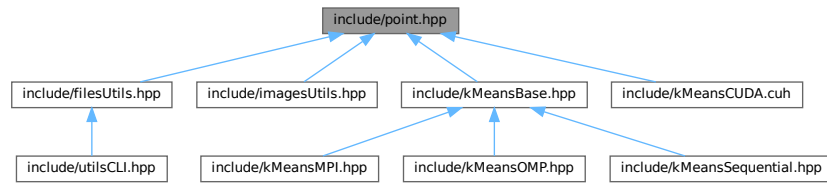
#include <cmath>
#include <vector>

```

Include dependency graph for point.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `km::Point`
Represents a point in a feature space.

Namespaces

- namespace `km`
Main namespace for the project.

8.20.1 Detailed Description

Point class representing a point in a feature space.

8.21 point.hpp

[Go to the documentation of this file.](#)

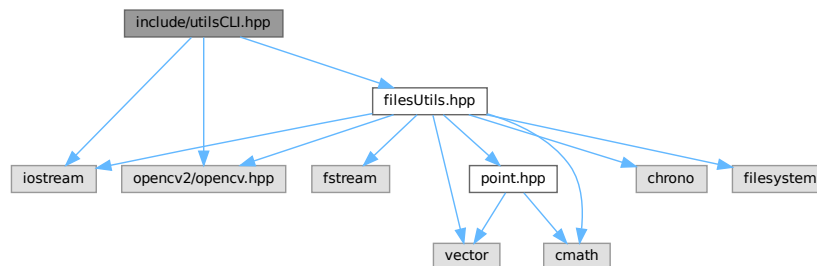
```

00001
00006 #ifndef POINT_HPP
00007 #define POINT_HPP
00008
00009 #include <cmath>
00010 #include <vector>
00011
00012 namespace km
00013 {
00019     class Point
00020     {
00021     public:
00022         int id{0};
00023         unsigned char r{0};
00024         unsigned char g{0};
00025         unsigned char b{0};
00026         int clusterId{-1};
00027
00032         Point();
00033
00039         Point(const int &id, const std::vector<int> &coordinates);
00040
00046         [[nodiscard]] auto distance(const Point &p) const -> double;
00047
00053         auto getFeature(int index) -> unsigned char &;
00054
00060         [[nodiscard]] auto getFeature_int(int index) const -> int;
00061
00067         auto setFeature(int index, int x) -> void;
00068     };
00069 } // namespace km
00070
00071 #endif // POINT_HPP
  
```

8.22 include/UtilsCLI.hpp File Reference

Utility functions for the command-line interface.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <filesUtils.hpp>
Include dependency graph for UtilsCLI.hpp:
```



Namespaces

- namespace [km::UtilsCLI](#)
Provides utility functions for the command-line interface.
- namespace [km](#)
Main namespace for the project.

Functions

- void [km::UtilsCLI::sequentialEncoderHeader](#) ()
Displays the header for the sequential encoder.
- void [km::UtilsCLI::mpiEncoderHeader](#) ()
Displays the header for the MPI encoder.
- void [km::UtilsCLI::ompEncoderHeader](#) ()
Displays the header for the OpenMP encoder.
- void [km::UtilsCLI::mainMenuHeader](#) ()
Displays the main menu header.
- void [km::UtilsCLI::decoderHeader](#) ()
Displays the decoder header.
- void [km::UtilsCLI::workDone](#) ()
Displays the work done message.
- void [km::UtilsCLI::compressionChoices](#) (int &levelsColorsChoice, int &typeCompressionChoice, std::string &outputPath, cv::Mat &image, int executionStandard)
Handles the compression choices.
- void [km::UtilsCLI::printCompressionInformations](#) (int &originalWidth, int &originalHeight, int &width, int &height, int &k, size_t &different_colors_size)
Prints the compression information.
- void [km::UtilsCLI::displayDecodingMenu](#) (std::string &path, std::vector< std::filesystem::path > &imageNames, std::filesystem::path &decodeDir)
Displays the decoding menu.

8.22.1 Detailed Description

Utility functions for the command-line interface.

