Camera Tampering Detection

Generated by Doxygen 1.8.11

Contents

1 Namespace Index **1**

1.1 Namespace List 1

2 Class Index **3**

2.1 Class List 3

3 File Index **5**

3.1 File List 5

4 Namespace Documentation **7**

4.1 cva Namespace Reference 7

4.2 cva::ctd Namespace Reference 7

4.2.1 Enumeration Type Documentation 8

4.2.1.1 DetectorParameter 8

4.2.1.2 TamperingType 8

4.2.2 Function Documentation 8

4.2.2.1 version() 8

5 Class Documentation **9**

5.1 cva::ctd::CameraTamperingDetector Class Reference . . . 9

5.1.1 Detailed Description . . . 10

5.1.2 Constructor & Destructor Documentation . . . 10

5.1.2.1 ~CameraTamperingDetector()=0 . . . 10

5.1.3 Member Function Documentation . . . 10

5.1.3.1 addIgnoredAngleForDisplacement(float angle_start, float angle_finish)=0 . . . 10

5.1.3.2 create() . . . 10

5.1.3.3 estimateIgnoredAnglesForDisplacement()=0 . . . 11

5.1.3.4 getParameterEstimation(DetectorParameter parameter_name) const =0 . . . 11

5.1.3.5 process(const cv::Mat &frame)=0 . . . 12

5.1.3.6 reset()=0 . . . 12

5.1.3.7 setFloatParameter(DetectorParameter parameter_name, float parameter_value)=0 12

5.1.3.8 setIntegerParameter(DetectorParameter parameter_name, std::int32_t parameter_value)=0 . . . 12

5.2 cva::ctd::Version Class Reference . . . 13

5.2.1 Detailed Description . . . 14

5.2.2 Constructor & Destructor Documentation . . . 14

5.2.2.1 Version(std::uint32_t major=0, std::uint32_t minor=0, std::uint32_t patch=0) . . . 14

5.2.3 Member Function Documentation . . . 14

5.2.3.1 major() const . . . 14

5.2.3.2 minor() const . . . 14

5.2.3.3 patch() const . . . 14

5.2.3.4 toString() const . . . 14

6 File Documentation **15**

6.1 ctd.hpp File Reference . . . 15

6.1.1 Detailed Description . . . 16

6.2 example.dox File Reference . . . 16

6.3 export.hpp File Reference . . . 16

6.3.1 Macro Definition Documentation . . . 16

6.3.1.1 CVA_CTD_EXPORT . . . 16

6.4 version.hpp File Reference . . . 16

6.4.1 Detailed Description . . . 16

7 Example Documentation **17**

7.1 main.cpp . . . 17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cva 7
cva::ctd 7

Chapter 2

Class Index

2.1 Class List

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

cva::ctd::CameraTamperingDetector
: The main class which provides functionality of camera tampering detection on the given sequence of images 9

cva::ctd::Version
: The class is used to represent the version number for the library 13

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

ctd.hpp
File containing definition of camera tampering detector interface 15
export.hpp 16
version.hpp
File containing definition of the interface for retrieving the version of the library 16

Chapter 4

Namespace Documentation

4.1 cva Namespace Reference

Namespaces

- ctd

4.2 cva::ctd Namespace Reference

Classes

- class CameraTamperingDetector

  The main class which provides functionality of camera tampering detection on the given sequence of images.
- class Version

  The class is used to represent the version number for the library.

Enumerations

- enum TamperingType {
  TAMPERING_INITIALIZING = 65536, TAMPERING_NONE = 0, TAMPERING_DEFOCUS = 1, TAMPERI↩
  NG_OCCLUSION = 2,
  TAMPERING_DISPLACEMENT = 4 }

  Tampering types detected by CameraTamperingDetector.
- enum DetectorParameter {
  DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD, DetectorParameter::OCCLUSION_AR↩
  EA_RATIO_WITHOUT_MOTION_THRESHOLD, DetectorParameter::DEFOCUS_RATIO_THRESHOLD,
  DetectorParameter::DISPLACEMENT_RATIO_THRESHOLD,
  DetectorParameter::OCCLUSION_COUNT_THRESHOLD, DetectorParameter::DEFOCUS_COUNT_TH↩
  RESHOLD, DetectorParameter::OCCLUSION_SENSITIVITY, DetectorParameter::HISTORY_SIZE_FOR↩
  _THRESHOLDS_ESTIMATION,
  DetectorParameter::FRAME_WIDTH }

  Tampering detector parameters.

