GECKO

1.0

Generated by Doxygen 1.7.4

Mon Dec 16 2013 00:14:49

Contents

1 Class Index

1.1 Class List

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

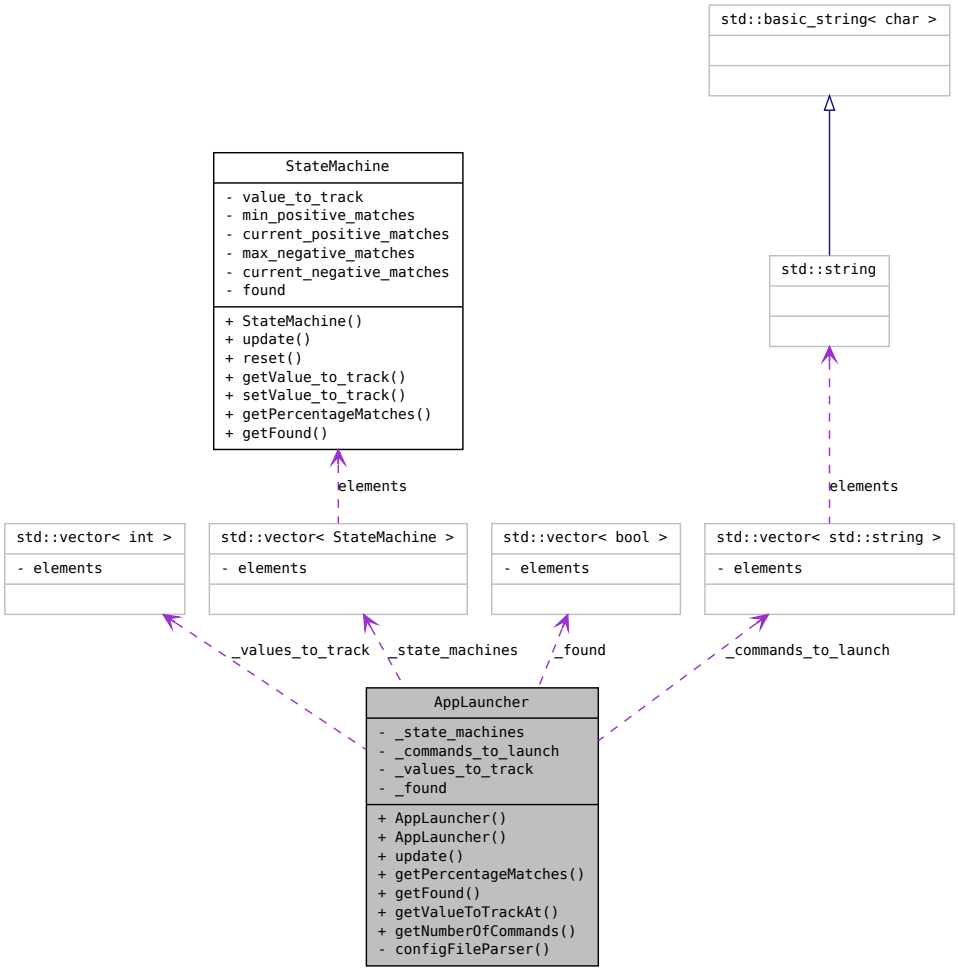
2 Class Documentation

2.1 AppLauncher Class Reference

Launchs applications triggered by events generated by **StateMachine** (p. **??**) (s)

`#include <AppLauncher.h>`

Collaboration diagram for AppLauncher:



Public Member Functions

- **AppLauncher** (std::string config_file, int positive_matches, int negative_matches)

  AppLauncher *(*p. **??***) constructor.*
- **AppLauncher** (std::string config_file, std::vector< int > positive_matches, std::vector< int > negative_matches)

  AppLauncher *(*p. **??***) constructor.*
- void **update** (int current_value)

  Update the state of internal state machines and trigger the corresponding commands.

- float **getPercentageMatches** (int i)

  Returns the current number of positive matches of the ith value as a percentage.
- bool **getFound** (int i)

  Gets the current state of the ith state machine.
- int **getValueToTrackAt** (int i)

  Returns the current value set to track for the ith command.
- int **getNumberOfCommands** ()

  Returns the number of commands loaded.

Private Member Functions

- void **configFileParser** (std::string config_file)

  Reads and parses a configuration file containing the commands and values that trigger them.

Private Attributes

- std::vector< **StateMachine** > **_state_machines**

  Vector containing the state machines for each command.
- std::vector< std::string > **_commands_to_launch**

  Vector containing the commmands to launch for each value.
- std::vector< int > **_values_to_track**

  Vector containing the values that trigger each command.
- std::vector< bool > **_found**

  Vector containing the current state of each state machine.

2.1.1 Detailed Description

Launchs applications triggered by events generated by **StateMachine** (p. **??**) (s)

This class loads the programs and values that trigger them from a file.

The format of the file is:

$program1$ 1

$program2$ 2

Where the string between the dollar characters is the command to launch and the value at the end of each line is the value that triggers that commands.

2.1.2 Constructor & Destructor Documentation

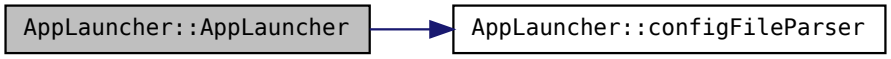
2.1.2.1 AppLauncher::AppLauncher (std::string *config_file*, int *positive_matches*, int *negative_matches*)

AppLauncher (p. **??**) constructor.

Parameters

config_file	Path of the configuration file containing the program info
positive_matches	Minimum number of positive matches needed to trigger a command
negative_matches	Maximum number of negative matches allowed between two positive matches

Here is the call graph for this function:



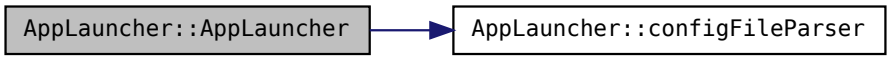
2.1.2.2 AppLauncher::AppLauncher (std::string *config_file*, std::vector< int > *positive_matches*, std::vector< int > *negative_matches*)

AppLauncher (p. **??**) constructor.

Parameters

config_file	Path of the configuration file containing the program info
positive_matches	Vector containing the minimum number of positive matches needed to trigger each command individually
negative_matches	Vector containing the maximum number of negative matches allowed between two positive matches for each command individually

Here is the call graph for this function:



2.1.3 Member Function Documentation

2.1.3.1 void AppLauncher::configFileParser (std::string *config_file*) [private]

Reads and parses a configuration file containing the commands and values that trigger them.

Parameters

config_file	Path to the configuration file

2.1.3.2 bool AppLauncher::getFound (int *i*)

Gets the current state of the ith state machine.

2.1.3.3 int AppLauncher::getNumberOfCommands ( )

Returns the number of commands loaded.

2.1.3.4 float AppLauncher::getPercentageMatches (int *i*)

Returns the current number of positive matches of the ith value as a percentage.

2.1.3.5 int AppLauncher::getValueToTrackAt (int *i*)

Returns the current value set to track for the ith command.

2.1.3.6 void AppLauncher::update (int *current_value*)

Update the state of internal state machines and trigger the corresponding commands.