8.23 utilsCLI.hpp

[Go to the documentation of this file.](#)

```

00001
00006 #ifndef UTILSCLI_HPP
00007 #define UTILSCLI_HPP
00008
00009 #include <iostream>
00010 #include <opencv2/opencv.hpp>
00011 #include <filesUtils.hpp>
00012
00018 namespace km
00019 {
00020     namespace utilsCLI
00021     {
00022
00026         void sequentialEncoderHeader();
00027
00031         void mpiEncoderHeader();
00032
00036         void ompEncoderHeader();
00040         void mainMenuHeader();
00041
00045         void decoderHeader();
00046
00050         void workDone();
00051
00060         void compressionChoices(int &levelsColorsChoice, int &typeCompressionChoice, std::string
&outputPath, cv::Mat &image, int executionStandard);
00061
00071         void printCompressionInformations(int &originalWidth, int &originalHeight, int &width, int
&height, int &k, size_t &different_colors_size);
00072
00079         void displayDecodingMenu(std::string &path, std::vector<std::filesystem::path> &imageNames,
std::filesystem::path &decodeDir);
00080     };
00081 }
00082
00083 #endif // UTILSCLI_HPP

```

8.24 README.md File Reference

Index

- ~KMeansBase
 - km::KMeansBase, [32](#)
- appendToCSV
 - km::Performance, [51](#)
- b
 - km::Point, [56](#)
- centroids
 - km::KMeansBase, [33](#)
 - km::KMeansCUDA, [37](#)
- checkVariableExists
 - km::ConfigReader, [26](#)
- choice
 - km::Performance, [52](#)
- clusterId
 - km::Point, [56](#)
- color_choice
 - km::ConfigReader, [28](#)
- compression_choice
 - km::ConfigReader, [28](#)
- compressionChoices
 - km::utilsCLI, [19](#)
- ConfigReader
 - km::ConfigReader, [26](#)
- createDecodingMenu
 - km::filesUtils, [14](#)
- createOrOpenCSV
 - km::Performance, [51](#)
- createOutputDirectories
 - km::filesUtils, [14](#)
- decoderHeader
 - km::utilsCLI, [19](#)
- defineKValue
 - km::imageUtils, [17](#)
- displayDecodingMenu
 - km::utilsCLI, [19](#)
- distance
 - km::Point, [55](#)
- extractFileName
 - km::Performance, [51](#)
- fifth_level_compression_color
 - km::ConfigReader, [28](#)
- fillPerformance
 - km::Performance, [52](#)
- first_level_compression_color
 - km::ConfigReader, [28](#)
- fourth_level_compression_color
 - km::ConfigReader, [28](#)
- g
 - km::Point, [56](#)
- getCentroids
 - km::KMeansBase, [33](#)
 - km::KMeansCUDA, [36](#)
- getColorChoice
 - km::ConfigReader, [26](#)
- getCompressionChoice
 - km::ConfigReader, [26](#)
- getFeature
 - km::Point, [55](#)
- getFeature_int
 - km::Point, [55](#)
- getFifthLevelCompressionColor
 - km::ConfigReader, [26](#)
- getFirstLevelCompressionColor
 - km::ConfigReader, [26](#)
- getFourthLevelCompressionColor
 - km::ConfigReader, [27](#)
- getInputImageFilePath
 - km::ConfigReader, [27](#)
- getIterations
 - km::KMeansBase, [33](#)
 - km::KMeansCUDA, [36](#)
- getPoints
 - km::KMeansBase, [33](#)
 - km::KMeansCUDA, [36](#)
- getResizingFactor
 - km::ConfigReader, [27](#)
- getSecondLevelCompressionColor
 - km::ConfigReader, [27](#)
- getThirdLevelCompressionColor
 - km::ConfigReader, [27](#)
- id
 - km::Point, [56](#)
- img
 - km::Performance, [52](#)
- include/configReader.hpp, [59](#), [60](#)
- include/filesUtils.hpp, [61](#), [62](#)
- include/imagesUtils.hpp, [63](#), [64](#)
- include/kmDocs.hpp, [64](#)
- include/kMeansBase.hpp, [65](#), [66](#)
- include/kMeansCUDA.cuh, [66](#)
- include/kMeansMPI.hpp, [67](#), [68](#)
- include/kMeansOMP.hpp, [68](#), [69](#)
- include/kMeansSequential.hpp, [70](#), [71](#)