Functions

- CVA_CTD_EXPORT Version version ()

  Get version number for the library.

4.2.1 Enumeration Type Documentation

4.2.1.1 enum cva::ctd::DetectorParameter [strong]

Tampering detector parameters.

Enumerator

OCCLUSION_AREA_RATIO_THRESHOLD Minimum occlusion size for detection relative to the frame size

OCCLUSION_AREA_RATIO_WITHOUT_MOTION_THRESHOLD Minimum motionless occlusion size for detection relative to the frame size

DEFOCUS_RATIO_THRESHOLD Minimum defocus level for detection

DISPLACEMENT_RATIO_THRESHOLD Minimum intersection size between proper camera view and new one for displacement detection relative to the frame size

OCCLUSION_COUNT_THRESHOLD Minimum number of consecutive frames with occlusion

DEFOCUS_COUNT_THRESHOLD Minimum number of consecutive frames with defocus

OCCLUSION_SENSITIVITY Sensitivity of occlusion detection

HISTORY_SIZE_FOR_THRESHOLDS_ESTIMATION Maximum history size for thresholds estimation

FRAME_WIDTH Frame width after internal resize

4.2.1.2 enum cva::ctd::TamperingType

Tampering types detected by CameraTamperingDetector.

Enumerator

TAMPERING_INITIALIZING Initializaing state of camera tampering detector (typicaly first 100 frames)

TAMPERING_NONE Normal state without tampering events

TAMPERING_DEFOCUS Defocus event detected on frame sequence (can be mixed with occlusion, displacement)

TAMPERING_OCCLUSION Occlusion event detected on frame sequence (can be mixed with defocus, displacement)

TAMPERING_DISPLACEMENT Displacement event detected on frame sequence (can be mixed with defocus, occlusion)

4.2.2 Function Documentation

4.2.2.1 CVA_CTD_EXPORT Version cva::ctd::version ( )

Get version number for the library.

Chapter 5

Class Documentation

5.1 cva::ctd::CameraTamperingDetector Class Reference

The main class which provides functionality of camera tampering detection on the given sequence of images.

`#include <ctd.hpp>`

Public Member Functions

- virtual TamperingType process (const cv::Mat &frame)=0

  Process the frame and detect camera tampering events.
- virtual void setFloatParameter (DetectorParameter parameter_name, float parameter_value)=0

  Set floating point parameter for camera tampering detector.
- virtual void setIntegerParameter (DetectorParameter parameter_name, std::int32_t parameter_value)=0

  Set integer parameter for camera tampering detector.
- virtual float getParameterEstimation (DetectorParameter parameter_name) const =0

  Get camera tampering detector parameter estimation for last DetectorParameter::HISTORY_SIZE_FOR_THRES↩ HOLDS_ESTIMATION frames.
- virtual void addIgnoredAngleForDisplacement (float angle_start, float angle_finish)=0

  Add ignore direction for displacement.
- virtual cv::Mat2f estimateIgnoredAnglesForDisplacement ()=0

  Estimate prohibited directions for displacement.
- virtual void reset ()=0

  Reset internal state to initial.
- virtual ~CameraTamperingDetector ()=0

Static Public Member Functions

- static CVA_CTD_EXPORT std::unique_ptr< CameraTamperingDetector > create ()

  A factory for CameraTamperingDetector.

5.1.1 Detailed Description

The main class which provides functionality of camera tampering detection on the given sequence of images.

Examples:

main.cpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 virtual cva::ctd::CameraTamperingDetector::~CameraTamperingDetector ( ) [pure virtual]

Destructor.

5.1.3 Member Function Documentation

5.1.3.1 virtual void cva::ctd::CameraTamperingDetector::addIgnoredAngleForDisplacement (float *angle_start,* float *angle_finish*) [pure virtual]

Add ignore direction for displacement.

Parameters

in	*angle_start*	start of the ignore angle.
in	*angle_finish*	finish of the ignore angle.

