Kitchen Occupation

1.0

Generated by Doxygen 1.8.5

Fri Dec 13 2013 22:31:48

Contents

Chapter 1

Kichen Occupation

1.1 About

This software was created as a part of the course Bilder och Grafik (TSBB11) at Linköping University. The goal of the project was to create a software system for counting people and eventually having the system being used to measure room usage intensity at the university. The software is designed to be as modular and general as possible in order to allow for easy replacement of different system components.

1.1.1 Modular Software Architechture

The software has a modular and easily extendible design, where each module is responsible for a specific domain of the system. The architecture of the system is that of a pipeline of algorithms, where data pertaining to the latest few frames is piped through each pipeline step by the main program. The order is specified in a configuration file and set in place during program initialisation. The frame data consists of both raw sensor data and computed data from the different pipeline steps.

1.1.2 Configuration File System

The whole system can be configured by editing the plain text YAML configuration files. Among other things, it is possible to set what image processing algorithms should be run, in what order to run them and the value of most parameters. Many components in the Debug GUI, camera settings and important file paths are also set in the configuration files.

1.1.3 User Interface

The system includes a debugging and configuration GUI where the results from each of the process steps can be viewed in real time, and one step at a time. It can also be used to assist in tuning and calibrating the system, and is necessary for proper set up of a new sensor or room.

Chapter 2

Namespace Index

2.1 Namespace List

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

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

Namespace Documentation

5.1 configuration Namespace Reference

Namespace enveloping application settings.

Classes

- class ConfigurationManager
  Manages application settings.

5.1.1 Detailed Description

Namespace enveloping application settings.

5.2 debugging Namespace Reference

Debugging utilities.

Classes

- struct LogEntry
  A log entry is a container of a log information message.
- struct ProfilerEntry
  A profiler entry is a container of profiler information.
- class Logger
  Logger is a logging manager that is globally accessable under the alias 'logObject'.

Typedefs

- typedef std::deque< LogEntry >
  ::iterator LogIterator
  LogEntry iterator.

Variables

- Logger **logObject**

5.2.1 Detailed Description

Debugging utilities. Contains debugging and logging functionality, primarely a logging system and a profiler.

5.3 evaluation Namespace Reference

The evaluation namespace contains functionality for system evaluation.

Classes

- struct inOutEvent

  Describes how many people entered or exited the room in the current frame.
- class EntryExitEvaluator

  Evaluates system counting performance.
- class Evaluation

  Evaluates system performance.
- class TrackerEvaluator

  Tracker accuracy evaluator.

5.3.1 Detailed Description

The evaluation namespace contains functionality for system evaluation.

5.4 image_processing Namespace Reference

Image processing contains functionality for the different states of image processing required for human detection and tracking.

Classes

- class BackgroundModelMoG

  Class which creates binary image using OpenCV function BackgroundModelMoG2.
- class CircleDetection

  Process step which detects circles in the image that are meant to be indicative of the presence of human heads.
- class EntryExitCounter

  Process step which determines if objects are lost in an entry area, creates bounding boxes.
- class ForegroundRegionExtractorDefault

  Process step which does foreground modulation, creates bounding boxes.
- class ImageProcessor

  The Image Processor is the interface to the image processing functionality.
- class KinectSegmentation

  Require a depth image "rawImage" in the current frame for each camera, where darker is closer and black means undefined. Produces a vector of detected objects stored in the current frame for each camera.
- struct FlowVector

  Represent optical flow in a single point.
- class OpticalFlowSegmentation

  Computes the optical flow and does some basic segmentation based on it.
- class StereoBlockMatching

Process step which creates a depth map from a stereo image pair.

- class TrackingBruteForce

  Process step which tracks bounding boxes between frames.

Functions

- bool isInsidePolygon (cv::Mat mask, cv::Point2d point)

  Checks if a 2D-point is inside a polygon.

5.4.1 Detailed Description

Image processing contains functionality for the different states of image processing required for human detection and tracking.

5.4.2 Function Documentation

5.4.2.1 bool image_processing::isInsidePolygon (cv::Mat *mask*, cv::Point2d *point*)

Checks if a 2D-point is inside a polygon.

The polygon is specified as a cv matrix in the mask parameter

Parameters

mask	A cv::Mat with value 255 inside the polygon
point	A point to check if it is inside the polygon

Returns

true if inside

5.5 kinect Namespace Reference

The kinect namespace contains functionality for reading rgb and depth images from a kinect 360 device.

Classes

- struct KinectFrame

  Struct for handling output data created by the KinectHandler Class.

- class KinectHandler

  Handler for the Kinect sensors.

5.5.1 Detailed Description

The kinect namespace contains functionality for reading rgb and depth images from a kinect 360 device.

5.6 network Namespace Reference

The network namespace contains all system I/O functionality (sensors and web interface).

Classes

- class Network

  The Network class handles all system I/O.

5.6.1 Detailed Description

The network namespace contains all system I/O functionality (sensors and web interface).

5.7 statistics Namespace Reference

Statistical analysis.

Classes

- class Analytics

  The Analytics is the interface to statistical analysis of Frames.
- struct flowVectorPair

  Stuct which is used for a vector with pair values, people entered in a specific frame number.
- struct CameraFlow

  Stuct which is used for a vector with pair values.
- class FlowEstimator

  Process step which calculates the flow of pepole through the door.
- struct SplineStrip

  A cubic bezier spline.
- struct DirectedQueEdge

  A directed edge between objects in the queue graph.
- struct Que

  A queue of persons.
- class QueDetector

  Process step which uses information about visible persons' position and velocity and determines if a que is present or not.
- struct FrameQueData

  A struct the containing information from each frame needed to determine queue status over time.
- class QueSeverityEstimator

  Process step that aggregates queue visibility over time and uses this information to output a stable classification of the current queue severity.

5.7.1 Detailed Description

Statistical analysis. Estimates the number of people, models, flows etc.

Chapter 6

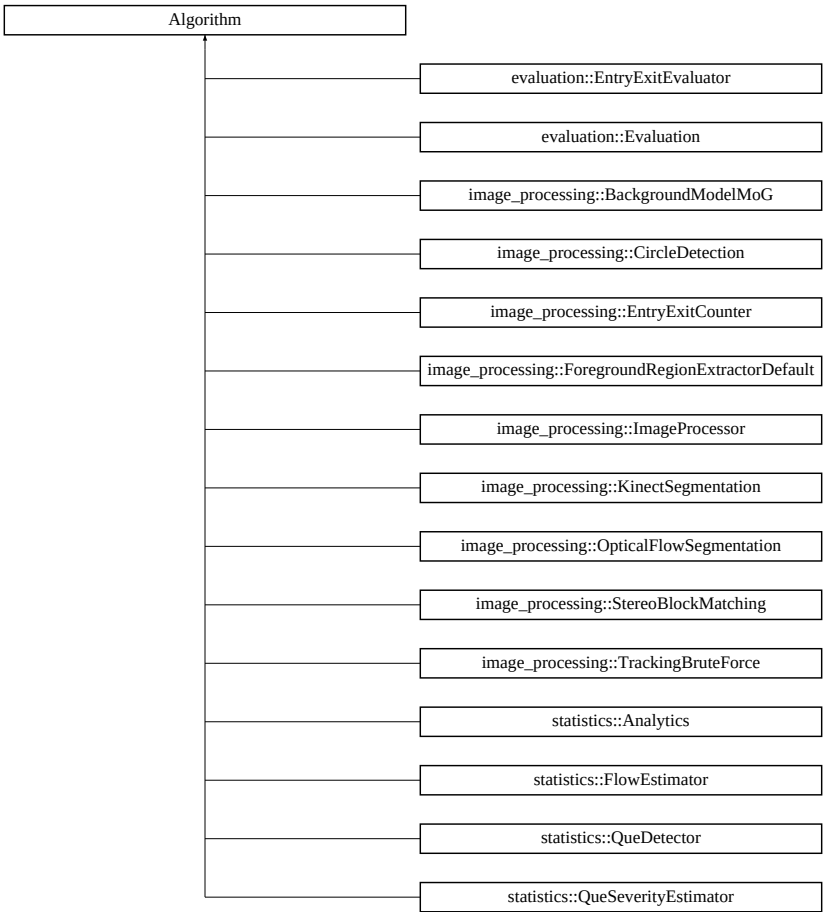
Class Documentation

6.1 Algorithm Class Reference

Base class for pipeline algorithms.

`#include <Algorithm.hpp>`

Inheritance diagram for Algorithm:



Public Member Functions

- Algorithm ()

Constructor.

- virtual ~Algorithm ()

  Destructor.
- virtual bool initialize (configuration::ConfigurationManager &settings)

  Initializes algorithm and all sub algorithms.
- bool populateSubAlgorithms (configuration::ConfigurationManager &settings, std::string algorithmName, AlgorithmFactory &algorithmFactory)

  Adds sub algorithms from settings, then calls this function for every sub algorithm.
- virtual void process (FrameList &frames)

  Processes all sub algorithms.
- void clearAlgorithms ()

  Remove all sub algorithms.

Protected Attributes

- bool isInitialized

  Is true if the algorithm is initialized.
- std::vector< Algorithm * > algorithms

  A vector of sub algorithms.
- std::vector< std::string > algorithmTag

  A vector of sub algorithms names, mapped in order to algorithms.

6.1.1 Detailed Description

Base class for pipeline algorithms.

Derive form this class to maintain a simple interface for all algorithms used in the processing pipeline.

6.1.2 Member Function Documentation

6.1.2.1 bool Algorithm::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initializes algorithm and all sub algorithms.

Parameters

settings	A ConfigurationManager object containing system settings.

Reimplemented in statistics::QueSeverityEstimator, image_processing::BackgroundModelMoG, image_processing::OpticalFlowSegmentation, image_processing::TrackingBruteForce, statistics::FlowEstimator, image_processing::KinectSegmentation, image_processing::EntryExitCounter, statistics::QueDetector, image_processing::CircleDetection, evaluation::Evaluation, image_processing::StereoBlockMatching, evaluation::EntryExitEvaluator, statistics::Analytics, image_processing::ImageProcessor, and image_processing::ForegroundRegionExtractorDefault.

6.1.2.2 bool Algorithm::populateSubAlgorithms (configuration::ConfigurationManager & *settings*, std::string *algorithmName*, AlgorithmFactory & *algorithmFactory*)

Adds sub algorithms from settings, then calls this function for every sub algorithm.

A sub algorithm is another Algorithm that is a part in this algorithms pipeline, meaning that it is initialized when this algorithm is initialized and it is executed when this algorithm is executed, both in the order specified in settings (the order they are stored in the algorithms vector).

Parameters

settings	A ConfigurationManager object containing system settings.
algorithmName	The name of the algortihm to be added to the pipeline.
algorithmFactory	The current pipeline part to be populated.

Returns

True if successful.

6.1.2.3 void Algorithm::process (FrameList & *frames*) [virtual]

Processes all sub algorithms.

Calls the process function on all populated sub algorithms in the order specified by the configuration file.

Parameters

frames	The current FrameList.

Reimplemented in statistics::QueSeverityEstimator, image_processing::TrackingBruteForce, statistics::QueDetector, image_processing::CircleDetection, statistics::Analytics, evaluation::EntryExitEvaluator, evaluation::Evaluation, image_processing::ImageProcessor, image_processing::OpticalFlowSegmentation, statistics::FlowEstimator, image_processing::ForegroundRegionExtractorDefault, image_processing::EntryExitCounter, image_processing::KinectSegmentation, image_processing::BackgroundModelMoG, and image_processing::StereoBlockMatching.