Parameters

current_value	Current value to compare with the value to track in all internal state machines

2.1.4 Member Data Documentation

2.1.4.1 std::vector<std::string> AppLauncher::_commands_to_launch [private]

Vector containing the commmands to launch for each value.

2.1.4.2 std::vector<bool> AppLauncher::_found [private]

Vector containing the current state of each state machine.

2.1.4.3 std::vector<StateMachine> AppLauncher::_state_machines [private]

Vector containing the state machines for each command.

2.1.4.4 std::vector<int> AppLauncher::_values_to_track [private]

Vector containing the values that trigger each command.

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

- AppLauncher.h
- AppLauncher.cpp

2.2 backgroundSubstractor Class Reference

`#include <backgroundSubstractor.h>`

Public Member Functions

- void **setbackgroundRatio** (float a)

2.2.1 Member Function Documentation

2.2.1.1 void backgroundSubstractor::setbackgroundRatio (float *a*) [inline]

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

- backgroundSubstractor.h

2.3 ConvexityDefect Struct Reference

Stores all the characteristics of a convexity defect in a more convenient way.

`#include <handUtils.h>`

Public Attributes

- cv::Point **start**

  Starting point of the defect.
- cv::Point **end**

  Ending point of the defect.
- cv::Point **depth_point**

  Deepest point of the defect.
- double **depth**

  Depth of the defect.
- int **start_index**

  Index of the starting point of the defect inside the original contour.
- int **end_index**

  Index of the ending point of the defect inside the original contour.

- int **depth_point_index**

  Index of the deepest point of the defect inside the original contour.

2.3.1 Detailed Description

Stores all the characteristics of a convexity defect in a more convenient way.

2.3.2 Member Data Documentation

2.3.2.1 double ConvexityDefect::depth

Depth of the defect.

2.3.2.2 cv::Point ConvexityDefect::depth_point

Deepest point of the defect.

2.3.2.3 int ConvexityDefect::depth_point_index

Index of the deepest point of the defect inside the original contour.

2.3.2.4 cv::Point ConvexityDefect::end

Ending point of the defect.

2.3.2.5 int ConvexityDefect::end_index

Index of the ending point of the defect inside the original contour.

2.3.2.6 cv::Point ConvexityDefect::start

Starting point of the defect.

2.3.2.7 int ConvexityDefect::start_index

Index of the starting point of the defect inside the original contour.

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

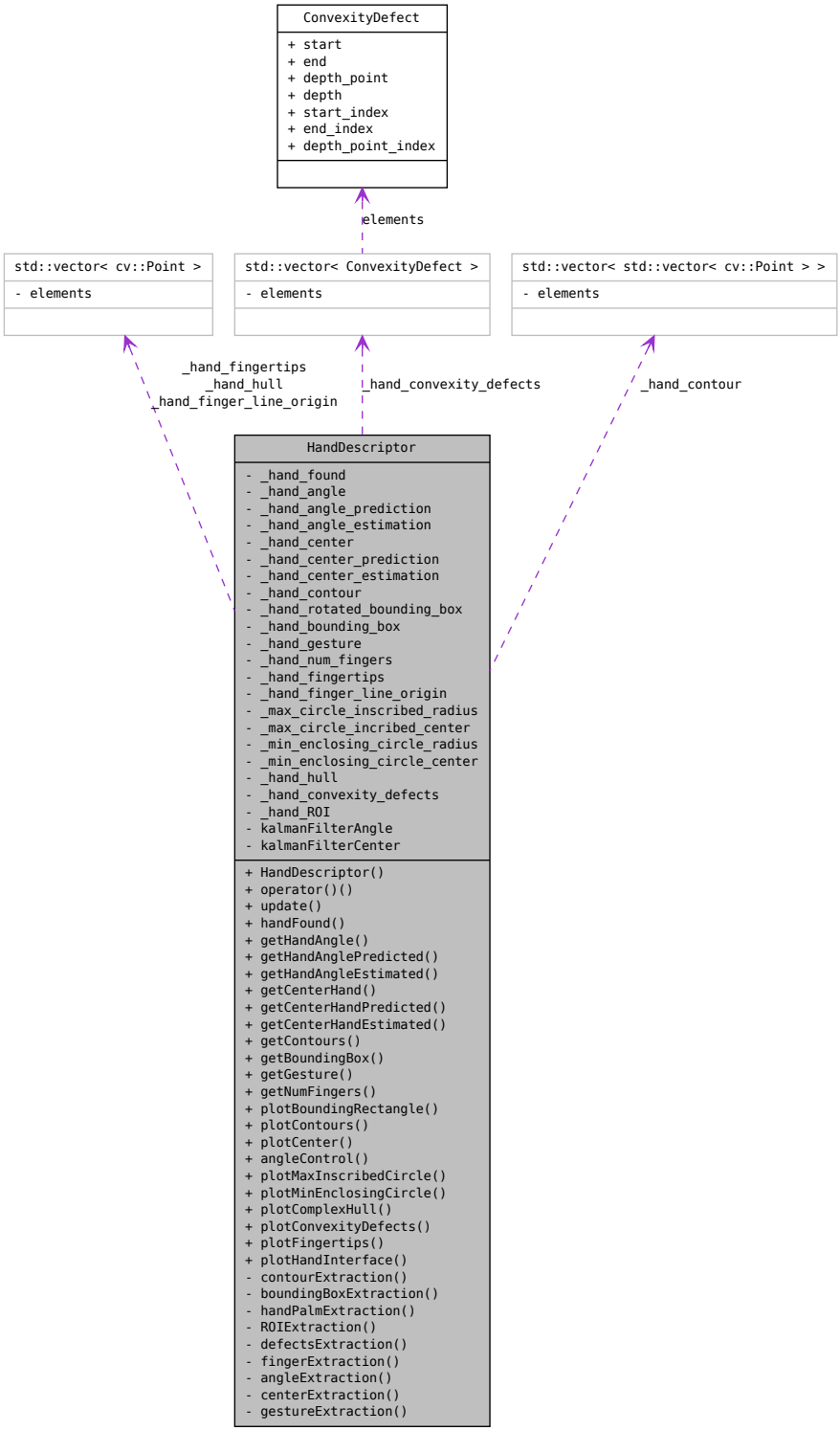
- handUtils.h

2.4 HandDescriptor Class Reference

Finds the main characteristics of the hand from a binary image containing a hand silouette.

```
#include <HandDescriptor.h>
```

Collaboration diagram for HandDescriptor:



Public Member Functions

- **HandDescriptor** ()

  Default constructor.
- void **operator()** (const cv::Mat &skinMask)

  Update the internal characteristics stored.
- void **update** (const cv::Mat &skinMask)

  Update the internal characteristics stored.
- bool **handFound** ()

  Returns true if a hand was found.
- double **getHandAngle** ()

  Returns the angle of the box enclosing the hand.
- double **getHandAnglePredicted** ()

  Returns the angle of the box enclosing the hand predicted by the Kalman filter.
- double **getHandAngleEstimated** ()

  Returns the angle of the box enclosing the hand estimated by the Kalman filter after updating the prediction with the actual value.
- cv::Point **getCenterHand** ()