Parameters constraints: $-\pi \leq$ `angle_start` $\leq$ `angle_finish` $\leq \pi$

If you want add angle [π - π/4, π + π/4], you should split it: [π - π/4, π], [$-\pi$, $-\pi$ + π/4].

5.1.3.2 static CVA_CTD_EXPORT std::unique_ptr<CameraTamperingDetector> cva::ctd::CameraTamperingDetector::create ( ) [static]

A factory for CameraTamperingDetector.

Returns

An instance of a camera tampering detector implementing CameraTamperingDetector interface.

Examples:

main.cpp.

5.1.3.3 virtual cv::Mat2f cva::ctd::CameraTamperingDetector::estimateIgnoredAnglesForDisplacement ( ) [pure virtual]

Estimate prohibited directions for displacement.

Returns

OpenCV matrix with two columns: start and finish of the ignore angle.

Estimation based on the value of DetectorParameter::DISPLACEMENT_RATIO_THRESHOLD. If frame size too small estimation may not be computed.

5.1.3.4 virtual float cva::ctd::CameraTamperingDetector::getParameterEstimation (DetectorParameter *parameter_name*) const [pure virtual]

Get camera tampering detector parameter estimation for last DetectorParameter::HISTORY_SIZE_FOR_THRESHOLDS_ESTIMATION frames.

Parameters

in	*parameter_name*	parameter name.

Returns

Estimated value.

Possible values of `parameter_name`:

- DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD (estimation based on the DetectorParameter::OCCLUSION_COUNT_THRESHOLD).
- DetectorParameter::OCCLUSION_AREA_RATIO_WITHOUT_MOTION_THRESHOLD (estimation based on the DetectorParameter::OCCLUSION_COUNT_THRESHOLD).
- DetectorParameter::DEFOCUS_RATIO_THRESHOLD (estimation based on the DetectorParameter::DEFOCUS_COUNT_THRESHOLD).
- DetectorParameter::DISPLACEMENT_RATIO_THRESHOLD.

After the call, the history for OCCLUSION and DEFOCUS is not cleared completely, the history for DISPLACEMENT is not cleared at all. For OCCLUSION last DetectorParameter::OCCLUSION_COUNT_THRESHOLD frames are left. For DEFOCUS last DetectorParameter::DEFOCUS_COUNT_THRESHOLD are left.

Parameter estimation on 1000 frames (default value for DetectorParameter::HISTORY_SIZE_FOR_THRESHOLDS_ESTIMATION) is not fast. To avoid this you can perform estimation every 250-500 frames and choose minimum value for DEFOCUS_RATIO_THRESHOLD and maximum for others.

For example, if you want to estimate DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD on first 1000 frames you can call getParameterEstimation(DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD) 4 times every 250 frames and choose maximum value from estimation or call getParameterEstimation(OCCLUSION_AREA_RATIO_THRESHOLD) once on the 1000 frame. Estimation will be the same.

Estimation performs under the assumption of zero false alarm rate, so in some cases, the tampering event may not be detected due to high risk of false alarms. If tampering detection rate more important than false alarm rate it is recommended to change values of estimated parameters.

Examples:

main.cpp.

5.1.3.5 virtual TamperingType cva::ctd::CameraTamperingDetector::process (const cv::Mat & *frame*) [pure virtual]

Process the frame and detect camera tampering events.

Parameters

in	*frame*	the input frame in BGR(RGBA) or grayscale format.

Returns

Tampering type detected by detector.

5.1.3.6 virtual void cva::ctd::CameraTamperingDetector::reset ( ) [pure virtual]

Reset internal state to initial.

5.1.3.7 virtual void cva::ctd::CameraTamperingDetector::setFloatParameter (DetectorParameter *parameter_name,* float *parameter_value*) [pure virtual]

Set floating point parameter for camera tampering detector.

Parameters

in	*parameter_name*	parameter name.
in	*parameter_value*	parameter value.