6.1.3 Member Data Documentation

6.1.3.1 std::vector<Algorithm *> Algorithm::algorithms [protected]

A vector of sub algorithms.

The sub algorithms will be initialized in order when initialize is called. They will also be processed in order when process is called, if process is not overriden.

6.1.3.2 bool Algorithm::isInitialized [protected]

Is true if the algorithm is initialized.

It should be set in the initialize method and may be modified by REQUIRE, if this macro is used. An algorithm which has isInitialized as false will not have its process method called.

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

- src/Utilities/Algorithm.hpp
- src/Utilities/Algorithm.cpp

6.2 AlgorithmFactory Class Reference

Algorithm Factory store available algorithms that can be added to the pipeline from config file.

`#include <AlgorithmFactory.hpp>`

Public Member Functions

- ~AlgorithmFactory ()

Destructor.

- bool has (std::string algorithmClassName)

  Queries wether an algorithm with a specific class name has been registered.
- Algorithm * get (std::string algorithmClassName)

  Get a pointer to a registered algorithm.
- void add (std::string algorithmClassName, Algorithm *algorithm)

  Add/register an algorithm.
- void clear ()

  Remove all algorithms and sub algorithms (delete).

6.2.1 Detailed Description

Algorithm Factory store available algorithms that can be added to the pipeline from config file.

Warning

Algorithms stored are currently shared, meaning that there exist only ONE algorithm of each type with the same name.
This should be improved with cloning of algorithms, since algorithms can contain member data which should not be shared between different instances in the pipeline of the same algorithm type.

6.2.2 Member Function Documentation

6.2.2.1 void AlgorithmFactory::add (std::string *algorithmClassName,* Algorithm * *algorithm*)

Add/register an algorithm.

Adds an algorithm with the specified name to this algorithm's list of sub algorithms.

Parameters

algorithmClassName	Name of the algorithm to be added.
algorithm	Pointer to the algorithm to be added.

6.2.2.2 Algorithm * AlgorithmFactory::get (std::string *algorithmClassName*)

Get a pointer to a registered algorithm.

Parameters

algorithmClassName	The name of registered algorithm.

Returns

Pointer to requested algorithm.

6.2.2.3 bool AlgorithmFactory::has (std::string *algorithmClassName*)

Queries wether an algorithm with a specific class name has been registered.

Parameters

algorithmClass-Name	The name of the algorithm

Returns

True if found.

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

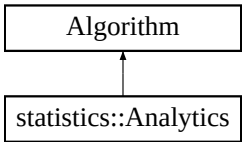
- src/Utilities/AlgorithmFactory.hpp
- src/Utilities/AlgorithmFactory.cpp

6.3 statistics::Analytics Class Reference

The Analytics is the interface to statistical analysis of Frames.

`#include <Analytics.hpp>`

Inheritance diagram for statistics::Analytics:



Public Member Functions

- Analytics ()

  Constructor.
- ~Analytics ()

  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Initialize the Analytics module.
- void reset ()

  Reset the analytics interface.
- void process (FrameList &frames)

  Perform all Statistics algorithms in order.

Additional Inherited Members

6.3.1 Detailed Description

The Analytics is the interface to statistical analysis of Frames.

Performs higher level analysis of the frame data, like estimating queue length etc

6.3.2 Member Function Documentation

6.3.2.1 bool statistics::Analytics::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initialize the Analytics module.

Configurates which algorithms to be applied in which order.

This algorithm acts as an interface to a set of subalgorithms and has no configurable parameters.

Parameters

settings	Configuration settings for all subalgorithms

Returns

True if successful.

Reimplemented from Algorithm.

6.3.2.2 void statistics::Analytics::process (FrameList & *frames*) [virtual]

Perform all Statistics algorithms in order.

Pass the frame list to each analytics algorithm in order

Parameters

frames	

Reimplemented from Algorithm.

6.3.2.3 void statistics::Analytics::reset ( )

Reset the analytics interface.

Not currently implemented

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

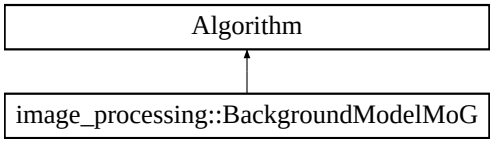
- src/Analytics/Analytics.hpp
- src/Analytics/Analytics.cpp

6.4 image_processing::BackgroundModelMoG Class Reference

Class which creates binary image using OpenCV function BackgroundModelMoG2.

`#include <BackgroundModelMoG.hpp>`

Inheritance diagram for image_processing::BackgroundModelMoG:



Public Member Functions

- BackgroundModelMoG ()
  Constructor.
- ~BackgroundModelMoG ()
  Destructor.
- void process (FrameList &frames)
  Performs the process step of the background-foreground segmentation.
- bool initialize (configuration::ConfigurationManager &conf)
  Initialize the algorithm.

Additional Inherited Members

6.4.1 Detailed Description

Class which creates binary image using OpenCV function BackgroundModelMoG2.

Z. Zivkovic. Improved adaptive Gausian mixture model for background subtraction. International Conference Pattern Recognition, UK, August, 2004.

6.4.2 Member Function Documentation

6.4.2.1 bool image_processing::BackgroundModelMoG::initialize (configuration::ConfigurationManager & *conf*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variables is not set.

Configurable algorithm parameters are:

- nMixtures: Toggles the maximum number of mixture models per pixel.
- backgroundRatio: Threshold defining whether the component is significant enough to be included into the background model or not. For alpha=0.001, it means that the mode should exist for approximately 105 frames before it is considered foreground.
- varThresholdGen: Threshold for the squared Mahalanobis distance that helps decide when a sample is close to the existing components.
- varThreshold: Threshold for the squared Mahalanobis distance that helps decide when a sample is close to the existing components.
- fVarInit: Initial variance for the newly generated components. It affects the speed of adaptation. The parameter value is based on your estimate of the typical standard deviation from the images.
- isShadowDetection: Parameter defining whether shadow detection should be enabled
- erotions: Number of erotions.
- dilations: Number of dilations.
- history: Length of the history.
- learningRate: Parameter which decides the learning rate. Defined between 0 and 1.
- downSamplingFactor: Downsampling parameter. Shirinnks the image's sides with 1/downSamplingFactor.

Returns

True if successful.

Reimplemented from Algorithm.

6.4.2.2 void image_processing::BackgroundModelMoG::process (FrameList & *frames*) [virtual]

Performs the process step of the background-foreground segmentation.

Creates a binary image, "foregroundMask", and performs erotions and dilations on the generated binary image.

Parameters

frames	Requires that each camera's latest frame contains a RGB-image called "rawImage".

Reimplemented from Algorithm.

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

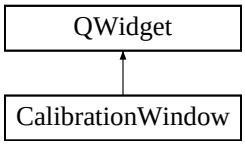
- src/ImageProcessing/BackgroundModelMoG.hpp
- src/ImageProcessing/BackgroundModelMoG.cpp

6.5 CalibrationWindow Class Reference

The CalibrationWindow class is a window where calibrates the threshold level for the system.

`#include <CalibrationWindow.hpp>`

Inheritance diagram for CalibrationWindow:



Public Slots

- void updateWindow (Frame currentFrame)

  updateWindow recievs a Frame and updates the windows accordingly.

Public Member Functions

- CalibrationWindow (QWidget *parent=0)

  Constructor.
- ~CalibrationWindow ()

  Destructor.
- void initialize (DenseKitchen *mainProgram)

  initialize sets up the CalibrationWindow

6.5.1 Detailed Description

The CalibrationWindow class is a window where calibrates the threshold level for the system.

This threshold level is used to adjust the system for the current installation height of the camera.

6.5.2 Member Function Documentation

6.5.2.1 void CalibrationWindow::initialize (DenseKitchen * *mainProgram*)

initialize sets up the CalibrationWindow

Parameters

mainProgram	is a pointer to the main program.

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

- src/Debugging/CalibrationWindow.hpp
- src/Debugging/CalibrationWindow.cpp

6.6 statistics::CameraFlow Struct Reference

Stuct which is used for a vector with pair values.

`#include <FlowEstimator.hpp>`

Public Attributes

- std::vector< flowVectorPair > **inFlow**
- std::vector< flowVectorPair > **outFlow**

6.6.1 Detailed Description

Stuct which is used for a vector with pair values.

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

- src/Analytics/FlowEstimator.hpp

6.7 CameraObject Class Reference

The CameraObject class represents a snapshot taken from a physical camera or similar sensor.

`#include <CameraObject.hpp>`

Public Member Functions

- CameraObject ()
  Constructor.
- ~CameraObject ()
  Destructor.
- void addImage (std::string tag, cv::Mat image)
  Add a new image to the camera stash.
- bool hasImage (std::string tag)
  Query if an image with the tag exist in the camera stash.
- cv::Mat getImage (std::string tag)
  Get an image from the camera stash.
- std::vector< Object > & getObjects ()
  Get a vector of the known objects.
- std::vector< Object > & getTransitionaryObjects ()
  Get a vector of the transitionary objects.
- std::vector< Object > & getNewlyFoundObjects ()
  Get a vector of the newly found elevated objects.

- std::vector< Object > & getPotentialObjects ()

  Get a vector of potential objects.
- std::map< std::string, cv::Mat > & getImages ()

  Get the map of cameraObject images.
- void setRoomID (std::string roomID)

  Set room ID.
- std::string & getRoomID ()

  Get room ID.
- void setEntered (int newEntered)

  Set total entered people.
- void setExited (int newExited)

  Set total exited peopple.
- int & getExited ()

  Get total number of exited people.
- int & getEntered ()

  Get total number of entered people.
- void setQueVisibility (bool queStatus)

  Set queue visibility.
- bool getQueVisibility ()

  Get queue visibility.

6.7.1 Detailed Description

The CameraObject class represents a snapshot taken from a physical camera or similar sensor.

This class contains intermediate steps of the image processing pipeline for this snapshot together with information about the number of people that has entered or exited in this frame as well as queue information.

6.7.2 Member Function Documentation

6.7.2.1 void CameraObject::addImage (std::string *tag*, cv::Mat *image*)

Add a new image to the camera stash.

Only one raw image exist, all other are processed images from different stages in the image processing pipeline.

Parameters

tag	Desired name of the image.
image	The image.

6.7.2.2 int& CameraObject::getEntered ( ) [inline]

Get total number of entered people.

Returns

The number of people that have entered the room so far.

6.7.2.3 int& CameraObject::getExited ( ) [inline]

Get total number of exited people.

Returns

The number of people that have exited the room so far.

6.7.2.4 cv::Mat CameraObject::getImage (std::string *tag*)

Get an image from the camera stash.

Parameters

tag	Image name

Returns

The image.

6.7.2.5 std::map<std::string, cv::Mat>& CameraObject::getImages ( ) [inline]

Get the map of cameraObject images.

Returns

A reference to the map containing all images.

6.7.2.6 std::vector<Object>& CameraObject::getNewlyFoundObjects ( ) [inline]

Get a vector of the newly found elevated objects.

Returns

A reference to the vector of newly found objects in the last frame.

6.7.2.7 std::vector<Object>& CameraObject::getObjects ( ) [inline]

Get a vector of the known objects.

Returns

A reference to the vector of known objects in the last frame.

6.7.2.8 std::vector<Object>& CameraObject::getPotentialObjects ( ) [inline]

Get a vector of potential objects.

Returns

A reference to the vector of potential objects in the last frame.

6.7.2.9 bool CameraObject::getQueVisibility ( ) [inline]

Get queue visibility.