  Returns the position of the center of the hand.
- cv::Point **getCenterHandPredicted** ()

  Returns the position of the center of the hand predicted by the Kalman filter.
- cv::Point **getCenterHandEstimated** ()

  Returns the position of the center of the hand estimated by the Kalman filter after updating the prediction with the actual value.
- std::vector< std::vector< cv::Point > > **getContours** ()

  Returns the contours of the detected hand.
- cv::Rect **getBoundingBox** ()

  Returns the bounding box enclosing the detected hand.
- int **getGesture** ()

  Returns the detected gesture.
- int **getNumFingers** ()

  Returns the number of fingers found.
- void **plotBoundingRectangle** (const cv::Mat &src, cv::Mat &dst, bool rotated=true)

  Plots the rectangle around the hand on display.
- void **plotContours** (const cv::Mat &src, cv::Mat &dst)

  Plot the contours on display.
- void **plotCenter** (const cv::Mat &src, cv::Mat &dst, bool show_corrected=true, bool show_actual=true, bool show_predicted=true)

  Plot the center of the hand on display.
- void **angleControl** (bool show_corrected=true, bool show_actual=true, bool show_predicted=true)

  Prints the angle gauge on a separate window.
- void **plotMaxInscribedCircle** (cv::Mat &src, cv::Mat &dst, bool show_center=true, cv::Scalar color=cv::Scalar(0, 255, 0), int thickness=1)

Prints the maximum inscribed circle:

- void **plotMinEnclosingCircle** (cv::Mat &src, cv::Mat &dst, bool show_center=true, cv::Scalar color=cv::Scalar(255, 0, 255), int thickness=1)

  Prints the minimum enclosing circle:
- void **plotComplexHull** (cv::Mat &src, cv::Mat &dst, bool show_points=false, cv::Scalar color=cv::Scalar(0, 255, 255), int thickness=1)

  Plot the hand complex hull:
- void **plotConvexityDefects** (cv::Mat &src, cv::Mat &dst, bool draw_points=true)

  Plot convexity defects:
- void **plotFingertips** (cv::Mat &src, cv::Mat &dst, bool draw_lines=true, cv::Scalar color=cv::Scalar(255, 0, 0), int thickness=2)

  Plot fingertip markers:
- void **plotHandInterface** (cv::Mat &src, cv::Mat &dst)

  Plot hand interface:

Private Member Functions

- void **contourExtraction** (const cv::Mat &skinMask)

  Extract the hand contours from a binary image containing hand candidates.
- void **boundingBoxExtraction** ()

  Extracts the bounding boxes around the hand contour (rectangle and rotated rectange)
- void **handPalmExtraction** ()

  Finds the maximum inscribed circle of the hand contour, which describes the hand palm.
- void **ROIExtraction** (const cv::Mat &src)

  Extracts a mask of the region of interest in which the hand is contained.
- void **defectsExtraction** ()

  Finds the convexity defects of the hand convex hull, that are used to find the fingers.
- void **fingerExtraction** ()

  Extracts the number, position and orientation of the fingers.
- void **angleExtraction** ()

  Extracts the hand angle from its bounding box and a Kalman Filter.
- void **centerExtraction** ()

  Extracts the hand center and applies a Kalman Filter for improved stability.
- void **gestureExtraction** ()

  Guesses the hand gesture using the hand characteristic data previouly found.

Private Attributes

- bool **_hand_found**

  Whether a hand was found or not:
- double **_hand_angle**

  Contains the actual angle of the box enclosing the hand.

- double **_hand_angle_prediction**

  Contains the predicted angle by the kalman filter.
- double **_hand_angle_estimation**

  Contains the corrected estimation by the kalman filter.
- cv::Point **_hand_center**

  Actual center of the hand.
- cv::Point **_hand_center_prediction**

  Predicted center of the hand by the kalman filter.
- cv::Point **_hand_center_estimation**

  Corrected estimation by the kalman filter.
- std::vector< std::vector< cv::Point > > **_hand_contour**

  Contours of the candidates to be a hand.
- cv::RotatedRect **_hand_rotated_bounding_box**

  Minimum RotatedRect enclosing the hand.
- cv::Rect **_hand_bounding_box**

  Minimum Rect enclosing the hand.
- int **_hand_gesture**

  Last detected gesture, coded as an integer (see constants for correspondence between integer and gesture)
- int **_hand_num_fingers**

  Number of fingers (visible)
- std::vector< cv::Point > **_hand_fingertips**

  Position of the fingertips.
- std::vector< cv::Point > **_hand_finger_line_origin**

  Position of the finger line origin points.
- double **_max_circle_inscribed_radius**

  Radius of the max. inscribed circle.
- cv::Point **_max_circle_incribed_center**

  Center of the max. incribed circle.
- float **_min_enclosing_circle_radius**

  Radius of the min. enclosing circle.
- cv::Point2f **_min_enclosing_circle_center**

  Center of the min. enclosing circle.
- std::vector< cv::Point > **_hand_hull**

  Complex hull of the hand.
- std::vector< **ConvexityDefect** > **_hand_convexity_defects**

  Convexity defects of the hand.
- cv::Mat **_hand_ROI**

  Mask containing ROI of the hand.
- cv::KalmanFilter **kalmanFilterAngle**

  Kalman filter for the angle of the box enclosing the hand.
- cv::KalmanFilter **kalmanFilterCenter**

  Kalman filter for the center of the hand.

2.4.1 Detailed Description

Finds the main characteristics of the hand from a binary image containing a hand silouette.

To look for or to update the stored characteristics of the hand and its gesture, the update member or the operator () can be called. One can then know if a hand was found using handFound member, prior to retrieving any of the hand characteristics / gesture.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 HandDescriptor::HandDescriptor ( )

Default constructor.

2.4.3 Member Function Documentation

2.4.3.1 void HandDescriptor::angleControl (bool *show_corrected* = `true`, bool *show_actual* = `true`, bool *show_predicted* = `true`)

Prints the angle gauge on a separate window.

2.4.3.2 void HandDescriptor::angleExtraction ( ) [private]

Extracts the hand angle from its bounding box and a Kalman Filter.

2.4.3.3 void HandDescriptor::boundingBoxExtraction ( ) [private]

Extracts the bounding boxes around the hand contour (rectangle and rotated rectange)

2.4.3.4 void HandDescriptor::centerExtraction ( ) [private]

Extracts the hand center and applies a Kalman Filter for improved stability.

2.4.3.5 void HandDescriptor::contourExtraction (const cv::Mat & *skinMask*) [private]

Extract the hand contours from a binary image containing hand candidates.

After the contour extraction it filters out the smaller contours, that are likely to be noise, and carries a polygon approximation to reduce the number of points in the contour.

Parameters

skinMask	Binary image containing the hand candidates, previously filtered by a **HandDetector** (p. **??**) object.

2.4.3.6 void HandDescriptor::defectsExtraction ( ) [private]

Finds the convexity defects of the hand convex hull, that are used to find the fingers.

2.4.3.7 void HandDescriptor::fingerExtraction ( ) [private]

Extracts the number, position and orientation of the fingers.