Name	Description	Value range	Default value
OCCLUSION_AREA_RATIO_THRESHOLD	Minimum occlusion size for detection relative to the frame size	0 - 1 (detect only occlusion closing camera lens completely)	0.45
OCCLUSION_AREA_RATIO_WITHOUT_MOTION_THRESHOLD	Minimum motionless occlusion size for detection relative to the frame size	0 - 1 (detect only motionless occlusion closing camera lens completely)	0.3
DEFOCUS_RATIO_THRESHOLD	Minimum defocus level for detection	0 (detect only strong defocus) - 1	0.3
DISPLACEMENT_RATIO_THRESHOLD	Minimum displacement ratio for detection	0 (detect any slight camera view change) - 1 (camera view changed completely)	0.5

Examples:

main.cpp.

5.1.3.8 virtual void cva::ctd::CameraTamperingDetector::setIntegerParameter (DetectorParameter *parameter_name,* std::int32_t *parameter_value*) [pure virtual]

Set integer parameter for camera tampering detector.

Parameters

in	*parameter_name*	parameter name.
in	*parameter_value*	parameter value.

Name	Description	Value range	Default value
OCCLUSION_COUNT_THRESHOLD	Minimum number of consecutive frames with occlusion	≥ 1	30
DEFOCUS_COUNT_THRESHOLD	Minimum number of consecutive frames with defocus	≥ 1	30
OCCLUSION_SENSITIVITY	Sensitivity of occlusion detection	1 (highest sensitive) - 128 (lowest sensitive)	25
HISTORY_SIZE_FOR_THRESHOLDS_ESTIMATION	Maximum history size for thresholds estimation. If 0 then do not collect data for threshold estimation	≥ 0	1000
FRAME_WIDTH	Frame width after internal resize. Height is computed relative to the original frame aspect ratio. You have the possibility of optimal trade-off choice between performance and accuracy. A small frame size can cause camera tampering events not to be detected	32 - 640	280

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

- ctd.hpp

5.2 cva::ctd::Version Class Reference

The class is used to represent the version number for the library.

`#include <version.hpp>`

Public Member Functions

- Version (std::uint32_t major=0, std::uint32_t minor=0, std::uint32_t patch=0)
- std::uint32_t major () const

  Get major number of the version.
- std::uint32_t minor () const

  Get minor number of the version.
- std::uint32_t patch () const

  Get patch number of the version.
- std::string toString () const

  Get version string: major.minor.revision.

5.2.1 Detailed Description

The class is used to represent the version number for the library.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 cva::ctd::Version::Version (std::uint32_t *major* = 0, std::uint32_t *minor* = 0, std::uint32_t *patch* = 0) [inline], [explicit]

5.2.3 Member Function Documentation

5.2.3.1 std::uint32_t cva::ctd::Version::major ( ) const [inline]

Get major number of the version.

5.2.3.2 std::uint32_t cva::ctd::Version::minor ( ) const [inline]

Get minor number of the version.

5.2.3.3 std::uint32_t cva::ctd::Version::patch ( ) const [inline]

Get patch number of the version.

5.2.3.4 std::string cva::ctd::Version::toString ( ) const [inline]

Get version string: major.minor.revision.

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

- version.hpp

Chapter 6

File Documentation

6.1 ctd.hpp File Reference

File containing definition of camera tampering detector interface.

```
#include <cva/ctd/export.hpp>
#include <cva/ctd/version.hpp>
#include <opencv2/core.hpp>
#include <memory>
```

Classes

- class cva::ctd::CameraTamperingDetector

  The main class which provides functionality of camera tampering detection on the given sequence of images.

Namespaces

- cva
- cva::ctd

Enumerations

- enum cva::ctd::TamperingType {
  cva::ctd::TAMPERING_INITIALIZING = 65536, cva::ctd::TAMPERING_NONE = 0, cva::ctd::TAMPERING↩
  _DEFOCUS = 1, cva::ctd::TAMPERING_OCCLUSION = 2,
  cva::ctd::TAMPERING_DISPLACEMENT = 4 }

  Tampering types detected by CameraTamperingDetector.
- enum cva::ctd::DetectorParameter {
  cva::ctd::DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD, cva::ctd::DetectorParameter::O↩
  CCLUSION_AREA_RATIO_WITHOUT_MOTION_THRESHOLD, cva::ctd::DetectorParameter::DEFOCU↩
  S_RATIO_THRESHOLD, cva::ctd::DetectorParameter::DISPLACEMENT_RATIO_THRESHOLD,
  cva::ctd::DetectorParameter::OCCLUSION_COUNT_THRESHOLD, cva::ctd::DetectorParameter::DEFOC↩
  US_COUNT_THRESHOLD, cva::ctd::DetectorParameter::OCCLUSION_SENSITIVITY, cva::ctd::Detector↩
  Parameter::HISTORY_SIZE_FOR_THRESHOLDS_ESTIMATION,
  cva::ctd::DetectorParameter::FRAME_WIDTH }

  Tampering detector parameters.

