Geli Vision

Generated by Doxygen 1.9.6

Chapter 1

Geli Vision

An image processing software.

Chapter 2

2.1 Vue

JavaScript Web

https://vuejs.org

Vue Mastery: https://www.vuemastery.com

Vuetify: https://vuetifyjs.com

Vue.js Developers: https://vuejsdevelopers.com

2.2 Element-Plus

Vue 3 UI

https://element-plus.org

2.3 LogicFlow

LogicFlow

https://docs.logic-flow.cn/docs/#/zh/guide/start

2.4 Spring Boot

Spring

https://spring.io/projects/spring-boot

Baeldung: https://www.baeldung.com/spring-boot

Spring Boot Tutorial: https://www.javatpoint.com/spring-boot-tutorial

Spring Framework Guru: https://springframework.guru

2.5 OpenCV

https://opencv.org

OpenCV C++ Tutorials: https://docs.opencv.org/master/d9/df8/tutorial_root.html

OpenCV-Python Tutorials: https://opencv-python-tutroals.readthedocs.io/en/latest/py↩
_tutorials/py_tutorials.html

Learn OpenCV: https://www.learnopencv.com

OpenCV Cookbook: https://www.packtpub.com/product/opencv-cookbook/9781789344912

2.6 Socket.IO

https://socket.io/docs/v4

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

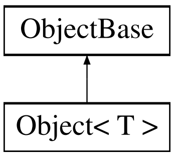
Class Documentation

6.1 Object< T > Struct Template Reference

A template class for wrapping objects of type T.

```
#include <object.h>
```

Inheritance diagram for Object< T >:



Public Member Functions

- Object (T ∗ptr)

  Constructs an Object from a pointer to an object of type T.
- ∼**Object** ()

  The destructor.
- T inner ()

  Gets the inner object.
- T & inner_ref ()

  Gets a reference to the inner object.
- const T & inner_const_ref ()

  Gets a const reference to the inner object.

6.1.1 Detailed Description

template<typename T>
struct Object< T >

A template class for wrapping objects of type T.

Template Parameters

T	The type of object to wrap.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Object()

```
template<typename T >
Object< T >::Object (
             T * ptr )
```

Constructs an Object from a pointer to an object of type T.

Parameters

ptr	A pointer to the object to wrap.

6.1.3 Member Function Documentation

6.1.3.1 inner()

```
template<typename T >
T Object< T >::inner
```

Gets the inner object.

Returns

The inner object.

6.1.3.2 inner_const_ref()

```
template<typename T >
const T & Object< T >::inner_const_ref
```

Gets a const reference to the inner object.

Returns

A const reference to the inner object.

6.1.3.3 inner_ref()

```
template<typename T >
T & Object< T >::inner_ref
```

Gets a reference to the inner object.

Returns

A reference to the inner object.

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

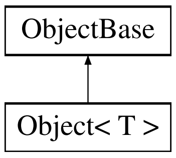
- object.h

6.2 ObjectBase Struct Reference

base class for all objects

```
#include <object.h>
```

Inheritance diagram for ObjectBase:



6.2.1 Detailed Description

base class for all objects

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

- object.h

Chapter 7

File Documentation

7.1 general.cpp File Reference

Contains functions for image processing.

```
#include "object.h"
```

Macros

- #define **__EXPORT** extern "C" __declspec(dllexport)

Functions

- __EXPORT void read (ParamPtrArray &params)

  Reads an image from file.
- __EXPORT void show (ParamPtrArray &params)

  Shows an image.
- __EXPORT void morph_open (ParamPtrArray &params)

  Applies morphological opening to an image.
- __EXPORT void morph_close (ParamPtrArray &params)

  Applies morphological closing to an image.
- __EXPORT void threshold (ParamPtrArray &params)

  Applies morphological closing to an image.
- __EXPORT void convert_color (ParamPtrArray &params)

  Convert an input image to a different color space.
- __EXPORT void median_blur (ParamPtrArray &params)

  Apply median blur to an input image.
- __EXPORT void find_contours (ParamPtrArray &params)

  Find contours in a binary image.
- __EXPORT void draw_contours (ParamPtrArray &params)

  Draw contours in a binary image.

7.1.1 Detailed Description

Contains functions for image processing.

7.1.2 Function Documentation

7.1.2.1 convert_color()

```
__EXPORT void convert_color (
            ParamPtrArray & params )
```

Convert an input image to a different color space.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the image. 2. ***input*** A `int` representing the conversion type. 3. ***output*** A `cv::Mat` object representing the processed image.

7.1.2.2 draw_contours()

```
__EXPORT void draw_contours (
            ParamPtrArray & params )
```

Draw contours in a binary image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the input binary image. 2. ***input*** A `std::vector<std::vector<cv::Point>>` object representing the contours. 3. ***output*** A `cv::Mat` object representing the image with drawn countours.

7.1.2.3 find_contours()

```
__EXPORT void find_contours (
            ParamPtrArray & params )
```

Find contours in a binary image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the input binary image. 2. ***output*** A `std::vector<std::vector<cv::Point>>` object representing the found contours.

7.1.2.4 median_blur()

```
__EXPORT void median_blur (
            ParamPtrArray & params )
```

Apply median blur to an input image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the image. 2. ***input*** A `int` representing the kernel size. 3. ***output*** A `cv::Mat` object representing the processed image.