2.4.3.8 void HandDescriptor::gestureExtraction ( ) [private]

Guesses the hand gesture using the hand characteristic data previouly found.

2.4.3.9 cv::Rect HandDescriptor::getBoundingBox ( )

Returns the bounding box enclosing the detected hand.

2.4.3.10 cv::Point HandDescriptor::getCenterHand ( )

Returns the position of the center of the hand.

2.4.3.11 cv::Point HandDescriptor::getCenterHandEstimated ( )

Returns the position of the center of the hand estimated by the Kalman filter after updating the prediction with the actual value.

2.4.3.12 cv::Point HandDescriptor::getCenterHandPredicted ( )

Returns the position of the center of the hand predicted by the Kalman filter.

2.4.3.13 std::vector< std::vector< cv::Point > > HandDescriptor::getContours ( )

Returns the contours of the detected hand.

2.4.3.14 int HandDescriptor::getGesture ( )

Returns the detected gesture.

2.4.3.15 double HandDescriptor::getHandAngle ( )

Returns the angle of the box enclosing the hand.

2.4.3.16 double HandDescriptor::getHandAngleEstimated ( )

Returns the angle of the box enclosing the hand estimated by the Kalman filter after updating the prediction with the actual value.

2.4.3.17 double HandDescriptor::getHandAnglePredicted ( )

Returns the angle of the box enclosing the hand predicted by the Kalman filter.

2.4.3.18 int HandDescriptor::getNumFingers ( )

Returns the number of fingers found.

2.4.3.19 bool HandDescriptor::handFound ( )

Returns true if a hand was found.

2.4.3.20 void HandDescriptor::handPalmExtraction ( ) [private]

Finds the maximum inscribed circle of the hand contour, which describes the hand palm.

2.4.3.21 void HandDescriptor::operator() (const cv::Mat & *skinMask*)

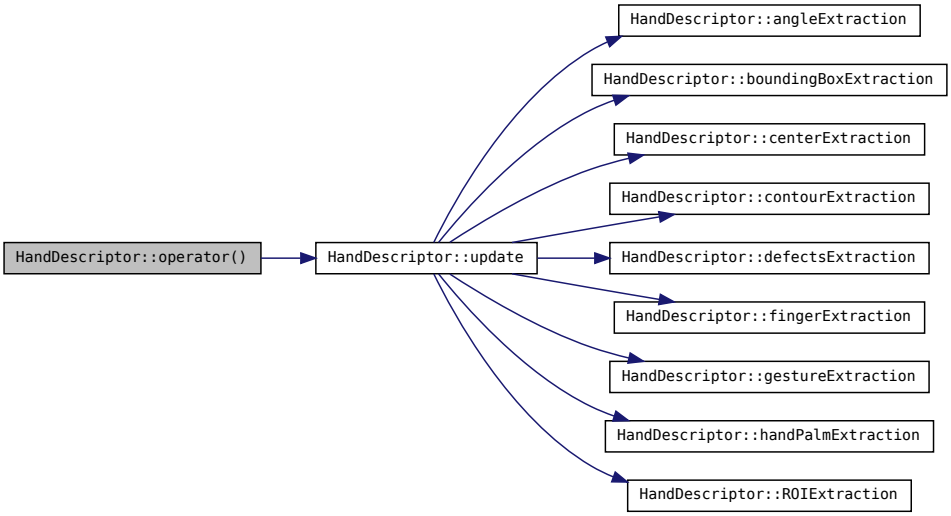
Update the internal characteristics stored.

This is a wrapper of the update function, to call it in a more intuitive way.

Parameters

skinMask	Binary image containing the skin zones of hand candidates

Here is the call graph for this function:



2.4.3.22 void HandDescriptor::plotBoundingRectangle (const cv::Mat & *src*, cv::Mat & *dst*, bool *rotated* = true)

Plots the rectangle around the hand on display.

2.4.3.23 void HandDescriptor::plotCenter (const cv::Mat & *src*, cv::Mat & *dst*, bool *show_corrected* = true, bool *show_actual* = true, bool *show_predicted* = true)

Plot the center of the hand on display.

2.4.3.24 void HandDescriptor::plotComplexHull (cv::Mat & *src*, cv::Mat & *dst*, bool *show_points* = `false`, cv::Scalar *color* = `cv::Scalar( 0, 255, 255)`, int *thickness* = `1`)

Plot the hand complex hull:

2.4.3.25 void HandDescriptor::plotContours (const cv::Mat & *src*, cv::Mat & *dst*)

Plot the contours on display.

2.4.3.26 void HandDescriptor::plotConvexityDefects (cv::Mat & *src*, cv::Mat & *dst*, bool *draw_points* = `true`)

Plot convexity defects:

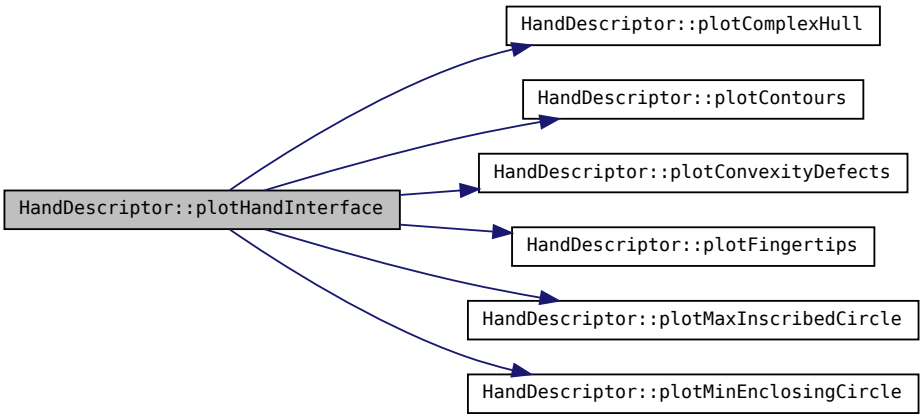
2.4.3.27 void HandDescriptor::plotFingertips (cv::Mat & *src*, cv::Mat & *dst*, bool *draw_lines* = `true`, cv::Scalar *color* = `cv::Scalar( 255, 0, 0)`, int *thickness* = `2`)

Plot fingertip markers:

2.4.3.28 void HandDescriptor::plotHandInterface (cv::Mat & *src*, cv::Mat & *dst*)

Plot hand interface:

Here is the call graph for this function:



2.4.3.29 void HandDescriptor::plotMaxInscribedCircle (cv::Mat & *src*, cv::Mat & *dst*, bool *show_center* = `true`, cv::Scalar *color* = `cv::Scalar(0, 255, 0)`, int *thickness* = `1`)

Prints the maximum inscribed circle:

2.4.3.30 void HandDescriptor::plotMinEnclosingCircle (cv::Mat & *src*, cv::Mat & *dst*, bool *show_center* = `true`, cv::Scalar *color* = `cv::Scalar(255,0,255)`, int *thickness* = `1`)

Prints the minimum enclosing circle:

2.4.3.31 void HandDescriptor::ROIExtraction (const cv::Mat & *src*) [private]

Extracts a mask of the region of interest in which the hand is contained.