Returns

True if queue is visible.

6.7.2.10 std::string& CameraObject::getRoomID ( ) [inline]

Get room ID.

Not currently in use since only single-room setups are supported.

Parameters

roomID	Desired room ID

6.7.2.11 std::vector<Object>& CameraObject::getTransitionaryObjects ( ) [inline]

Get a vector of the transitionary objects.

Returns

A reference to the vector of transitionary objects in the last frame.

6.7.2.12 bool CameraObject::hasImage (std::string *tag*)

Query if an image with the tag exist in the camera stash.

Parameters

tag	Name of requested image

Returns

True if the image exists

6.7.2.13 void CameraObject::setEntered (int *newEntered*) [inline]

Set total entered people.

Parameters

newEntered	New total number of people entered

6.7.2.14 void CameraObject::setExited (int *newExited*) [inline]

Set total exited peopple.

Parameters

newEntered	New total number of people exited

6.7.2.15 void CameraObject::setQueVisibility (bool *queStatus*) [inline]

Set queue visibility.

Parameters

queStatus	Status of the current queue

6.7.2.16 void CameraObject::setRoomID (std::string *roomID*) [inline]

Set room ID.

Not currently in use since only single-room setups are supported.

Parameters

roomID	Desired room ID

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

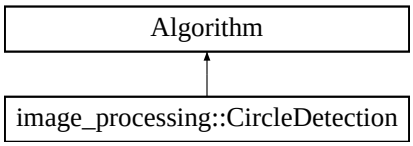
- src/Utilities/CameraObject.hpp
- src/Utilities/CameraObject.cpp

6.8 image_processing::CircleDetection Class Reference

Process step which detects circles in the image that are meant to be indicative of the presence of human heads.

`#include <CircleDetection.hpp>`

Inheritance diagram for image_processing::CircleDetection:



Public Member Functions

- CircleDetection ()

  Constructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Initialize the algorithm.
- void process (FrameList &frames)

  Uses Canny edge detector on each image channel.

Additional Inherited Members

6.8.1 Detailed Description

Process step which detects circles in the image that are meant to be indicative of the presence of human heads.

The approach is based on [Gardel, A.; Bravo, I.; Jimenez, P.; Lazaro, J.L.; Torquemada, A. "Real Time Head Detection for Embedded Vision Modules", Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on, On page(s): 1 - 6]. Although effective for simple cases we did not find the approach to be as usefull as hinted in the paper for more complex senarios. The algorithm class is therefore not completed.

6.8.2 Member Function Documentation

6.8.2.1 bool image_processing::CircleDetection::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variables is not set.

Configurable algorithm parameters are:

- lowThreshold: Lower threshold for the Canny edge detection algorithm.
- highThreshold: Upper threshold for the Canny edge detection algorithm.
- kernelSize: Kernel size for the smoothing step inte Canny algorithm.
- downSamplingFactor: x and y axis scale factor for image downsampling.
- averageCircleFilterSize: Average size of the hough circle filter.
- circleFilterRadiusDifference: Diversity of circle filter size.
- maskOutForeground: Toggle if background subtraction is to be used or not.
- detectionThreshold: Voting threshold. A voting score higher than detectionThreshold is needed to genereate a detection.

Returns

True if successful.

Reimplemented from Algorithm.

6.8.2.2 void image_processing::CircleDetection::process (FrameList & *frames*) [virtual]

Uses Canny edge detector on each image channel.

Combines them individual per-channel Canny output and convolves the result with a kernel designed to detect circes and ellipses. The result is thresholded and high enough values are used as person hypotheses.

Reimplemented from Algorithm.

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

- src/ImageProcessing/CircleDetection.hpp
- src/ImageProcessing/CircleDetection.cpp

6.9 configuration::ConfigurationManager Class Reference

Manages application settings.

`#include <ConfigurationManager.hpp>`

Public Member Functions

- ConfigurationManager ()

  Empty constructor.
- ~ConfigurationManager ()

  Destructor.
- bool readConfig (std::string filePath)

  Reads a configuration file.
- bool hasBool (std::string name)

  Checks availability of an int-type property.
- bool hasInt (std::string name)

  Checks availability of an int-type property.
- bool hasDouble (std::string name)

  Checks availability of a double-type property.
- bool hasString (std::string name)

  Checks availability of a string-type property.
- bool hasStringSeq (std::string name)

  Checks availability of a vector<string>-type property.
- bool getBool (std::string name)

  Gets value of bool with key "name".
- int getInt (std::string name)

  Gets value of int with key "name".
- double getDouble (std::string name)

  Gets value of double with key "name".
- std::string getString (std::string name)

  Gets value of string with key "name".
- std::vector< std::string > getStringSeq (std::string name)

  Gets the string sequence with key "name".
- void setBool (std::string name, bool value)

  Sets value of bool with key "name".
- void setInt (std::string name, int value)

  Sets value of int with key "name".
- void setDouble (std::string name, double value)

  Sets value of double with key "name".
- void setString (std::string name, std::string value)

  Sets value of string with key "name".
- void setStringSeq (std::string name, std::vector< std::string > value)

  Sets value of string sequence with key "name".
- bool configure (std::string name, bool &variable, bool defaultValue)

  Sets variable if it exists otherwise, sets it to devaultValue.
- bool configure (std::string name, int &variable, int defaultValue)

  Sets variable if it exists otherwise, sets it to devaultValue.
- bool configure (std::string name, double &variable, double defaultValue)

  Sets variable if it exists otherwise, sets it to devaultValue.
- bool configure (std::string name, std::string &variable, std::string defaultValue)

  Sets variable if it exists otherwise, sets it to devaultValue.
- bool writeToFile ()

  Writes all stored settings in the configuration manager to file.

6.9.1 Detailed Description

Manages application settings.

This class reads the configuration file and stores all settings as local variables. Data can be accessed by using the has and get functions, and data is added to the settings object by using the set functions. Finally all data can be written back to the file if desired.

6.9.2 Member Function Documentation

6.9.2.1 bool configuration::ConfigurationManager::configure (std::string *name,* bool & *variable,* bool *defaultValue*)

Sets variable if it exists otherwise, sets it to devaultValue.

Parameters

name	The name of the variable
variable	The variable value
variable	The default variable value

Returns

True if variable already existed, false if it was assigned to defaultValue.

6.9.2.2 bool configuration::ConfigurationManager::configure (std::string *name,* int & *variable,* int *defaultValue*)

Sets variable if it exists otherwise, sets it to devaultValue.

Parameters

name	The name of the variable
variable	The variable value
variable	The default variable value

Returns

True if variable already existed, false if it was assigned to defaultValue.

6.9.2.3 bool configuration::ConfigurationManager::configure (std::string *name,* double & *variable,* double *defaultValue*)

Sets variable if it exists otherwise, sets it to devaultValue.

Parameters

name	The name of the variable
variable	The variable value
variable	The default variable value

Returns

True if variable already existed, false if it was assigned to defaultValue.

6.9.2.4 bool configuration::ConfigurationManager::configure (std::string *name,* std::string & *variable,* std::string *defaultValue*)

Sets variable if it exists otherwise, sets it to devaultValue.

Parameters

name	The name of the variable
variable	The variable value
variable	The default variable value

Returns

True if variable already existed, false if it was assigned to defaultValue.

6.9.2.5 bool configuration::ConfigurationManager::getBool (std::string *name*)

Gets value of bool with key "name".

Parameters

name	Name of the bool variable in question.

Returns

Value of requested boolean.

6.9.2.6 double configuration::ConfigurationManager::getDouble (std::string *name*)

Gets value of double with key "name".

Parameters

name	Name of the double variable in question.

Returns

Value of requested double.

6.9.2.7 int configuration::ConfigurationManager::getInt (std::string *name*)

Gets value of int with key "name".

Parameters

name	Name of the int variable in question.

Returns

Value of requested int.

6.9.2.8 std::string configuration::ConfigurationManager::getString (std::string *name*)

Gets value of string with key "name".

Parameters

name	Name of the string variable in question.

Returns

Value of requested string.

6.9.2.9 std::vector< std::string > configuration::ConfigurationManager::getStringSeq (std::string *name*)

Gets the string sequence with key "name".

Parameters

name	Name of the string sequence in question.

Returns

Requesed string vector.

6.9.2.10 bool configuration::ConfigurationManager::hasBool (std::string *name*)

Checks availability of an int-type property.

Parameters

name	Name of requested property.

Returns

Returns true if specified property exists.

6.9.2.11 bool configuration::ConfigurationManager::hasDouble (std::string *name*)

Checks availability of a double-type property.

Parameters

name	Name of requested property.

Returns

Returns true if specified property exists.

6.9.2.12 bool configuration::ConfigurationManager::hasInt (std::string *name*)

Checks availability of an int-type property.

Parameters

name	Name of requested property.

Returns

Returns true if specified property exists.

6.9.2.13 bool configuration::ConfigurationManager::hasString (std::string *name*)

Checks availability of a string-type property.

Parameters

item	Name of requested property.

Returns

Returns true if specified property exists.

6.9.2.14 bool configuration::ConfigurationManager::hasStringSeq (std::string *name*)

Checks availability of a vector<string>-type property.

Parameters

item	Name of requested property.

Returns

Returns true if specified property exists.

6.9.2.15 bool configuration::ConfigurationManager::readConfig (std::string *filePath*)

Reads a configuration file.

The data that is read from the file is stored in local std::maps within this class.

Parameters

filePath	Location of configuration file.

Returns

Returns true if successful.

6.9.2.16 void configuration::ConfigurationManager::setBool (std::string *name,* bool *value*)

Sets value of bool with key "name".

Parameters

name	Name of the bool variable in question.
value	New value of the bool variable in question.

6.9.2.17 void configuration::ConfigurationManager::setDouble (std::string *name,* double *value*)

Sets value of double with key "name".

Parameters

name	Name of the double variable in question.
value	New value of the double variable in question.

6.9.2.18 void configuration::ConfigurationManager::setInt (std::string *name,* int *value*)

Sets value of int with key "name".

Parameters

name	Name of the int variable in question.
value	New value of the int variable in question.

6.9.2.19 void configuration::ConfigurationManager::setString (std::string *name*, std::string *value*)

Sets value of string with key "name".

Parameters

name	Name of the string variable in question.
value	New value of the string variable in question.

6.9.2.20 void configuration::ConfigurationManager::setStringSeq (std::string *name*, std::vector< std::string > *value*)

Sets value of string sequence with key "name".

Parameters

name	Name of the string sequence variable in question.
value	New value of the string sequence in question.

6.9.2.21 bool configuration::ConfigurationManager::writeToFile ( )

Writes all stored settings in the configuration manager to file.

Parameters

filePath	Location and name of output file.

Returns

Returns true if successful.

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

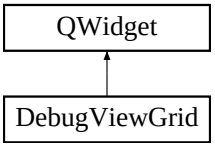
- src/Configuration/ConfigurationManager.hpp
- src/Configuration/ConfigurationManager.cpp

6.10 DebugViewGrid Class Reference

The DebugViewGrid class is a container for DebugViewWidgets.

`#include <DebugViewGrid.hpp>`

Inheritance diagram for DebugViewGrid:



Public Member Functions

- DebugViewGrid (QWidget *parent=0)

  Constructor.
- ~DebugViewGrid ()

  Destructor.
- void initialize (int nColumns)

  Initialize takes the desired number of columns in the DebugViewGrid and sets up the auto-adaption.
- void addWidget (DebugViewWidget *widget)

  Inserts an DebugViewWidget in the grid.
- void clearGrid ()

  Removes everything that is currently in the grid.