6.1.1 Detailed Description

File containing definition of camera tampering detector interface.

6.2 example.dox File Reference

6.3 export.hpp File Reference

Macros

- #define CVA_CTD_EXPORT __attribute__((visibility("default")))

6.3.1 Macro Definition Documentation

6.3.1.1 #define CVA_CTD_EXPORT __attribute__((visibility("default")))

6.4 version.hpp File Reference

File containing definition of the interface for retrieving the version of the library.

```
#include <cva/ctd/export.hpp>
#include <cstdint>
#include <string>
```

Classes

- class cva::ctd::Version

  The class is used to represent the version number for the library.

Namespaces

- cva
- cva::ctd

Functions

- CVA_CTD_EXPORT Version cva::ctd::version ()

  Get version number for the library.

6.4.1 Detailed Description

File containing definition of the interface for retrieving the version of the library.

Chapter 7

Example Documentation

7.1 main.cpp

```
/*
    Copyright 2018 Intel Corporation.

    This software and the related documents are Intel copyrighted materials,
    and your use of them is governed by the express license under which they
    were provided to you (Intel Simplified Software License (Version April 2018))
    Unless the License provides otherwise, you may not use, modify,
    copy, publish, distribute, disclose or transmit this software or
    the related documents without Intel's prior written permission.

    This software and the related documents are provided as is, with no
    express or implied warranties, other than those that are expressly
    stated in the License.
*/

#include "cva/ctd/ctd.hpp"

#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/videoio.hpp>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>

const char* kWindowName     = "Camera tampering detection demo";
const int kEscapeKey        = 27;
const size_t kFramesForInit = 100;

const char* kAbout = "This is camera tampering detection example.\n";
const char* kOptions        =
    "{ h ? help usage |                   | print help message }"
    "{@video          |                   | input video (if not specified then use camera device with id 0)
       }"
    "{show-gui s      | false             | whether to show gui }"
    ;

void SetupParameters(cva::ctd::CameraTamperingDetector &ct_detector) {
  try {
    float occlusion_area_ratio_threshold_estimation =
        ct_detector.getParameterEstimation(
      cva::ctd::DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD
      );
    float occlusion_area_ratio_threshold = 0.45f;
    occlusion_area_ratio_threshold = std::max(occlusion_area_ratio_threshold,
                                              occlusion_area_ratio_threshold_estimation);
    ct_detector.setFloatParameter(
      cva::ctd::DetectorParameter::OCCLUSION_AREA_RATIO_THRESHOLD
      ,
                                  occlusion_area_ratio_threshold);

    float occlusion_area_ratio_without_motion_threshold_estimation =
        ct_detector.getParameterEstimation(
      cva::ctd::DetectorParameter::OCCLUSION_AREA_RATIO_WITHOUT_MOTION_THRESHOLD
      );
```

```
    float occlusion_area_ratio_without_motion_threshold = 0.3f;
    occlusion_area_ratio_without_motion_threshold = std::max(occlusion_area_ratio_without_motion_threshold,

      occlusion_area_ratio_without_motion_threshold_estimation);
    ct_detector.setFloatParameter(
      cva::ctd::DetectorParameter::OCCLUSION_AREA_RATIO_WITHOUT_MOTION_THRESHOLD
      ,
                                  occlusion_area_ratio_without_motion_threshold);

    float defocus_ratio_threshold_estimation =
        ct_detector.getParameterEstimation(
      cva::ctd::DetectorParameter::DEFOCUS_RATIO_THRESHOLD);
    float defocus_ratio_threshold = 0.5f;
    defocus_ratio_threshold = std::min(defocus_ratio_threshold, defocus_ratio_threshold_estimation);