Parameters

src	Binary image containing the hand candidates, to extract the dimensions of the mask image

2.4.3.32 void HandDescriptor::update (const cv::Mat & *skinMask*)

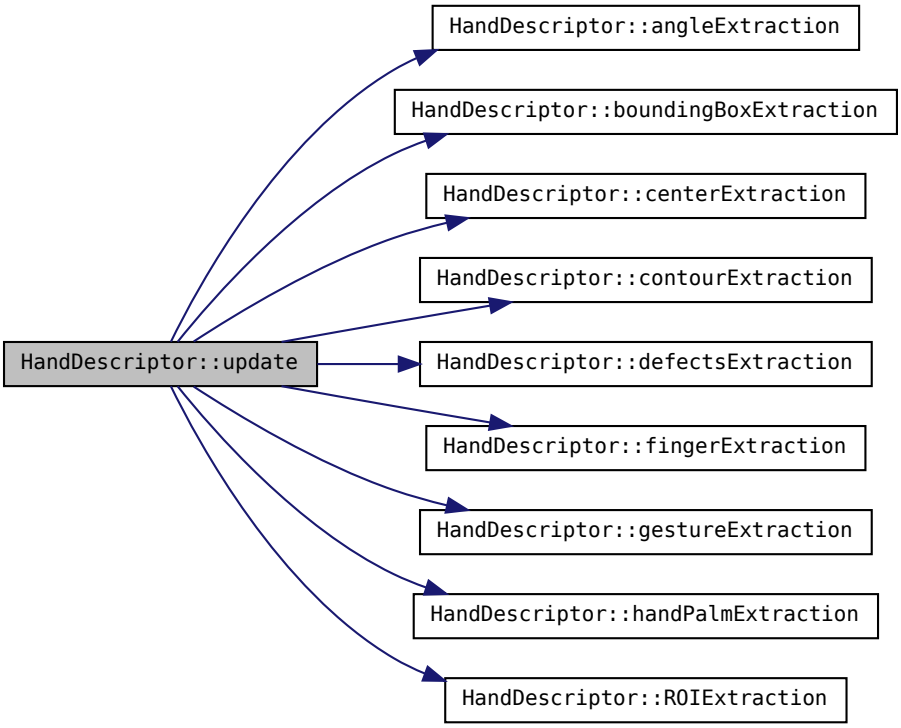
Update the internal characteristics stored.

Extracts all the hand characteristics and guesses the current gesture

Parameters

skinMask	Binary image containing the skin zones of hand candidates

Here is the call graph for this function:



2.4.4 Member Data Documentation

2.4.4.1 double HandDescriptor::_hand_angle [private]

Contains the actual angle of the box enclosing the hand.

2.4.4.2 double HandDescriptor::_hand_angle_estimation [private]

Contains the corrected estimation by the kalman filter.

2.4.4.3 double HandDescriptor::_hand_angle_prediction [private]

Contains the predicted angle by the kalman filter.

2.4.4.4 cv::Rect HandDescriptor::_hand_bounding_box [private]

Minimum Rect enclosing the hand.

2.4.4.5 cv::Point HandDescriptor::_hand_center [private]

Actual center of the hand.

2.4.4.6 cv::Point HandDescriptor::_hand_center_estimation [private]

Corrected estimation by the kalman filter.

2.4.4.7 cv::Point HandDescriptor::_hand_center_prediction [private]

Predicted center of the hand by the kalman filter.

2.4.4.8 std::vector< std::vector<cv::Point> > HandDescriptor::_hand_contour [private]

Contours of the candidates to be a hand.

2.4.4.9 std::vector< ConvexityDefect > HandDescriptor::_hand_convexity_defects [private]

Convexity defects of the hand.

2.4.4.10 std::vector< cv::Point > HandDescriptor::_hand_finger_line_origin [private]

Position of the finger line origin points.

2.4.4.11 std::vector< cv::Point > HandDescriptor::_hand_fingertips [private]

Position of the fingertips.

2.4.4.12 bool HandDescriptor::_hand_found [private]

Whether a hand was found or not:

2.4.4.13 int HandDescriptor::_hand_gesture [private]

Last detected gesture, coded as an integer (see constants for correspondence between integer and gesture)

2.4.4.14 std::vector< cv::Point > HandDescriptor::_hand_hull [private]

Complex hull of the hand.

2.4.4.15 int HandDescriptor::_hand_num_fingers [private]

Number of fingers (visible)

2.4.4.16 cv::Mat HandDescriptor::_hand_ROI [private]

Mask containing ROI of the hand.

2.4.4.17 cv::RotatedRect HandDescriptor::_hand_rotated_bounding_box [private]

Minimum RotatedRect enclosing the hand.

2.4.4.18 cv::Point HandDescriptor::_max_circle_incribed_center [private]

Center of the max. incribed circle.

2.4.4.19 double HandDescriptor::_max_circle_inscribed_radius [private]

Radius of the max. inscribed circle.

2.4.4.20 cv::Point2f HandDescriptor::_min_enclosing_circle_center [private]

Center of the min. enclosing circle.

2.4.4.21 float HandDescriptor::_min_enclosing_circle_radius [private]

Radius of the min. enclosing circle.

2.4.4.22 cv::KalmanFilter HandDescriptor::kalmanFilterAngle [private]

Kalman filter for the angle of the box enclosing the hand.

2.4.4.23 cv::KalmanFilter HandDescriptor::kalmanFilterCenter [private]

Kalman filter for the center of the hand.