6.10.1 Detailed Description

The DebugViewGrid class is a container for DebugViewWidgets.

Processs steps that are selected and poped from the MainDebugWindow will end up here. An arbitrarilly number of widgets can be pop to the grid and these are automatically arranged and resized to fit the current grid size.

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

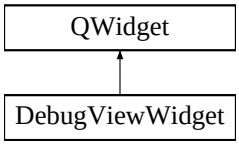
- src/Debugging/DebugViewGrid.hpp
- src/Debugging/DebugViewGrid.cpp

6.11 DebugViewWidget Class Reference

The DebugViewWidget class if simply a container for a cv::Mat representing a certain step in the preocess chain.

`#include <DebugViewWidget.hpp>`

Inheritance diagram for DebugViewWidget:



Public Slots

- void updateView (Frame)

  updateView receivs a Frame, stores it in the widget and displays the new one.

Signals

- void **aboutToClose** (std::string)

Public Member Functions

- DebugViewWidget (QWidget *parent=0)

  Constructor.

- ~DebugViewWidget ()

  Destructor.
- void initialize (const std::string processStepName, int camNumber)

  Sets up the widget, adding a label with the represented preocesses step and camera.
- void showImage ()

  Converts the cv::Mat into a QImage and displays it.
- std::string getIdentifier ()

  Returns the process step and camera of the widget.

6.11.1 Detailed Description

The DebugViewWidget class if simply a container for a cv::Mat representing a certain step in the preocess chain.

6.11.2 Member Function Documentation

6.11.2.1 void DebugViewWidget::initialize (const std::string *processStepName*, int *camNumber*)

Sets up the widget, adding a label with the represented preocesses step and camera.

Parameters

processStepName	is the name of a debug image set by the developer.
camNumber	is number of the camera that the process step belongs to.

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

- src/Debugging/DebugViewWidget.hpp
- src/Debugging/DebugViewWidget.cpp

6.12 DenseKitchen Class Reference

Main program class.

`#include <DenseKitchen.hpp>`

Public Member Functions

- DenseKitchen ()

  Constructor.
- ~DenseKitchen ()

  Destructor.
- bool initialize (std::string path)

  Initialize the program using a specified configuration file.
- void reset ()

  Reset program completely.
- bool singleIteration ()

  Run one iteration of the program.
- FrameList * getFrames ()

  Get the current FrameList.
- configuration::ConfigurationManager * getSettings ()

  Get the current settings.

6.12.1 Detailed Description

Main program class.

This class provides the interface for people detection and counting engine. Run in order: initialize() once and then, singleIteration() as many times as wished.

6.12.2 Member Function Documentation

6.12.2.1 FrameList * DenseKitchen::getFrames ( )

Get the current FrameList.

Returns

A pointer to the active FrameList.

6.12.2.2 configuration::ConfigurationManager * DenseKitchen::getSettings ( )

Get the current settings.

Returns

A pointer to the active settings.

6.12.2.3 bool DenseKitchen::initialize (std::string *path*)

Initialize the program using a specified configuration file.

Loads system settings and configures all the different program modules.

Parameters

path	Path to the configuration file

Returns

Returns false if any of its modules fail.

6.12.2.4 void DenseKitchen::reset ( )

Reset program completely.

Clears all temporary system settings and variables.

6.12.2.5 bool DenseKitchen::singleIteration ( )

Run one iteration of the program.

Deque one frame, perform person tracking and update statistics.

Returns

False if the program wants to terminate, otherwise True.

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

- src/DenseKitchen.hpp
- src/DenseKitchen.cpp

6.13 statistics::DirectedQueEdge Struct Reference

A directed edge between objects in the queue graph.

`#include <Que.hpp>`

Public Member Functions

- DirectedQueEdge (Object fromObj, Object toObj, std::vector< SplineStrip > splineStrips, float dist)

  Constructor.
- DirectedQueEdge (Object fromObj)

  Constructor.

Public Attributes

- Object from

  Object that edge starts from.
- Object to

  Object that edge ends at.
- std::vector< SplineStrip > spline

  The spline that draws the path between them.
- double distance

  The distance between the two objects.

6.13.1 Detailed Description

A directed edge between objects in the queue graph.

The cost/length of an edge is determined by the length of the spline from the first object/person to the last.

6.13.2 Member Data Documentation

6.13.2.1 double statistics::DirectedQueEdge::distance

The distance between the two objects.

The distance is measured as the length of the connecting spline.

6.13.2.2 std::vector<SplineStrip> statistics::DirectedQueEdge::spline

The spline that draws the path between them.

A container holding all the splinestrips resulting from subdivision of the original spline representation. These are not necessarily in order.

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

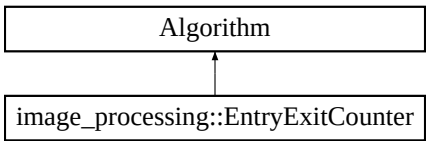
- src/Analytics/Que.hpp

6.14 image_processing::EntryExitCounter Class Reference

Process step which determines if objects are lost in an entry area, creates bounding boxes.

`#include <EntryExitCounter.hpp>`

Inheritance diagram for image_processing::EntryExitCounter:



Public Member Functions

- EntryExitCounter ()

  Constructor.
- ~EntryExitCounter ()

  Destructor.
- void process (FrameList &frames)

  Performs the process step, counts people.
- bool initialize (configuration::ConfigurationManager &conf)

  Initialize the algorithm.

Additional Inherited Members

6.14.1 Detailed Description

Process step which determines if objects are lost in an entry area, creates bounding boxes.

Registers if objects enters and exits the room the objects has to fulfill some requirements. To be considered as entered, an object has to be created for the first time in the set door area and pass the three circle lines and be elevated to a real object. To be registered as exited an object has to be a real object and disappear inside the door area, while also at least once passed the three lines.

6.14.2 Member Function Documentation

6.14.2.1 bool image_processing::EntryExitCounter::initialize (configuration::ConfigurationManager & *conf*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set.

This algorithm has no configurable parameters.

Reimplemented from Algorithm.

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

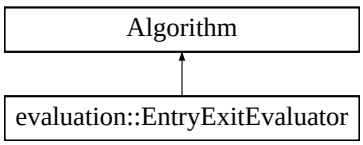
- src/ImageProcessing/EntryExitCounter.hpp
- src/ImageProcessing/EntryExitCounter.cpp

6.15 evaluation::EntryExitEvaluator Class Reference

Evaluates system counting performance.

`#include <EntryExitEvaluator.hpp>`

Inheritance diagram for evaluation::EntryExitEvaluator:



Public Member Functions

- EntryExitEvaluator ()

  Constructor.
- ~EntryExitEvaluator ()

  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Initializes the entryexitevaluation module.
- void process (FrameList &frames)

  Evaluates and updates the results for the system.
- void printToLog (unsigned int cameraIndex)

  Prints entry/exit accuracy information to the debug log.

Additional Inherited Members

6.15.1 Detailed Description

Evaluates system counting performance.

System performance is evaluated here by comparing the total number of people that have exited the room to a pre-recorded ground truth file.

6.15.2 Member Function Documentation

6.15.2.1 bool evaluation::EntryExitEvaluator::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initializes the entryexitevaluation module.

The module is intitalized by loading all ground truth from files specified by the configurationManager object

Parameters

settings	Configuration-object containing the location of the ground truth files and other relevant settings.

Returns

Returns true if successful, otherwise false.

Reimplemented from Algorithm.

6.15.2.2 void evaluation::EntryExitEvaluator::printToLog (unsigned int *cameraIndex*)

Prints entry/exit accuracy information to the debug log.

Parameters

cameraIndex	Index of the camera where information is located.

6.15.2.3 void evaluation::EntryExitEvaluator::process (FrameList & *frames*) [virtual]

Evaluates and updates the results for the system.

Is called upon after each iteration in order to calculate the difference between the systems entry/exit estimate and the ground truth.

Parameters

frames	The Frames to be evaluated.

Reimplemented from Algorithm.

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

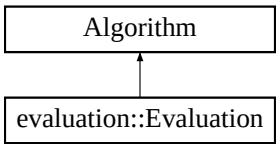
- src/Evaluation/EntryExitEvaluator.hpp
- src/Evaluation/EntryExitEvaluator.cpp

6.16 evaluation::Evaluation Class Reference

Evaluates system performance.

`#include <Evaluation.hpp>`

Inheritance diagram for evaluation::Evaluation:



Public Member Functions

- Evaluation ()
  Empty constructor.
- ~Evaluation ()
  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)
  Initializes the evaluation module.
- void process (FrameList &frames)
  Evaluates and updates the results for the system.
- void printToLog ()
  Print stats to the debug log.

Additional Inherited Members

6.16.1 Detailed Description

Evaluates system performance.

System performance is evaluated by comparing system output to pre-labeled ground truth data. How performanced is measured is specified by the subalgorithm who inherit this class. Whhich sub algorithms that are to be used is specified in a configurationManager object that is passed to the initialize function.

6.16.2 Member Function Documentation

6.16.2.1 bool evaluation::Evaluation::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initializes the evaluation module.

Parameters

settings	Configuration-object containing the location of the ground truth files.

Returns

True if successful.

Reimplemented from Algorithm.

6.16.2.2 void evaluation::Evaluation::process (FrameList & *frames*) [virtual]

Evaluates and updates the results for the system.

Is called upon after each iteration in order to calculate the different performance metrics by comparing the tracker system output to ground truth.

Parameters

frameList	The FrameList to be evaluated.

Reimplemented from Algorithm.

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

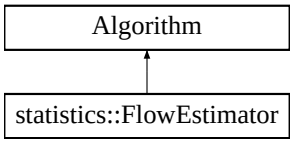
- src/Evaluation/Evaluation.hpp
- src/Evaluation/Evaluation.cpp

6.17 statistics::FlowEstimator Class Reference

Process step which calculates the flow of pepole through the door.

`#include <FlowEstimator.hpp>`

Inheritance diagram for statistics::FlowEstimator:



Public Member Functions

- FlowEstimator ()

  Constructor.
- ~FlowEstimator ()

Destructor.

- void process (FrameList &frames)

  Performs the process step.

- bool initialize (configuration::ConfigurationManager &conf)

  Initialize the algorithm.

Additional Inherited Members

6.17.1 Detailed Description

Process step which calculates the flow of pepole through the door.

6.17.2 Member Function Documentation

6.17.2.1 bool statistics::FlowEstimator::initialize (configuration::ConfigurationManager & *conf*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set.

This algorithm has no configurable parameters

Reimplemented from Algorithm.

6.17.2.2 void statistics::FlowEstimator::process (FrameList & *frames*) [virtual]

Performs the process step.

Compute flow of detected objects passing trough the door as pepole per frame

Reimplemented from Algorithm.

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

- src/Analytics/FlowEstimator.hpp
- src/Analytics/FlowEstimator.cpp

6.18 image_processing::FlowVector Struct Reference

Represent optical flow in a single point.

`#include <OpticalFlowSegmentation.hpp>`

Public Attributes

- cv::Point2f **pos**
- cv::Point2f **flow**

6.18.1 Detailed Description

Represent optical flow in a single point.

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

- src/ImageProcessing/OpticalFlowSegmentation.hpp

6.19 statistics::flowVectorPair Struct Reference

Stuct which is used for a vector with pair values, people entered in a specific frame number.

`#include <FlowEstimator.hpp>`

Public Member Functions

- **flowVectorPair** (int _flow, unsigned int _frameCount)

Public Attributes

- int **flow**
- unsigned int **frameCount**

6.19.1 Detailed Description

Stuct which is used for a vector with pair values, people entered in a specific frame number.