7.1.2.5 morph_close()

```
__EXPORT void morph_close (
            ParamPtrArray & params )
```

Applies morphological closing to an image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the image. 2. ***input*** A `cv::Size` object representing the kernel size for the operation. 3. ***output*** A `cv::Mat` object representing the processed image.

7.1.2.6 morph_open()

```
__EXPORT void morph_open (
            ParamPtrArray & params )
```

Applies morphological opening to an image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the image. 2. ***input*** A `cv::Size` object representing the kernel size for the operation. 3. ***output*** A `cv::Mat` object representing the processed image.

7.1.2.7 read()

```
__EXPORT void read (
            ParamPtrArray & params )
```

Reads an image from file.

Parameters

params	An array of parameters containing: 1. ***input*** A `string` representing the file path of the image. 2. ***input*** An `integer` representing the desired color type of the image. 3. ***output*** A `cv::Mat` object representing the read image.

7.1.2.8 show()

```
__EXPORT void show (
            ParamPtrArray & params )
```

Shows an image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the image. 2. ***output*** A `cv::Mat` object representing the image.

7.1.2.9 threshold()

```
__EXPORT void threshold (
             ParamPtrArray & params )
```

Applies morphological closing to an image.

Parameters

params	An array of parameters containing: 1. ***input*** A `cv::Mat` object representing the image. 2. ***input*** A `double` representing the threshold. 3. ***input*** A `double` representing the max value. 4. ***input*** A `int` representing the threshold type. 5. ***output*** A `cv::Mat` object representing the processed image.

7.2 object.h File Reference

Contains container for various types of data.

```
#include <string>
#include <vector>
#include <opencv2/opencv.hpp>
```

Classes

- struct ObjectBase

  base class for all objects
- struct Object< T >

  A template class for wrapping objects of type T.

Typedefs

- using **ParamPtr** = std::shared_ptr< ObjectBase >
- using **ParamPtrArray** = std::vector< ParamPtr >
- using **MatObject** = Object< cv::Mat >
- using **SizeObject** = Object< cv::Size >
- using **IntObject** = Object< int >
- using **DoubleObject** = Object< double >
- using **StringObject** = Object< std::string >

Functions

- template<typename T >
  T get_inner (ParamPtr param_ptr)

  Gets a copy of the inner object.
- template<typename T >
  T & get_inner_ref (ParamPtr param_ptr)

  Gets a reference of the inner object.
- template<typename T >
  const T & get_inner_const_ref (ParamPtr param_ptr)

  Gets a const reference of the inner object.
- template<typename T >
  ParamPtr make_param (T ∗value_ptr)

  Creates a ParamPtr of an inner object.

7.2.1 Detailed Description

Contains container for various types of data.

7.2.2 Function Documentation

7.2.2.1 get_inner()

```
template<typename T >
T get_inner (
             ParamPtr param_ptr )
```

Gets a copy of the inner object.

Parameters

param_ptr	pointer to the object.

Returns

A copy of the inner object.

7.2.2.2 get_inner_const_ref()

```
template<typename T >
const T & get_inner_const_ref (
             ParamPtr param_ptr )
```

Gets a const reference of the inner object.

Parameters

param_ptr	pointer to the object.

Returns

A const reference of the inner object.

7.2.2.3 get_inner_ref()

```
template<typename T >
T & get_inner_ref (
             ParamPtr param_ptr )
```

Gets a reference of the inner object.

Parameters

param_ptr	pointer to the object.

Returns

A reference of the inner object.

7.2.2.4 make_param()

```
template<typename T >
ParamPtr make_param (
             T * value_ptr )
```

Creates a ParamPtr of an inner object.

Parameters

value_ptr	a pointer to the inner object.

Returns

A ParamPtr of the object.

7.3 object.h

Go to the documentation of this file.

```
#pragma once
#include<string>
#include<vector>
#include<opencv2/opencv.hpp>

struct ObjectBase {
public:
    virtual ~ObjectBase() {}
};

template<typename T>
struct Object : public ObjectBase {
public:
    Object(T* ptr);
    ~Object();

    T         inner();
    T&        inner_ref();
    const T& inner_const_ref();
private:
    T* ptr;
};

template<typename T>
Object<T>::Object(T* ptr) : ptr(ptr) {}

template<typename T>
Object<T>::~Object() {
    delete ptr;
}

template<typename T>
T Object<T>::inner() {
    return *ptr;
}

template<typename T>
const T& Object<T>::inner_const_ref() {
    return *ptr;
}

template<typename T>
T& Object<T>::inner_ref() {
    return *ptr;
}

using ParamPtr = std::shared_ptr<ObjectBase>;
using ParamPtrArray = std::vector<ParamPtr>;

using MatObject = Object<cv::Mat>;
using SizeObject = Object<cv::Size>;
using IntObject = Object<int>;
using DoubleObject = Object<double>;
using StringObject = Object<std::string>;

template<typename T>
T get_inner(ParamPtr param_ptr) {
    return dynamic_cast<Object<T>*>(param_ptr.get())->inner();
}
template<typename T>
T& get_inner_ref(ParamPtr param_ptr) {
    return dynamic_cast<Object<T>*>(param_ptr.get())->inner_ref();
}
template<typename T>
const T& get_inner_const_ref(ParamPtr param_ptr) {
    return dynamic_cast<Object<T>*>(param_ptr.get())->inner_const_ref();
}
template<typename T>
ParamPtr make_param(T* value_ptr) {
    return ParamPtr(new Object<T>(value_ptr));
}
```

Index