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

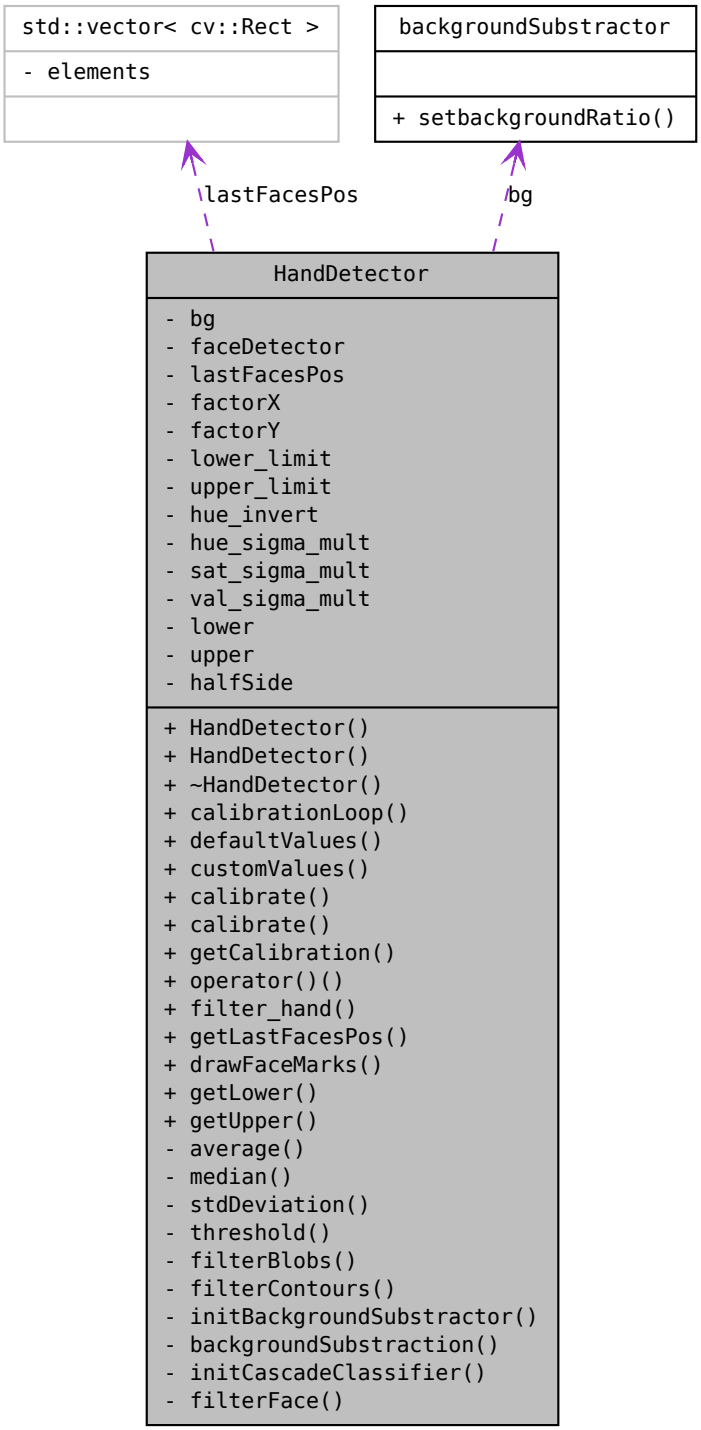
- HandDescriptor.h
- HandDescriptor.cpp

2.5 HandDetector Class Reference

Segments the hand silouette from the original image and returns the information in a binary image.

```
#include <HandDetector.h>
```

Collaboration diagram for HandDetector:



Public Member Functions

- **HandDetector** ()

  Default constructor.
- **HandDetector** (cv::Mat &ROI)

  Constructor that takes as argument an image of the hand's skin to obtain the custom HSV range.
- ∼**HandDetector** ()
- void **calibrationLoop** (cv::VideoCapture)

  Allows the user to adjust the HSV range manually to improve the segmentation.
- void **defaultValues** (cv::VideoCapture cap)

  Shows an image with information and starts calibrationLoop function.
- void **customValues** (cv::VideoCapture cap)

  Obtains the custom HSV skin range.
- void **calibrate** (cv::Mat &ROI)

  Changes the HSV range accordingly with the input skin color image.
- void **calibrate** (cv::Scalar **lower_limit**=cv::Scalar(0, 58, 89), cv::Scalar **upper_limit**=cv::Scalar(25, 173, 229))

  Sets the HSV skin colors range with the inputs. If there are no inputs, default values are set.
- void **getCalibration** (cv::Scalar &**lower_limit**, cv::Scalar &**upper_limit**)

  Returns the HSV range.
- void **operator()** (cv::Mat &src, cv::Mat &dst)

  Update the segmented hand binary image.
- void **filter_hand** (cv::Mat &src, cv::Mat &dst)

  Update the segmented hand image using the new frame of the video input.
- std::vector< cv::Rect > & **getLastFacesPos** ()

  Returns the last position of the face.
- void **drawFaceMarks** (const cv::Mat &src, cv::Mat &dst, cv::Scalar color=cv::Scalar(0, 255, 0), int thickness=1)

  Tracks and covers the faces that appear in the image.
- cv::Scalar **getLower** ()

  Returns the lower HSV skin values.
- cv::Scalar **getUpper** ()

  Returns the upper HSV skin values.

Private Member Functions

- int **average** (cv::Mat &ROI)

  Returns the average of the pixel's values.
- int **median** (cv::Mat &ROI)

  Returns the median of the pixel's values.
- int **stdDeviation** (cv::Mat &ROI)

  Returns the standard deviation of the pixel's values.

- void **threshold** (const cv::Mat &src, cv::Mat &dst)

  Thresholds the input image using the HSV range.
- void **filterBlobs** (const cv::Mat &src, cv::Mat &dst)

  Applies Gaussian Blur and thresholding to improve the final binary image.
- void **filterContours** (std::vector< std::vector< cv::Point > > &contours, std::vector< std::vector< cv::Point > > &filteredContours)
- void **initBackgroundSubstractor** ()

  Sets the parameters of the background subtractor object.
- void **backgroundSubstraction** (cv::Mat &src, cv::Mat &dst)

  Substracts the background from the input image.
- void **initCascadeClassifier** ()

  Initializes the face detector.
- void **filterFace** (const cv::Mat &src, cv::Mat &dstMask)

  Removes the face from the src image.

Private Attributes

- **backgroundSubstractor bg**

  Background Subtractor object that derives from cv::BackgroundSubtractorMOG2.
- cv::CascadeClassifier **faceDetector**

  Cascade classifier to detect faces:
- std::vector< cv::Rect > **lastFacesPos**

  Position of the last faces found:
- double **factorX**

  Resize the rectangles.
- double **factorY**
- cv::Scalar **lower_limit**

  Lower limit of the HSV skin range.
- cv::Scalar **upper_limit**

  Upper limit of the HSV skin range.
- bool **hue_invert**

  Boolean that will be true if color limit is arround 0.
- int **hue_sigma_mult**

  Hue sigma multiplier used when calculating the custom HSV skin range.
- int **sat_sigma_mult**

  Saturation sigma multiplier used when calculating the custom HSV skin range.
- int **val_sigma_mult**

  Value sigma multiplier used when calculating the custom HSV skin range.
- cv::Scalar **lower**

  Lower limit of the HSV skin range.
- cv::Scalar **upper**

  Upper limit of the HSV skin range.

Static Private Attributes

- static const int **halfSide** = 40

  Size of the calibration box used when capturing the custom HSV range.

2.5.1 Detailed Description

Segments the hand silouette from the original image and returns the information in a binary image.

This class offers different means of calibrating the skin: -Theoretical HSV values -Custom HSV values calculated from a sample of the user's skin color

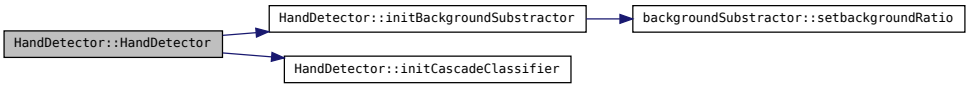
In order to obtain the segmented hand, the operator () or the function filter_hand may be used

2.5.2 Constructor & Destructor Documentation

2.5.2.1 HandDetector::HandDetector ( )

Default constructor.

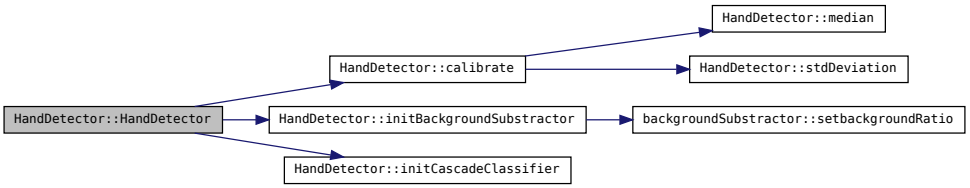
Here is the call graph for this function:



2.5.2.2 HandDetector::HandDetector (cv::Mat & *ROI*)

Constructor that takes as argument an image of the hand's skin to obtain the custom HSV range.