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

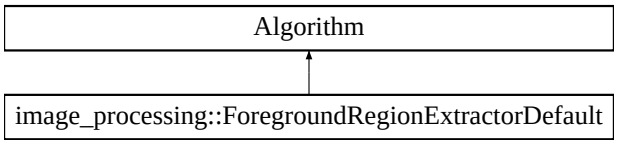
- src/Analytics/FlowEstimator.hpp
- src/Analytics/FlowEstimator.cpp

6.20 image_processing::ForegroundRegionExtractorDefault Class Reference

Process step which does foreground modulation, creates bounding boxes.

`#include <ForegroundRegionExtractorDefault.hpp>`

Inheritance diagram for image_processing::ForegroundRegionExtractorDefault:



Public Member Functions

- ForegroundRegionExtractorDefault ()

  Constructor.
- ~ForegroundRegionExtractorDefault ()

  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Initialize the algorithm.
- void process (FrameList &frames)

  Performs the foreground modulation.

Additional Inherited Members

6.20.1 Detailed Description

Process step which does foreground modulation, creates bounding boxes.

Creates bounding boxes in a binary image using OpenCV's built in functions findContours and boundingRect.

6.20.2 Member Function Documentation

6.20.2.1 bool image_processing::ForegroundRegionExtractorDefault::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set in the config file.

Configurable algorithm parameters are:

- minimalArea: Sets the minimal area for a bounding box.

Returns

True if successful.

Reimplemented from Algorithm.

6.20.2.2 void image_processing::ForegroundRegionExtractorDefault::process (FrameList & *frames*) [virtual]

Performs the foreground modulation.

Process step which does foreground modulation, creates bounding boxes

Reimplemented from Algorithm.

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

- src/ImageProcessing/ForegroundRegionExtractorDefault.hpp
- src/ImageProcessing/ForegroundRegionExtractorDefault.cpp

6.21 Frame Class Reference

A container of a snap shot of a discrete time step.

`#include <Frame.hpp>`

Public Member Functions

- Frame ()
  Constructor.
- ~Frame ()
  Destructor.
- void addCamera (CameraObject c)
  Adds a CameraObject to the frame.
- std::vector< CameraObject > & getCameras ()
  Returns all camera objects.

- std::vector< cv::Mat > getRoomImages (std::string roomID)

  Returns a vector of all raw images from all cameras in a room.
- std::vector< std::string > getRoomIDs ()

  Returns the room IDs for all rooms.
- double getMomentaryFps () const

  Get the momentary FPS value.
- void setMomentaryFps (double value)

  Set the momentary FPS variable.
- void initRoomPopulations (std::vector< CameraObject > &_cameras)

  Initializes the population of all rooms (sets to zero).
- void setPopulationInRoomID (int _newVal, std::string &_currID)

  Set the room population in a room.
- int getPopulationInRoomID (std::string &_currID)

  Get the number of people in a room.
- void setQueStatus (int newQueStatus)

  Set the the status of "queueness" estimated to this frame.
- int getQueStatus ()

  Get the the status of "queueness" estimated to this frame.

6.21.1 Detailed Description

A container of a snap shot of a discrete time step.

Contains raw images from all camreas taken at the same time. Contains persons seen. Contains all intermediate debuging data.

6.21.2 Member Function Documentation

6.21.2.1 void Frame::addCamera (CameraObject *c*)

Adds a CameraObject to the frame.

Parameters

c	A CameraObject, representing a physical camera.

6.21.2.2 std::vector< CameraObject > & Frame::getCameras ( )

Returns all camera objects.

Returns

Reference to a vector of CameraObjects, each representing a physical camera.

6.21.2.3 double Frame::getMomentaryFps ( ) const

Get the momentary FPS value.

Returns

1/t where t is the time (in seconds) spent since previous call to Network::dequeFrame.

6.21.2.4 int Frame::getPopulationInRoomID (std::string & *_currID*)

Get the number of people in a room.

Parameters

_currID	ID of the room.

Returns

Number of people currently in the room.

6.21.2.5 int Frame::getQueStatus ( )

Get the the status of "queueness" estimated to this frame.

Returns

Current "queueness".

6.21.2.6 std::vector< std::string > Frame::getRoomIDs ( )

Returns the room IDs for all rooms.

Returns

A vector containing all room names.

6.21.2.7 std::vector< cv::Mat > Frame::getRoomImages (std::string *roomID*)

Returns a vector of all raw images from all cameras in a room.

Parameters

roomID	The room ID.

Returns

Vector with all unprocessed images.

6.21.2.8 void Frame::initRoomPopulations (std::vector< **CameraObject** > & *_cameras*)

Initializes the population of all rooms (sets to zero).

Parameters

_cameras	A vector of cameraObjects representing all physical sensors.

6.21.2.9 void Frame::setMomentaryFps (double *value*)

Set the momentary FPS variable.

Parameters

value	New value.

6.21.2.10 void Frame::setPopulationInRoomID (int *_newVal*, std::string & *_currID*)

Set the room population in a room.

Parameters

_newVal	Desired new value
_currID	ID of the room whose value is to be changed.

6.21.2.11 void Frame::setQueStatus (int *newQueStatus*)

Set the the status of "queueness" estimated to this frame.

Parameters

newQueStatus	Desired "queueness" value.

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

- src/Utilities/Frame.hpp
- src/Utilities/Frame.cpp

6.22 FrameList Class Reference

A container of cronoligicaly ordered Frames.

`#include <FrameList.hpp>`

Public Member Functions

- FrameList (int framesToKeep=10)
  Constructor.
- ~FrameList ()
  Destructor.
- Frame & getCurrent ()
  Get the current (latest) Frame.
- Frame & getPrevious ()
  Get the previous (previously latest) Frame.
- bool hasPrevious ()
  Weather there are two or more frames.
- void append (Frame newFrame)
  Append the FrameList with the latest Frame.
- int size ()
  Get the number of frames in history.
- int getFrameCount ()
  Get the current frame counter.
- cv::Mat getExclusionMask () const
  Returns a matrix the binary exclusion mask.
- void setExclusionMask (const cv::Mat &value)

Defines the exclusion mask (an area of the image set to constant zero).

- bool hasExclusionMask ()

  Query if an exclusion mask exists.
- cv::Mat getDoorMask () const

  Get the doorway.
- void setDoorMask (const cv::Mat &value)

  Defines the doorway.
- bool hasDoorMask ()

  Query if a door mask exists.
- cv::Mat getInclusionMask () const

  Returns the inverse of the exclusion mask.
- void setInclusionMask (const cv::Mat &value)

  Sets the inclusion mask.
- bool hasInclusionMask ()

  Query if an inclusion mask exists.
- bool hasCheckPointMasks ()

  Query if checkpoint masks are set.
- cv::Mat getCheckPointMaskSmall () const

  Returns the smallest checkpoint mask.
- void setCheckPointMaskSmall (const cv::Mat &value)

  Returns the smallest checkpoint mask.
- cv::Mat getCheckPointMaskMedium () const

  Returns the smallest checkpoint mask.
- void setCheckPointMaskMedium (const cv::Mat &value)

  Returns the smallest checkpoint mask.
- cv::Mat getCheckPointMaskLarge () const

  Returns the smallest checkpoint mask.
- void setCheckPointMaskLarge (const cv::Mat &value)

  Returns the smallest checkpoint mask.

6.22.1 Detailed Description

A container of cronoligicaly ordered Frames.

Keeps infinitly or a configurable number of Frames as history apart from the current Frame.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 FrameList::FrameList (int *framesToKeep* = `10`)

Constructor.

Parameters

framesToKeep	The last framesToKeep frames are saved in the FrameList object.

6.22.3 Member Function Documentation

6.22.3.1 cv::Mat FrameList::getCheckPointMaskLarge ( ) const

Returns the smallest checkpoint mask.

Returns

A binary image containing the large checkpoint mask.

6.22.3.2 cv::Mat FrameList::getCheckPointMaskMedium ( ) const

Returns the smallest checkpoint mask.

Returns

A binary image containing the medium checkpoint mask.

6.22.3.3 cv::Mat FrameList::getCheckPointMaskSmall ( ) const

Returns the smallest checkpoint mask.

Returns

A binary image containing the small checkpoint mask.

6.22.3.4 cv::Mat FrameList::getDoorMask ( ) const

Get the doorway.

Returns

A binary mask defining the doorway.

6.22.3.5 cv::Mat FrameList::getInclusionMask ( ) const

Returns the inverse of the exclusion mask.

Returns

The inclusionMask.

6.22.3.6 bool FrameList::hasCheckPointMasks ( )

Query if checkpoint masks are set.

Returns

True if checkpoint masks exist.

6.22.3.7 bool FrameList::hasDoorMask ( )

Query if a door mask exists.

Returns

True if exists

6.22.3.8 bool FrameList::hasExclusionMask ( )

Query if an exclusion mask exists.

Returns

True if exists

6.22.3.9 bool FrameList::hasInclusionMask ( )

Query if an inclusion mask exists.

Returns

True if an inclusion mask exists.

6.22.3.10 void FrameList::setCheckPointMaskLarge (const cv::Mat & *value*)

Returns the smallest checkpoint mask.

Parameters

value	A binary image containing the large checkpoint mask.

6.22.3.11 void FrameList::setCheckPointMaskMedium (const cv::Mat & *value*)

Returns the smallest checkpoint mask.

Parameters

A	binary image containing the medium checkpoint mask.

6.22.3.12 void FrameList::setCheckPointMaskSmall (const cv::Mat & *value*)

Returns the smallest checkpoint mask.

Parameters

value	A binary image containing the small checkpoint mask.

6.22.3.13 void FrameList::setDoorMask (const cv::Mat & *value*)

Defines the doorway.

Parameters

value	A binary mask.

6.22.3.14 void FrameList::setExclusionMask (const cv::Mat & *value*)

Defines the exclusion mask (an area of the image set to constant zero).

Parameters

value	A binary mask.

6.22.3.15 void FrameList::setInclusionMask (const cv::Mat & *value*)

Sets the inclusion mask.

Parameters

value	A binary image.

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

- src/Utilities/FrameList.hpp
- src/Utilities/FrameList.cpp

6.23 statistics::FrameQueData Struct Reference

A struct the containing information from each frame needed to determine queue status over time.

`#include <QueSeverityEstimator.hpp>`

Public Member Functions

- FrameQueData (int peopleInRoom=0, bool queIsPresent=false)

  Constructor.

Public Attributes

- int peopleInRoom

  The number of people inside the room.
- bool queIsPresent

  Whether a queue is visible in this frame or not.

6.23.1 Detailed Description

A struct the containing information from each frame needed to determine queue status over time.

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

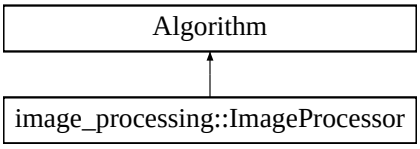
- src/Analytics/QueSeverityEstimator.hpp

6.24 image_processing::ImageProcessor Class Reference

The Image Processor is the interface to the image processing functionality.

`#include <ImageProcessor.hpp>`

Inheritance diagram for image_processing::ImageProcessor:



Public Member Functions

- ImageProcessor ()
  Constructor.
- ~ImageProcessor ()
  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)
  Initialize all image processing algorithms.
- void reset ()
  Clear all sub algorithms.
- void process (FrameList &frames)
  Perform all image processing algorithms in the proper order.