    ct_detector.setFloatParameter(
      cva::ctd::DetectorParameter::DEFOCUS_RATIO_THRESHOLD,
                                  defocus_ratio_threshold);

    float displacement_ratio_threshold_estimation =
        ct_detector.getParameterEstimation(
      cva::ctd::DetectorParameter::DISPLACEMENT_RATIO_THRESHOLD
      );
    float displacement_ratio_threshold = 0.5f;
    displacement_ratio_threshold = std::max(displacement_ratio_threshold,
      displacement_ratio_threshold_estimation);
    ct_detector.setFloatParameter(
      cva::ctd::DetectorParameter::DISPLACEMENT_RATIO_THRESHOLD
      ,
                                  displacement_ratio_threshold);
  } catch (cv::Exception) {
    std::cerr << "Something wrong with parameters configuration!" << std::endl;
    exit(-1);
  }
}

int main(int argc, const char** argv) {
  cv::CommandLineParser parser(argc, argv, kOptions);
  parser.about(kAbout);

  if (parser.has("help")) {
    parser.printMessage();
    return 0;
  }

  cv::String video_path = parser.get<cv::String>("@video");
  if (!parser.check()) {
    parser.printErrors();
    return -1;
  }

  cv::VideoCapture capture;
  if (video_path.empty()) {
    capture.open(0);
    if (!capture.isOpened()) {
      std::cout << "Failed to open camera device" << std::endl;
      return -1;
    }
  } else {
    capture.open(video_path);

    if (!capture.isOpened()) {
      std::cout << "Failed to open video: " << video_path << std::endl;
      return -1;
    }
  }

  bool use_gui = parser.get<bool>("show-gui");

  if (use_gui) {
    cv::namedWindow(kWindowName, cv::WINDOW_AUTOSIZE);
    cv::moveWindow(kWindowName, 0, 0);
  }

  // Create camera tampering detector
  auto ct_detector = cva::ctd::CameraTamperingDetector::create();

  cv::Mat frame;
  capture >> frame;
  size_t frame_idx = 0;
  bool is_tampering = false;

  while (!frame.empty()) {
    cva::ctd::TamperingType tampering_type = ct_detector->process(frame);
    std::string state;

    if (frame_idx < kFramesForInit) {  // Collect data for parameters esimation
      state = "Waiting";
```

```
    } else if (frame_idx == kFramesForInit) {  // Estimate and setup parameters
      state = "Updating parameters";
      SetupParameters(*ct_detector);
    } else {  // Process camera tampering events
      switch (tampering_type) {
        case cva::ctd::TAMPERING_INITIALIZING: {
          state = "INITIALIZING";
          break;
        }
        case cva::ctd::TAMPERING_NONE: {
          state = "OK";
          if (is_tampering) {  // Reset after Tampering event (Occlusion or Defocus only)
            is_tampering = false;
            ct_detector->reset();
            frame_idx = 0;
          }
          break;
        }
        default: {
          is_tampering = true;
          if (tampering_type & cva::ctd::TAMPERING_DEFOCUS)
            state += "|DEFOCUS|";
          if (tampering_type & cva::ctd::TAMPERING_OCCLUSION)
            state += "|OCCLUSION|";
          if (tampering_type & cva::ctd::TAMPERING_DISPLACEMENT)
            state += "|DISPLACEMENT|";
        }
      };
    }

    const int         font_face          = cv::FONT_HERSHEY_COMPLEX;
    const double      font_scale         = 1.;
    const int         font_thickness     = 2;
    const cv::Scalar  foreground_color   = cv::Scalar(0, 0, 255);
    cv::putText(frame, state, cv::Point(10, 30), font_face, font_scale, foreground_color, font_thickness);

    if (use_gui) {
      cv::imshow(kWindowName, frame);

      int key = cv::waitKey(10) & 0x00FF;
      if (key == kEscapeKey) {
        break;
      }
    } else {
      std::stringstream ss;
      ss << std::setw(6) << std::setfill('0') << frame_idx;
      cv::imwrite(ss.str() + ".png", frame);
    }

    capture >> frame;
    ++frame_idx;
  }

  return 0;
}
```