Here is the call graph for this function:



2.5.2.3 HandDetector::~HandDetector ( )

2.5.3 Member Function Documentation

2.5.3.1 int HandDetector::average (cv::Mat & *ROI*) [private]

Returns the average of the pixel's values.

2.5.3.2 void HandDetector::backgroundSubstraction (cv::Mat & *src*, cv::Mat & *dst*) [private]

Substracts the background from the input image.

Parameters

src	Input image
dst	Output image

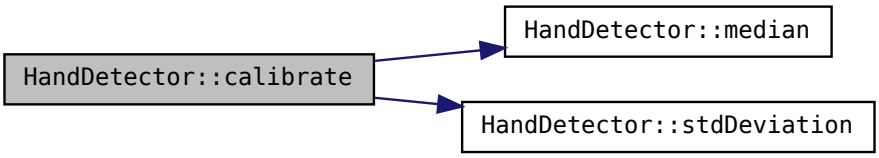
2.5.3.3 void HandDetector::calibrate (cv::Scalar *lower_limit* = cv::Scalar(0, 58, 89), cv::Scalar *upper_limit* = cv::Scalar(25, 173, 229))

Sets the HSV skin colors range with the inputs. If there are no inputs, default values are set.

2.5.3.4 void HandDetector::calibrate (cv::Mat & *ROI*)

Changes the HSV range accordingly with the input skin color image.

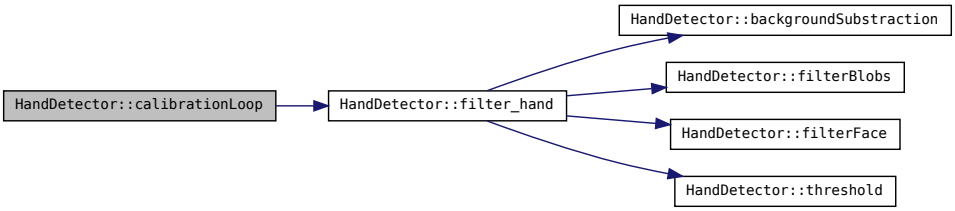
Here is the call graph for this function:



2.5.3.5 void HandDetector::calibrationLoop (cv::VideoCapture *cap*)

Allows the user to adjust the HSV range manually to improve the segmentation.

Here is the call graph for this function:

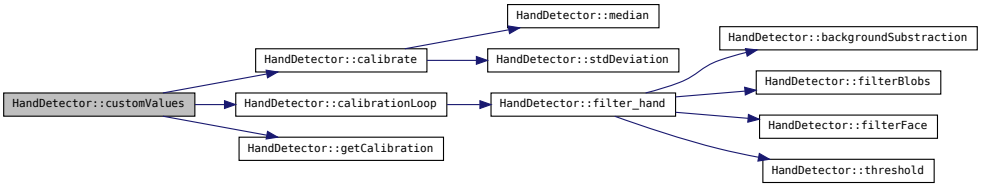


2.5.3.6 void HandDetector::customValues (cv::VideoCapture *cap*)

Obtains the custom HSV skin range.

Shows and image with information, then shows the calibration image and from the information obtained calculates the HSV skin color range. Finally, it calls the calibrationLoop function

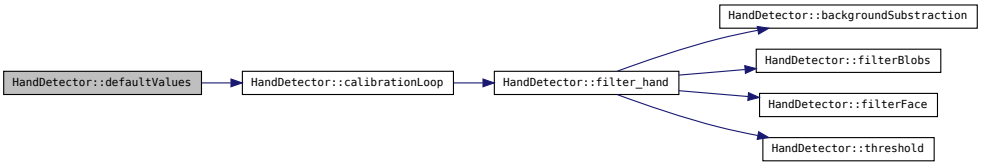
Here is the call graph for this function:



2.5.3.7 void HandDetector::defaultValues (cv::VideoCapture *cap*)

Shows an image with information and starts calibrationLoop function.

Here is the call graph for this function:



2.5.3.8 void HandDetector::drawFaceMarks (const cv::Mat & *src*, cv::Mat & *dst*, cv::Scalar *color* = `cv::Scalar(0, 255, 0)`, int *thickness* = `1`)

Tracks and covers the faces that appear in the image.

The faces are covered with a square so they do not interfere with the rest of the segmentation.

Parameters

src	Original image coming from the video input.
dst	Output image with the squares over the faces.
color	Color of the squares, default is black.
thickness	Thickness of the square drawn over the faces.

2.5.3.9 void HandDetector::filter_hand (cv::Mat & *src*, cv::Mat & *dst*)

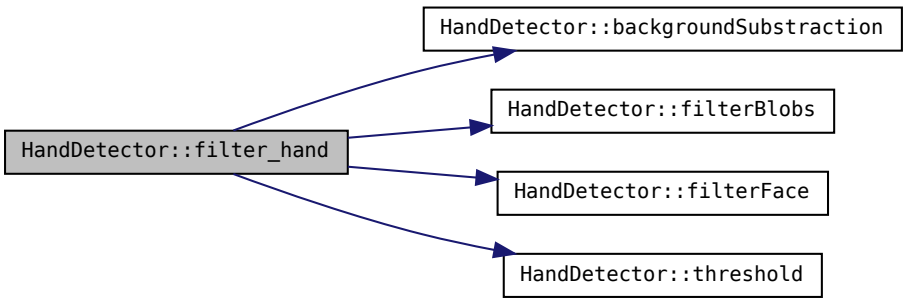
Update the segmented hand image using the new frame of the video input.

Removes the background, thresholds the skin color and makes morphology transformations to improve the binary output image

Parameters

src	Original image coming from the video input.
dst	Final binary image containing the segmented image.

Here is the call graph for this function:



2.5.3.10 void HandDetector::filterBlobs (const cv::Mat & *src*, cv::Mat & *dst*) [private]

Applies Gaussian Blur and thresholding to improve the final binary image.

Parameters

src	Input image
dst	Binary output image

2.5.3.11 void HandDetector::filterContours (std::vector< std::vector< cv::Point > > & *contours*, std::vector< std::vector< cv::Point > > & *filteredContours*) [private]

2.5.3.12 void HandDetector::filterFace (const cv::Mat & *src*, cv::Mat & *dstMask*) [private]

Removes the face from the src image.

--

2.5.3.13 void HandDetector::getCalibration (cv::Scalar & *lower_limit*, cv::Scalar & *upper_limit*)

Returns the HSV range.

2.5.3.14 std::vector< cv::Rect > & HandDetector::getLastFacesPos ( )

Returns the last position of the face.

2.5.3.15 cv::Scalar HandDetector::getLower ( )

Returns the lower HSV skin values.

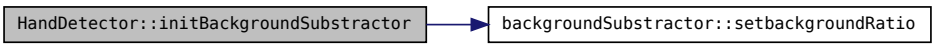
2.5.3.16 cv::Scalar HandDetector::getUpper ( )

Returns the upper HSV skin values.