Additional Inherited Members

6.24.1 Detailed Description

The Image Processor is the interface to the image processing functionality.

6.24.2 Member Function Documentation

6.24.2.1 bool image_processing::ImageProcessor::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initialize all image processing algorithms.

Returns false if any of the sub algorithms fail during their initialization.

This algorithm acts as an interface and has no configurable parameters:

Returns

True if successful.

Reimplemented from Algorithm.

6.24.2.2 void image_processing::ImageProcessor::process (FrameList & *frames*) [virtual]

Perform all image processing algorithms in the proper order.

Parameters

frames	Current FrameList object.

Reimplemented from Algorithm.

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

- src/ImageProcessing/ImageProcessor.hpp
- src/ImageProcessing/ImageProcessor.cpp

6.25 evaluation::inOutEvent Struct Reference

Describes how many people entered or exited the room in the current frame.

`#include <EntryExitEvaluator.hpp>`

Public Attributes

- int **in**
- int **out**

6.25.1 Detailed Description

Describes how many people entered or exited the room in the current frame.

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

- src/Evaluation/EntryExitEvaluator.hpp

6.26 kinect::KinectFrame Struct Reference

Struct for handling output data created by the KinectHandler Class.

`#include <KinectHandlerOpenNi.hpp>`

Public Attributes

- cv::Mat **bgrImage**
- cv::Mat **depthImage**
- std::uint32_t **timestamp**

6.26.1 Detailed Description

Struct for handling output data created by the KinectHandler Class.

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

- src/Network/KinectHandlerFreenect.hpp
- src/Network/KinectHandlerOpenNi.hpp

6.27 kinect::KinectHandler Class Reference

Handler for the Kinect sensors.

`#include <KinectHandlerFreenect.hpp>`

Public Member Functions

- KinectHandler ()

  Constructor.
- ~KinectHandler ()

  Destructor.

- bool initialize ()

  Initializes the Kinect sensor.
- KinectFrame * readFrame (int deviceID)

  readFrame Returns a pointer to a KinectFrame containing the last images.
- int getnDevices ()

  getnDevices
- KinectHandler ()

  Constructor.
- ~KinectHandler ()

  Destructor.
- bool initialize ()

  Initializes the Kinect sensor.
- KinectFrame * readFrame (int deviceId)

  Returns a pointer to the latest KinectFrame containing the last images.
- int getnDevices ()

  getnDevices

6.27.1 Detailed Description

Handler for the Kinect sensors.

The class is a hardware interface for the Microsoft Kinect and contains functionality to convert output RGB and depth streams into OpenCV images.

6.27.2 Member Function Documentation

6.27.2.1 int kinect::KinectHandler::getnDevices ( )

getnDevices

Returns

The number of active devices.

6.27.2.2 int kinect::KinectHandler::getnDevices ( )

getnDevices

Returns

The number of active devices.

6.27.2.3 bool kinect::KinectHandler::initialize ( )

Initializes the Kinect sensor.

Returns

True if any devices were found.

6.27.2.4 bool kinect::KinectHandler::initialize ( )

Initializes the Kinect sensor.

Returns

True if any devices were found.

6.27.2.5 KinectFrame * kinect::KinectHandler::readFrame (int *deviceID*)

readFrame Returns a pointer to a KinectFrame containing the last images.

Parameters

deviceId	The index (zero based) of the device to be read.

Returns

Returns zero if no frame is available.

6.27.2.6 KinectFrame* kinect::KinectHandler::readFrame (int *deviceId*)

Returns a pointer to the latest KinectFrame containing the last images.

Parameters

deviceId	The index (zero based) of the device to be read.

Returns

Returns zero if no frame is available.

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

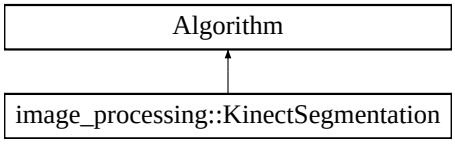
- src/Network/KinectHandlerFreenect.hpp
- src/Network/KinectHandlerOpenNi.hpp
- src/Network/KinectHandlerFreenect.cpp
- src/Network/KinectHandlerOpenNi.cpp

6.28 image_processing::KinectSegmentation Class Reference

Require a depth image "rawImage" in the current frame for each camera, where darker is closer and black means undefined. Produces a vector of detected objects stored in the current frame for each camera.

`#include <KinectSegmentation.hpp>`

Inheritance diagram for image_processing::KinectSegmentation:



Public Member Functions

- KinectSegmentation ()

  Constructor.
- ~KinectSegmentation ()

  Destructor.
- void process (FrameList &frames)

  The processing step of the Kinect Segmentation of human heads (and other large enough tall objects).
- bool initialize (configuration::ConfigurationManager &conf)

  Initialize the algorithm.

Additional Inherited Members

6.28.1 Detailed Description

Require a depth image "rawImage" in the current frame for each camera, where darker is closer and black means undefined. Produces a vector of detected objects stored in the current frame for each camera.

The algorithm perform the following steps:

1) The depth image is converted to gray scale and inverted except for the black color (and some margin: distanceToCameraMargin). Now darker is further away.

2) All values below lowestDistanceOverFloor is set to zero (black).

3) Contours are found in the image, any contour with an area less than minimalHumanArea is ignored.

4) Opening, Closening and Gaussian blurring is performed.

5) The maximum value in each contour is found and all values in respective contour less than headHeightMargin below the region max is set to zero, the rest form a combined binary mask.

6) Contours are found on the binary mask and each contour larger than minimalHeadArea is added as a potentially detected human to the current frame (of the processed camera), together with its center of mass and contour.

6.28.2 Member Function Documentation

6.28.2.1 bool image_processing::KinectSegmentation::initialize (configuration::ConfigurationManager & *conf*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set.

Configurable algorithm parameters are:

- headHeightMargin: The height interval croped from the top of the detected head and downwards. Affects the possibility of detecting shorter persons very close to long persons.
- lowestDistanceOverFloor: The limit of how short a person can be.
- distanceToCameraMargin: A margin length from the camera lens to the highest person possible to detect.
- minimalHumanArea: The minimal area required of a human, for it to be possible to detect.
- minimalHeadArea: The minimal area required of a human head, for it to be possible t detected.

Returns

True if successful.

Reimplemented from Algorithm.

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

- src/ImageProcessing/KinectSegmentation.hpp
- src/ImageProcessing/KinectSegmentation.cpp

6.29 debugging::LogEntry Struct Reference

A log entry is a container of a log information message.

`#include <Logger.hpp>`

Public Member Functions

- LogEntry (std::string tag, std::string message, std::string callingFunction, std::string fileName, std::string lineNumber)

  Constructor.
- std::string toString (std::string format="[%tag]%message\n(%file::%function@%line)[%time]")

  Get a string with default format or with specified formating.

Public Attributes

- std::string **tag**
- std::string **message**
- std::string **callingFunction**
- std::string **fileName**
- std::string **lineNumber**
- std::string **time**
- std::string **date**

6.29.1 Detailed Description

A log entry is a container of a log information message.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 debugging::LogEntry::LogEntry (std::string *tag,* std::string *message,* std::string *callingFunction,* std::string *fileName,* std::string *lineNumber*) [inline]

Constructor.

Parameters

tag	Is a string useful for filtering. Could be a name or identifier.
message	A log message.

callingFunction	The name of the function or method in which the logging is called.
fileName	The name of the file in which the logging is called.

6.29.3 Member Function Documentation

6.29.3.1 std::string debugging::LogEntry::toString (std::string *format* = "[%tag]%message\n(%file:-:%function@%line)[%time]")

Get a string with default format or with specified formating.

Format keywords are the field names with a %-prefix (see the default format below).

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

- src/Debugging/Logger.hpp
- src/Debugging/Logger.cpp

6.30 debugging::Logger Class Reference

Logger is a logging manager that is globally accessable under the alias 'logObject'.

`#include <Logger.hpp>`

Public Member Functions

- Logger ()
  Constructor.
- ~Logger ()
  Destructor.
- void append (LogEntry)
  Add a log entry and give it a time stamp.
- LogEntry pop ()
  Pop a log entry.
- LogEntry get (int index)
  Get a specific log entry.
- int size ()
  The number of stored log entries.
- bool isEmpty ()
  Weather or not no log entries exist.
- void clear ()
  Remove all log entries.
- void reset ()
  Calls clear() and resets iteration counter.
- LogIterator begin ()
  Get the begin iterator for the log entries.
- LogIterator end ()
  Get the end iterator for the log entries.
- std::string getLastEntry (std::string format="[%tag]%msg(%file::%function@%line)[%time]")
  Getter for the most recent log entry as a string, with a varity of formating options for the string content.
- std::vector< std::string > flushLog ()
  Empties the log into a vector of strings.
- void dumpToConsole ()

Empties the log and dumps all entries as strings in the console.

- ProfilerEntry popProfiler ()

  Pop the oldest profiler entry.
- int profilerSize ()

  Get the number of profiler entries.
- void profilerBeginIteration ()

  Initiate a new iteration for the profiler.
- void profilerEndIteration ()

  Finalize an iteration for the profiler.
- void profilerBeginSection (std::string tag)

  Initiate a new section for the profiler.
- void profilerEndSection ()

  Initialize a section for the profiler.
- ProfilerEntry * getLatestIteration ()

  Get the most recent iteration from the profiler.
- void profilerDumpSectionToConsole (ProfilerEntry *pe, int depth=0)

  Dump a specific iteration from the profiler.

6.30.1 Detailed Description

Logger is a logging manager that is globally accessable under the alias 'logObject'.

Logger manages LogEnty's and provides an interface for easy readouts of log entries.

6.30.2 Member Function Documentation

6.30.2.1 LogEntry debugging::Logger::get (int *index*) [inline]

Get a specific log entry.

Warning

index must be between 0 and size()-1

6.30.2.2 std::string debugging::Logger::getLastEntry (std::string *format* = "[%tag]%msg(%file:-:%function@%line)[%time]")

Getter for the most recent log entry as a string, with a varity of formating options for the string content.

Format keywords are the field names with a %-prefix (see the default format below).

6.30.2.3 void debugging::Logger::profilerDumpSectionToConsole (ProfilerEntry * *pe*, int *depth* = 0)

Dump a specific iteration from the profiler.

Parameters

pe	Using the node pe as root when dumping its entire profiler tree..
depth	The number of subdivisions to dump. 0 means all of them.

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

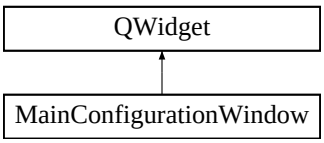
- src/Debugging/Logger.hpp
- src/Debugging/Logger.cpp

6.31 MainConfigurationWindow Class Reference

The MainConfigurationWindow class is a window where you can cnfigure different settings in the system.

`#include <MainConfigurationWindow.hpp>`

Inheritance diagram for MainConfigurationWindow:



Public Slots

- void updateWindow (Frame currentFrame)

  updateWindow recieves a frame and updates the content in the MainConfigurationWindow.

Public Member Functions

- MainConfigurationWindow (QWidget *parent=0)

  Constructor.
- ~MainConfigurationWindow ()

  Destructor.
- void initialize (DenseKitchen *mainProgram, std::string masksConfigFile)

  initialize sets up the MainConfigurationWindow

6.31.1 Detailed Description

The MainConfigurationWindow class is a window where you can cnfigure different settings in the system.

Especially masks used to define doors and exceptions for a camera. The door mask is used to check if a person is currently in a doorway and the exception mask is used to speed up and harden the system by defining ares that are not interesrting for the system.