- include/performanceEvaluation.hpp, 71, 72
- include/point.hpp, 72, 73
- include/UtilsCLI.hpp, 74, 75
- inputImagePath
 - km::ConfigReader, 29
- isCorrectExtension
 - km::filesUtils, 15
- k
 - km::KMeansBase, 33
 - km::KMeansCUDA, 37
- km, 13
- km::ConfigReader, 23
 - checkVariableExists, 26
 - color_choice, 28
 - compression_choice, 28
 - ConfigReader, 26
 - fifth_level_compression_color, 28
 - first_level_compression_color, 28
 - fourth_level_compression_color, 28
 - getColorChoice, 26
 - getCompressionChoice, 26
 - getFifthLevelCompressionColor, 26
 - getFirstLevelCompressionColor, 26
 - getFourthLevelCompressionColor, 27
 - getInputImagePath, 27
 - getResizingFactor, 27
 - getSecondLevelCompressionColor, 27
 - getThirdLevelCompressionColor, 27
 - inputImagePath, 29
 - pattern, 29
 - readConfigFile, 28
 - requiredVariables, 29
 - resizing_factor, 29
 - second_level_compression_color, 29
 - third_level_compression_color, 29
- km::filesUtils, 14
 - createDecodingMenu, 14
 - createOutputDirectories, 14
 - isCorrectExtension, 15
 - readBinaryFile, 15
 - writeBinaryFile, 15
- km::imageUtils, 17
 - defineKValue, 17
 - pointsFromImage, 17
 - preprocessing, 18
- km::KMeansBase, 30
 - ~KMeansBase, 32
 - centroids, 33
 - getCentroids, 33
 - getIterations, 33
 - getPoints, 33
 - k, 33
 - KMeansBase, 32
 - number_of_iterations, 34
 - points, 34
 - run, 33
- km::KMeansCUDA, 34
 - centroids, 37
 - getCentroids, 36
 - getIterations, 36
 - getPoints, 36
 - k, 37
 - KMeansCUDA, 35
 - number_of_iterations, 37
 - plotClusters, 36
 - points, 37
 - printClusters, 36
 - run, 36
- km::KMeansMPI, 38
 - KMeansMPI, 40, 41
 - local_points, 41
 - run, 41
- km::KMeansOMP, 42
 - KMeansOMP, 44
 - run, 45
- km::KMeansSequential, 45
 - KMeansSequential, 48
 - run, 49
- km::Performance, 49
 - appendToCSV, 51
 - choice, 52
 - createOrOpenCSV, 51
 - extractFileName, 51
 - fillPerformance, 52
 - img, 52
 - method, 53
 - Performance, 50
 - writeCSV, 52
- km::Point, 53
 - b, 56
 - clusterId, 56
 - distance, 55
 - g, 56
 - getFeature, 55
 - getFeature_int, 55
 - id, 56
 - Point, 54
 - r, 56
 - setFeature, 56
- km::utilsCLI, 18
 - compressionChoices, 19
 - decoderHeader, 19
 - displayDecodingMenu, 19
 - mainMenuHeader, 20
 - mpiEncoderHeader, 20
 - ompEncoderHeader, 20
 - printCompressionInformations, 20
 - sequentialEncoderHeader, 20
 - workDone, 21
- KMEANS_CUDA_HPP
 - kMeansCUDA.cuh, 67
- KMeansBase
 - km::KMeansBase, 32
- KMeansCUDA
 - km::KMeansCUDA, 35
- kMeansCUDA.cuh

- KMEANS_CUDA_HPP, [67](#)
- KMeansMPI
 - km::KMeansMPI, [40](#), [41](#)
- KMeansOMP
 - km::KMeansOMP, [44](#)
- KMeansSequential
 - km::KMeansSequential, [48](#)
- local_points
 - km::KMeansMPI, [41](#)
- mainMenuHeader
 - km::utilsCLI, [20](#)
- method
 - km::Performance, [53](#)
- mpiEncoderHeader
 - km::utilsCLI, [20](#)
- number_of_iterations
 - km::KMeansBase, [34](#)
 - km::KMeansCUDA, [37](#)
- ompEncoderHeader
 - km::utilsCLI, [20](#)
- Parallel Kmeans Images Compressor, [1](#)
- pattern
 - km::ConfigReader, [29](#)
- Performance
 - km::Performance, [50](#)
- plotClusters
 - km::KMeansCUDA, [36](#)
- Point
 - km::Point, [54](#)
- points
 - km::KMeansBase, [34](#)
 - km::KMeansCUDA, [37](#)
- pointsFromImage
 - km::imageUtils, [17](#)
- preprocessing
 - km::imageUtils, [18](#)
- printClusters
 - km::KMeansCUDA, [36](#)
- printCompressionInformations
 - km::utilsCLI, [20](#)
- r
 - km::Point, [56](#)
- readBinaryFile
 - km::filesUtils, [15](#)
- readConfigFile
 - km::ConfigReader, [28](#)
- README.md, [75](#)
- requiredVariables
 - km::ConfigReader, [29](#)
- resizing_factor
 - km::ConfigReader, [29](#)
- run
 - km::KMeansBase, [33](#)
 - km::KMeansCUDA, [36](#)
 - km::KMeansMPI, [41](#)
 - km::KMeansOMP, [45](#)
 - km::KMeansSequential, [49](#)
- second_level_compression_color
 - km::ConfigReader, [29](#)
- sequentialEncoderHeader
 - km::utilsCLI, [20](#)
- setFeature
 - km::Point, [56](#)
- third_level_compression_color
 - km::ConfigReader, [29](#)
- workDone
 - km::utilsCLI, [21](#)
- writeBinaryFile
 - km::filesUtils, [15](#)
- writeCSV
 - km::Performance, [52](#)