2.5.3.17 void HandDetector::initBackgroundSubstractor ( ) [private]

Sets the parameters of the background subtractor object.

Here is the call graph for this function:



2.5.3.18 void HandDetector::initCascadeClassifier ( ) [private]

Initializes the face detector.

--

2.5.3.19 int HandDetector::median (cv::Mat & *ROI*) [private]

Returns the median of the pixel's values.

2.5.3.20 void HandDetector::operator() (cv::Mat & *src*, cv::Mat & *dst*)

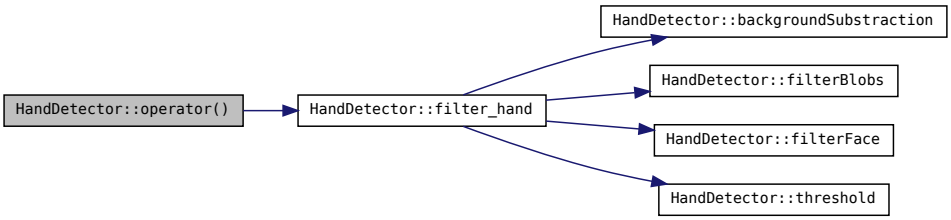
Update the segmented hand binary image.

This operator is a wrapper of the filter_hand function.

Parameters

src	Original image coming from the video input.
dst	Final binary image containing the segmented image.

Here is the call graph for this function:



2.5.3.21 int HandDetector::stdDeviation (cv::Mat & *ROI*) [private]

Returns the standard deviation of the pixel's values.

2.5.3.22 void HandDetector::threshold (const cv::Mat & *src*, cv::Mat & *dst*) [private]

Thresholds the input image using the HSV range.

Parameters

src	Input image
dst	Binary output image

2.5.4 Member Data Documentation

2.5.4.1 backgroundSubstractor HandDetector::bg [private]

Background Subtractor object that derives from cv::BackgroundSubtractorMOG2.

2.5.4.2 cv::CascadeClassifier HandDetector::faceDetector [private]

Cascade classifier to detect faces:

--

2.5.4.3 double HandDetector::factorX [private]

Resize the rectangles.

--

2.5.4.4 double HandDetector::factorY [private]

2.5.4.5 const int HandDetector::halfSide = 40 [static, private]

Size of the calibration box used when capturing the custom HSV range.

2.5.4.6 bool HandDetector::hue_invert [private]

Boolean that will be true if color limit is arround 0.

2.5.4.7 int HandDetector::hue_sigma_mult [private]

Hue sigma multiplier used when calculating the custom HSV skin range.

2.5.4.8 std::vector< cv::Rect > HandDetector::lastFacesPos [private]

Position of the last faces found:

--

2.5.4.9 cv::Scalar HandDetector::lower [private]

Lower limit of the HSV skin range.

2.5.4.10 cv::Scalar HandDetector::lower_limit [private]

Lower limit of the HSV skin range.

2.5.4.11 int HandDetector::sat_sigma_mult [private]

Saturation sigma multiplier used when calculating the custom HSV skin range.

2.5.4.12 cv::Scalar HandDetector::upper [private]

Upper limit of the HSV skin range.

2.5.4.13 cv::Scalar HandDetector::upper_limit [private]

Upper limit of the HSV skin range.

2.5.4.14 int HandDetector::val_sigma_mult [private]

Value sigma multiplier used when calculating the custom HSV skin range.

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

- HandDetector.h
- HandDetector.cpp

2.6 StateMachine Class Reference

Simple state machine for value tracking and event recognition.

`#include <StateMachine.h>`

Public Member Functions

- **StateMachine** (int **value_to_track**, unsigned int **min_positive_matches**, unsigned int **max_negative_matches**)

  StateMachine *(*p. **??***) constructor.*
- void **update** (int current_value)

  Update the state of the state machine according to the current value passed to it.
- void **reset** ()

  Reset the state of the state machine (0 positive matches and 0 negative matches)
- int **getValue_to_track** () const

  Returns the current value set to track.
- void **setValue_to_track** (const int &value)

  Changes the current value to track.
- float **getPercentageMatches** ()

  Returns the current number of positive matches as a percentage.
- bool **getFound** ()

  Gets the current state of the state machine.

Private Attributes

- int **value_to_track**

  Value to track.
- unsigned int **min_positive_matches**

  Minimum number of positive matches needed to consider a value found.
- unsigned int **current_positive_matches**

  Current number of positive matches.
- unsigned int **max_negative_matches**

  Maximum number of negative matches needed to reset the state/counter.
- unsigned int **current_negative_matches**

  Current number of negative matches.

- bool **found**

  State of the state machine, true if current number of positive matches is equal that the minimum needed.

2.6.1 Detailed Description

Simple state machine for value tracking and event recognition.

It has internally two counters, one counts the number of positive matches with the value to track, and the other the number of consecutive negative matches.

If the number of consecutive negative matches is equal to the maximum value allowed, it resets the positive matches counter. If the the number of positive matches equals the minimum value of positive matches needed, it sets the found variable true, until it is reseted (either by the user or by the arrival of negative matches).

2.6.2 Constructor & Destructor Documentation

2.6.2.1 StateMachine::StateMachine (int *value_to_track,* unsigned int *min_positive_matches,* unsigned int *max_negative_matches*)

StateMachine (p. **??**) constructor.

Parameters

value_to_track	Value to track for the positive matches
min_positive_matches	Minimum number of coincidences with the value to track before it considers it has been found
max_negative_matches	Maximum number of negative matches allowed between two positive matches

2.6.3 Member Function Documentation

2.6.3.1 bool StateMachine::getFound ( )

Gets the current state of the state machine.

2.6.3.2 float StateMachine::getPercentageMatches ( )

Returns the current number of positive matches as a percentage.

2.6.3.3 int StateMachine::getValue_to_track ( ) const

Returns the current value set to track.

2.6.3.4 void StateMachine::reset ( )

Reset the state of the state machine (0 positive matches and 0 negative matches)

2.6.3.5 void StateMachine::setValue_to_track (const int & *value*)

Changes the current value to track.

Here is the call graph for this function:

2.6.3.6 void StateMachine::update (int *current_value*)

Update the state of the state machine according to the current value passed to it.

Parameters

current_value	Current value to compare with the value to track

Here is the call graph for this function:

2.6.4 Member Data Documentation

2.6.4.1 unsigned int StateMachine::current_negative_matches [private]

Current number of negative matches.

2.6.4.2 unsigned int StateMachine::current_positive_matches [private]

Current number of positive matches.

2.6.4.3 bool StateMachine::found [private]

State of the state machine, true if current number of positive matches is equal that the minimum needed.

2.6.4.4 unsigned int StateMachine::max_negative_matches [private]

Maximum number of negative matches needed to reset the state/counter.

2.6.4.5 unsigned int StateMachine::min_positive_matches [private]

Minimum number of positive matches needed to consider a value found.

2.6.4.6 int StateMachine::value_to_track [private]

Value to track.

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

- StateMachine.h
- StateMachine.cpp