6.31.2 Member Function Documentation

6.31.2.1 void MainConfigurationWindow::initialize (DenseKitchen * *mainProgram*, std::string *masksConfigFile*)

initialize sets up the MainConfigurationWindow

Parameters

mainProgram	is a poninter to the main program.
masksConfigFile	contains the coordinates for the masks used by the system.

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

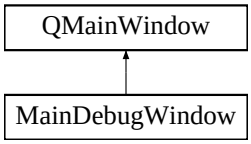
- src/Debugging/MainConfigurationWindow.hpp
- src/Debugging/MainConfigurationWindow.cpp

6.32 MainDebugWindow Class Reference

The MainDebugWindow class is a debug interface to speed up development, testing and validation of image processing algorithms.

`#include <MainDebugWindow.hpp>`

Inheritance diagram for MainDebugWindow:



Signals

- void updateDebugViews (Frame currentFrame)

  updateDebugViews is used to send a fresh Frame to all sub-widgets and update their content.

Public Member Functions

- MainDebugWindow (QWidget *parent=0)

  Constructor.
- ~MainDebugWindow ()

  Destructor.
- void init (std::string mainConfigFile, std::string guiConfigFile)

  Initializes the GUI with values specified in guiConfig.yml.

Friends

- class DenseKitchen

  The GUI needs full access to DenseKitchen.

6.32.1 Detailed Description

The MainDebugWindow class is a debug interface to speed up development, testing and validation of image processing algorithms.

6.32.2 Member Function Documentation

6.32.2.1 void MainDebugWindow::init (std::string *mainConfigFile*, std::string *guiConfigFile*)

Initializes the GUI with values specified in guiConfig.yml.

Parameters

mainConfigFile	contains settings for the main program pipeline.
guiConfigFile	contains settings for the GUI.

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

- src/Debugging/MainDebugWindow.hpp
- src/Debugging/MainDebugWindow.cpp

6.33 network::Network Class Reference

The Network class handles all system I/O.

`#include <Network.hpp>`

Public Member Functions

- Network ()

  Constructor.
- ~Network ()

  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Initializes the network module.
- void reset ()

  Destroys the video streams, thereby resetting the network module.
- Frame * dequeFrame (void)

  dequeFrame Returns a pointer to the next frame.
- void broadcastData (Frame &frame)

  Send data to the web server.

6.33.1 Detailed Description

The Network class handles all system I/O.

The Network class manages all system input/output, i.e. sampling the sensors and posting output data on the web server specified by the configuration file. Currently, support exists for all OpenCV-compatible cameras as well as the Microsoft Kinect depth sensor. The class is designed to be as modular as possible, allowing for a relatively easy integration of new sensor types into the system. The Network class also supports running several sensors in parallel.

6.33.2 Member Function Documentation

6.33.2.1 void network::Network::broadcastData (Frame & *frame*)

Send data to the web server.

The data that is presented on the web server is:

- Number of people currently in side the room.
- If there is a queue to enter the room or not.

Parameters

frame	Frame whose data is to be broadcasted

6.33.2.2 Frame * network::Network::dequeFrame (void)

dequeFrame Returns a pointer to the next frame.

Tries to sample the different vidoe streams and returns a pointer to a newly created frame object if all samplings are successful.

Returns

Returns zero if no frame is available.

6.33.2.3 bool network::Network::initialize (configuration::ConfigurationManager & *settings*)

Initializes the network module.

The tries to initialize a Network object by opening up the necessary video streams specified by the settings parameter. If this for some reason is not possible, an error log message is created and the function retirns false.

Parameters

settings	A configuration object containing program settings.

Returns

True if successful.

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

- src/Network/Network.hpp
- src/Network/Network.cpp

6.34 Object Struct Reference

A movable object that has been detected, and that potentially might be a human.

`#include <Object.hpp>`

Public Member Functions

- Object ()
  Constructor.
- ~Object ()
  Destructor.
- Object (std::vector< cv::Point > &contour, cv::Rect &boundingBox, cv::Point2f &centerOfMass, double area)
  Constructor using a cv::Rect for initialization.
- void merge (Object *previousState)
  Merge current state of an object with the previous.
- void enter ()
  Called when an object has entered the view.
- void exit ()
  Called when an object has exited the view.

Public Attributes

- int **id**
- cv::Rect **boundingBox**
- std::vector< cv::Point > **contour**
- double **area**
- cv::Point2f **centerOfMass**
- cv::Point2f **velocity**
- cv::Point2f **positionPrediction**
- cv::Point2f **velocityPrediction**
- cv::Point2d **entryPoint**
- cv::Point2d **exitPoint**
- bool **lost**

- bool **hasPassedMasksOne**
- bool **hasPassedMasksTwo**
- bool **hasPassedMasksThree**
- bool **hasAlreadyEntered**
- int **lifeSpan**
- cv::KalmanFilter **kalmanFilter**

6.34.1 Detailed Description

A movable object that has been detected, and that potentially might be a human.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 Object::Object (std::vector< cv::Point > & *contour,* cv::Rect & *boundingBox,* cv::Point2f & *centerOfMass,* double *area*)

Constructor using a cv::Rect for initialization.

Parameters

contour	The 2-dimensional contour of the object.
boundingBox	An axis-aligned bounding box.
centerOfMass	The center of mass of the object contour
area	Object contour area.

6.34.3 Member Function Documentation

6.34.3.1 void Object::enter ( )

Called when an object has entered the view.

Sets the Object property entryPoint to the current center of mass, thus saving information about where the object spawned for the first time.

6.34.3.2 void Object::exit ( )

Called when an object has exited the view.

Sets the Object property exitPoint to the to the point were the object was last seen, thus saving information about where the object exited the field of view.

6.34.3.3 void Object::merge (Object * *previousState*)

Merge current state of an object with the previous.

First all variables of the previous are copied to this object, including the kalman filter object. Then the kalman filter perform a measurement update, estimating the current velocity, and a prediction of the next position and velocity.

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

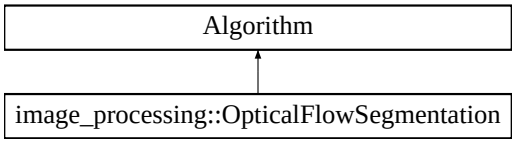
- src/Utilities/Object.hpp
- src/Utilities/Object.cpp

6.35 image_processing::OpticalFlowSegmentation Class Reference

Computes the optical flow and does some basic segmentation based on it.

`#include <OpticalFlowSegmentation.hpp>`

Inheritance diagram for image_processing::OpticalFlowSegmentation:



Public Member Functions

- OpticalFlowSegmentation ()
  constructor
- ~OpticalFlowSegmentation ()
  destructor
- void process (FrameList &frames) override
  The processing performs an optical flow calculation and averages the resulting vectors blockwise.
- bool initialize (configuration::ConfigurationManager &settings) override
  Initialize the algorithm.

Additional Inherited Members

6.35.1 Detailed Description

Computes the optical flow and does some basic segmentation based on it.

The algorithm performs the following steps

1) The algorithm converts the image to grayscale 2) The algorithm runs a keypoint detector 3) The algorithm tracks the points over a few frames 4) Re-run the detector when to many points have been lost 5) Average the flow vectors blockwise The algorithm does not: 1) Compute an acual segmentation 2) Run fast enough to be practical

6.35.2 Member Function Documentation

6.35.2.1 bool image_processing::OpticalFlowSegmentation::initialize (configuration::ConfigurationManager & *settings*) [override], [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set.

Configurable algorithm parameters are: This algorigthm does not have any configurable parameters

Returns

True if successful.

Reimplemented from Algorithm.

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

- src/ImageProcessing/OpticalFlowSegmentation.hpp
- src/ImageProcessing/OpticalFlowSegmentation.cpp

6.36 ProcessHistoryEntry Struct Reference

A process history entry is a container of image processing history.

`#include <CameraObject.hpp>`

Public Member Functions

- **ProcessHistoryEntry** (std::string tag, std::string time, cv::Mat image)

Public Attributes

- std::string **tag**
- std::string **time**
- cv::Mat **image**

6.36.1 Detailed Description

A process history entry is a container of image processing history.

Stored is a tag and a time associated with a resulting image matrix.

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

- src/Utilities/CameraObject.hpp

6.37 debugging::ProfilerEntry Struct Reference

A profiler entry is a container of profiler information.

`#include <Logger.hpp>`

Public Member Functions

- ProfilerEntry (std::string tag, int64 begun, ProfilerEntry *parent)

  Contructor.

Public Attributes

- std::string **tag**
- double **milliseconds**
- int64 **begun**
- int64 **ended**
- std::list< ProfilerEntry > **children**
- ProfilerEntry * **parent**

6.37.1 Detailed Description

A profiler entry is a container of profiler information.

It contain a tag (name), time for the entire node in milliseconds and subdivisions stored as children which ar other ProfilerEntry objects.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 debugging::ProfilerEntry::ProfilerEntry (std::string *tag*, int64 *begun*, ProfilerEntry * *parent*) [inline]

Contructor.

Parameters

tag	Is a string useful for filtering. May be a name or identifier.

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

- src/Debugging/Logger.hpp

6.38 statistics::Que Struct Reference

A queue of persons.

`#include <Que.hpp>`

Public Attributes

- std::vector< SplineStrip > splineStrips

  All the splinestrips in the queue.

- std::map< int, Object > queObjects

  A map containing all of the objects in the queue.

- std::vector< DirectedQueEdge > queEdges

  All the directed edges in the queue.

6.38.1 Detailed Description

A queue of persons.

6.38.2 Member Data Documentation

6.38.2.1 std::map<int,Object> statistics::Que::queObjects

A map containing all of the objects in the queue.

The key is the objects id number.

6.38.2.2 std::vector<SplineStrip > statistics::Que::splineStrips

All the splinestrips in the queue.

The splines are subdiveded and ready to draw.

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

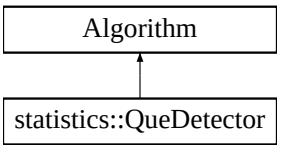
- src/Analytics/Que.hpp

6.39 statistics::QueDetector Class Reference

Process step which uses information about visible persons' position and velocity and determines if a que is present or not.

`#include <QueDetector.hpp>`

Inheritance diagram for statistics::QueDetector:



Public Member Functions

- QueDetector ()

  Constructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Handles initialization.
- void process (FrameList &frames)

  Performs the algorithm.

Additional Inherited Members

6.39.1 Detailed Description

Process step which uses information about visible persons' position and velocity and determines if a que is present or not.

A cubic bezier spline is fit to every pair of visible object/persons (one spline per direction) for each camera. The objects are used as the two on-curve control points and the objects' velocities are used to calculate the two off-curve control points. Objects that are connected by a sufficiently short spline are considered to be in the same queue. If any queues are found, this is indicated in the camera object for which the queue is visible. The class is written such that it would be easy to extend it so to be able to do more advanced analysis of the queue structures.

6.39.2 Member Function Documentation

6.39.2.1 bool statistics::QueDetector::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Handles initialization.

Configurable algorithm parameters are:

- maxSplineSegmentLength: Spline subdivision stops when the longest segment in the spline drops below this length (in pixels).
- velocityScaleFactor: The scale factor applied to the velocity vectors when creating a spline. (To account for spline parametrization)
- splineLengthThreshold: Pairs of objects connected by a spline longer than this are not considered part of the same queue.
- maxRecursionDepth: Spline subdivisions are stopped at this depth of recursions.

Parameters

settings	A configuration::ConfigurationManager object that potentially includes values for the algorithm's constants.

Returns

Returns false if the initialization fails, e.g. if a required variable is not set in the config file.

Reimplemented from Algorithm.

6.39.2.2 void statistics::QueDetector::process (FrameList & *frames*) [virtual]

Performs the algorithm.

Parameters

frames	A FrameList object containing the current and some of the previos frames.

Reimplemented from Algorithm.

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

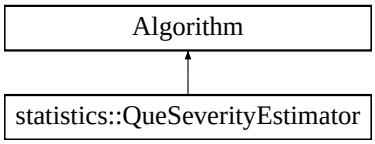
- src/Analytics/QueDetector.hpp
- src/Analytics/QueDetector.cpp

6.40 statistics::QueSeverityEstimator Class Reference

Process step that aggregates queue visibility over time and uses this information to output a stable classification of the current queue severity.

`#include <QueSeverityEstimator.hpp>`

Inheritance diagram for statistics::QueSeverityEstimator:



Public Member Functions

- QueSeverityEstimator ()
  Constructor.
- bool initialize (configuration::ConfigurationManager &settings)
  Handles initialization.
- void process (FrameList &frames)
  Performs the algorithm.

Additional Inherited Members

6.40.1 Detailed Description

Process step that aggregates queue visibility over time and uses this information to output a stable classification of the current queue severity.

The calculated queue classification value (0,1,2) is placed in the current frame object.

6.40.2 Member Function Documentation

6.40.2.1 bool statistics::QueSeverityEstimator::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Handles initialization.

Configurable algorithm parameters are:

- historyLength: The number of frames considered when classifying queue severity.
- lowQueThreshold: If the proportion of considered frames with a detected queue is less than this, queue severity is 0.
- highQueThreshold: If the proportion of considered frames with a detected queue is above this, queue severity is 2.
- lowOccupancyThreshold: If there are more people than this and less than highOccupancyThreshold, queue severity is 1.
- highOccupancyThreshold: If there are more people than this, queue severity is 2.

Parameters

settings	A configuration::ConfigurationManager object that potentially includes values for the algorithm's constants.

Returns

Returns false if the initialization fails, e.g. if a required variable is not set in the config file.

Reimplemented from Algorithm.

6.40.2.2 void statistics::QueSeverityEstimator::process (FrameList & *frames*) [virtual]

Performs the algorithm.

Parameters

frames	A FrameList object containing the current and some of the previos frames.

Reimplemented from Algorithm.

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

- src/Analytics/QueSeverityEstimator.hpp
- src/Analytics/QueSeverityEstimator.cpp

6.41 roomPopulation Struct Reference

Struct for keeping track of people entering the same room from different cameras.

`#include <Frame.hpp>`

Public Attributes

- int **people**
- std::string **roomID**

6.41.1 Detailed Description

Struct for keeping track of people entering the same room from different cameras.

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

- src/Utilities/Frame.hpp

6.42 statistics::SplineStrip Struct Reference

A cubic bezier spline.

`#include <Que.hpp>`

Public Member Functions

- SplineStrip (cv::Point2f P0, cv::Point2f C1, cv::Point2f C2, cv::Point2f P1)
  Constructor.
- float length ()
  Calculates upper bound of spline length.
- float maxSegmentLength ()
  Calculates th maximum segment length.

Public Attributes

- cv::Point2f p0
  First on-curve control point.
- cv::Point2f c1
  First off-curve control point.
- cv::Point2f c2
  Second off-curve control point.
- cv::Point2f p1
  Second on-curve control point.

6.42.1 Detailed Description

A cubic bezier spline.

This struct holds the four control points of a cubic bezier spline and can be queried about its (approximate) length and its longest segment.

6.42.2 Member Function Documentation

6.42.2.1 float statistics::SplineStrip::length ( ) [inline]

Calculates upper bound of spline length.

The control points of a bezier spline define a convex hull for the curve. The upper bound for the curve is therefore calculated as the lengths of the lines between each of the control points.

Returns

The lenght.

6.42.2.2 float statistics::SplineStrip::maxSegmentLength ( ) [inline]

Calculates th maximum segment length.

The maximum segment length is the longest distance between consecutive control points. It can be used to determin when to stop a recursive spline subdivision.

Returns

Maximum segment length.

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

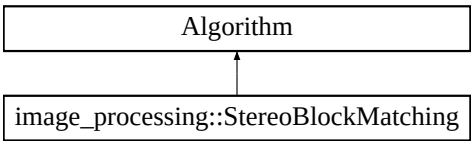
- src/Analytics/Que.hpp

6.43 image_processing::StereoBlockMatching Class Reference

Process step which creates a depth map from a stereo image pair.

`#include <StereoBlockMatching.hpp>`

Inheritance diagram for image_processing::StereoBlockMatching:



Public Member Functions

- StereoBlockMatching ()
  Constructor.
- ~StereoBlockMatching ()
  Destructor.
- void process (FrameList &frames)
  Performs stereo block matching using stereo block matching.
- bool initialize (configuration::ConfigurationManager &conf)
  Initialize the algorithm.

Additional Inherited Members

6.43.1 Detailed Description

Process step which creates a depth map from a stereo image pair.

6.43.2 Member Function Documentation

6.43.2.1 bool image_processing::StereoBlockMatching::initialize (configuration::ConfigurationManager & *conf*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set.

This alghorithm has no configurable parameters.

Returns

True if successful

Reimplemented from Algorithm.

6.43.2.2 void image_processing::StereoBlockMatching::process (FrameList & *frames*) [virtual]

Performs stereo block matching using stereo block matching.

The stereo block matching algrothm used is the OpenCV semi-global block matcher (cv::StereoSGBM) This algortihm is based on one defined by Hirschmuller, H in 2008.

Hirschmuller, H. Stereo Processing by Semiglobal Matching and Mutual Information, PAMI(30), No. 2, February 2008, pp. 328-341

Parameters

frames	The current FrameList

Reimplemented from Algorithm.

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

- src/ImageProcessing/StereoBlockMatching.hpp
- src/ImageProcessing/StereoBlockMatching.cpp

6.44 Timer Class Reference

Timer that measures time durations from latest reset.

`#include <Timer.hpp>`

Public Member Functions

- Timer ()
  Constructor, resets the timer.
- void reset ()
  Resets timer, starts to mesure the time from now.
- void resetFromLast ()
  Reset timer from last sampling (using any get method).
- double getSeconds ()
  Get the time in seconds since last reset.
- double getMilliseconds ()
  Get the time in milliseconds since last reset.
- double getMicroseconds ()
  Get the time in microseconds since last reset.
- double getNanoseconds ()
  Get the time in nanoseconds since last reset.

6.44.1 Detailed Description

Timer that measures time durations from latest reset.

Derive form this class to maintain a simple interface for all algorithms used in the processing pipeline.

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

- src/Utilities/Timer.hpp
- src/Utilities/Timer.cpp

6.45 evaluation::TrackerEvaluator Class Reference

Tracker accuracy evaluator.

`#include <TrackerEvaluator.hpp>`

Public Member Functions

- TrackerEvaluator ()

  Empty Constructor.
- ~TrackerEvaluator ()

  Destructor.
- bool initialize (std::string groundTruthPath, int precisionThresh)

  Initializes the tracking module.
- void process (CameraObject &cam)

  Calculates MOTA and MOTP for the CameraObject.
- void printToLog (unsigned int camIndex)

  Prints the MOTA and MOTP values ro the debug log.

6.45.1 Detailed Description

Tracker accuracy evaluator.

Evaluates the performance of the tracker by comparing the output to ground truth data. Performance is measured using the MOTA and MOTP methods described in 2008 by Bernardin, K. and Stiefelhagen, R. in their paper:

"Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics". Interactive Systems Lab, Institut für Theoretische Informatik, Universität Karlsruhe, 76131 Karlsruhe, Germany.

6.45.2 Member Function Documentation

6.45.2.1 bool evaluation::TrackerEvaluator::initialize (std::string *groundTruthPath*, int *precisionThresh*)

Initializes the tracking module.

The Tracker evaluator is initialized by reading the ground truth file using the rapidxml library.

Parameters

groundTruthPath	Path to ground truth xml-file.
precision-Threshold	Threshold for the MOTA/MOTP measurements (see paper).

Returns

Returns true if successful.

6.45.2.2 void evaluation::TrackerEvaluator::printToLog (unsigned int *camIndex*)

Prints the MOTA and MOTP values ro the debug log.

Parameters

camIndex	Index of the current camera.

6.45.2.3 void evaluation::TrackerEvaluator::process (CameraObject & *cam*)

Calculates MOTA and MOTP for the CameraObject.

Parameters

cam	The CameraObject in question.

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

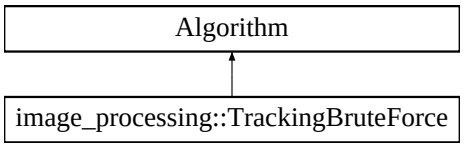
- src/Evaluation/TrackerEvaluator.hpp
- src/Evaluation/TrackerEvaluator.cpp

6.46 image_processing::TrackingBruteForce Class Reference

Process step which tracks bounding boxes between frames.

`#include <TrackingBruteForce.hpp>`

Inheritance diagram for image_processing::TrackingBruteForce:



Public Member Functions

- TrackingBruteForce ()

  Constructor.
- ~TrackingBruteForce ()

  Destructor.
- bool initialize (configuration::ConfigurationManager &settings)

  Initialize the algorithm.
- void process (FrameList &frames)

  Performs the tracking.

Additional Inherited Members

6.46.1 Detailed Description

Process step which tracks bounding boxes between frames.

The tracking algorithm performs six different steps for each frame iteration. The tracker has a list with objects from the last frame and a list with current objects found by earlier image processing steps. Objects are separated in different classes; candidate objects, lost objects and real objects. Candidate objects are objects that have been found but not lived long enough to be considered as a real object. Lost objects are objects that once were real objects but have been lost in the tracking. If an new object reappears close to the lost object it will be considered to

be found and directly considered as a real object again. Real objects are the objects which has lived long enough and therefore considered as humans. Below the pipeline is described in six steps.

1) Check if the objects, received from the image processing made earlier in the pipeline, are inside any of the checkpoint masks and set flags. The checkpoint masks are the door area and circles which are set uped in the initial configuration. 2) Previous objects from the last frame are paired with current candidate objects and moved to current object list. Pairing is done by relating the closest pairs to each other. 3) A list with previous potential objects are paired with the remaining current candidate objects 4) Any remaining current candidates in the list is added as (new) current potential object. 5) Any remaining previous objects are flagged as lost. If one remaining object has been lost longer than a set maximum time, the object is removed. 6) Elevate previous candidate objects to real objects if they have lived long enough.

6.46.2 Member Function Documentation

6.46.2.1 bool image_processing::TrackingBruteForce::initialize (configuration::ConfigurationManager & *settings*) [virtual]

Initialize the algorithm.

Returns false if initialization fails, e.g. if a required variable is not set in the config file.

Configurable algorithm parameters are:

- maximumDistance: The maximum distance a object can be considered to have moved since last frame.
- minimumLifeSpan: The minimal time (in # frames) a potential object must have existed (and been tracked) before it is considered a real object.
- maximumTimeLostStill: The maximum time (in # frames) a object is allowed to be lost before it is forgotten.
- maximumTimeLostInDoorArea: The maximum time (in # frames) a object is allowed to be lost in the door area before it is forgotten.

Returns

True if successful.

Reimplemented from Algorithm.

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

- src/ImageProcessing/TrackingBruteForce.hpp
- src/ImageProcessing/TrackingBruteForce.cpp

Index