OMNIDIRECTIONAL VISUAL TRACKING

By
Mark Borg

APPENDICES:

SOURCE CODE LISTING

Contents

Introduction

The following are the version of the OpenCV library and the PLS libraries used by the OmniTracking application. For more information about these libraries, see §5.3 of the main document. Instructions on how to download the OpenCV library (and the related PLS libraries) can be found at: http://www.intel.com/research/mrl/research/opencv.

Library name	Version	Release Date
OpenCV	Beta 3.1	27 Feb 2003
IJL	v. 1.5.4	15 Sept 2000
IPL	v. 2.5	7 Sept 2000

The application was compiled using Microsoft Visual C++ v. 6.0 (1998). The project file OmniApp2.dsw is available on the CD-ROM that comes with these appendices. Alternatively, a make file could be used.

The source code for the OmniTracking application is divided into the following folders:

Folder name	Description
framework	This folder contains the source code that implements the generic pipeline framework as described in §5.5.
core	This folder contains the source code that implements the algorithms specific to the OmniTracking application.
tools	Miscellaneous source files that range from data structures, a trace log handling class, etc.
gui	This folder contains the source code that interfaces with the Windows GUI to display the application's results.

The next set of tables give an overview of what each source file in each folder (package) is used for. The source files are listed in alphabetical order both in the tables below and also in the actual code listings.

Pipeline Framework package

Source file	Description
Colour2GrayNode.h	A node that performs colour to greyscale conversion of the image it obtains from its source nodes, and sends the output to the sink nodes.
CorrectAspectRatio.h	This node resizes an image. It is used to correct the aspect ratio (if any) of an image. This node pre-calculates a look-up table on the first time it is used and then uses the LUT for faster conversions for the successive images.
framework.h	A header file that includes all the other header files of this package.
Gray2ColourNode.h	This node converts greyscale to colour images.
IplData.h	Provides a concrete implementation of the type RESOURCE used by the memory pool. This class stores an image structure of the type IplImage as defined by the IPL and OpenCV libraries.
IplImageResource.h	This class works in conjunction with IplData and serves as a resource descriptor for IplImage and is used by the memory pool to identify if a free image resource it has in its pool is similar to the one being requested by the caller. If similar, then the image is re-used; if not, a new image structure is allocated.
MemoryPool.h	This class keeps a memory pool of objects of abstract type RESOURCE. Reference counting (see §5.5) is used to determine if resource objects are free or not.

MemoryResource.h	Implements the reference counting mechanism for resources. This class is used by MemoryPool as a wrapper around the abstract type RESOURCE.
ProcessorNode.h	The main class of the pipeline framework. This is the base class that implements the 'node' of the pipeline process. It handles the arrival of data from the source nodes and sends processed data to the sink nodes. It basically performs the action given in Figure 5.3 and serves as the base class for actual nodes that do some interesting processing stuff. All the nodes in this folder are sub-classed from ProcessorNode
ReadJpegSequence.h	This node reads a sequence of JPEG images from disk and passes them on to the sink nodes.
RunnableObject.h	This class encapsulates a thread within an object. From the point of view of the object, it is active, in the sense that it (and anything sub-classed from it) runs on its own once it is started. It also serves to hide the Windows thread functions and means that in the future these functions can be exchanged with POSIX-compliant thread functions w/o having to change the rest of the application code.
SlideShowViewerNode.h	This node forwards a set of images it gets from its source nodes to the sink node. This node can be viewed as received input in parallel from multiple source nodes and then it 'serialises' them to the sink node in sequence with a pre-defined delay between each image. This node is used to display the (partial) results of the calibration process.
SyncNode.h	This node synchronises the pipeline by forcing the source nodes to wait for its synchronisation beat. It uses the OS millisecond timers in conjunction with the high performance clock on the processor to achieve precise resolution of up to 1 msec. (see §5.4).

ViewerNode.h	This is a node that passes images to the GUI system (serves as a bridge between the pipeline and the GUI). If the GUI element to which it sends data is hidden/closed/minimised then the node does not forward the data
WriteBmpSequence.h	This nodes saves a sequence of images it receives from the source nodes to disk. This node is normally used for debugging purposes. The BMP format was selected over JPEG because it is lossless.

Application Core package

Source file	Description
BivariateGaussian.h	Implements a bivariate Gaussian distribution using an abstract template parameter for the data vector.
BkgSubtracter.h	This node implements the background subtraction algorithm as described in §8.7.
Blob.h	The data structure used to represent groups of contiguous pixels, as used during blob tracking.
BlobTracker.h	The blob-tracking method described in §10.4.
BoundaryDetector.h	Detects the boundary of the mirror in an omnidirectional image, by performing circle detection using the Hierarchical Hough transform (see §7.4.2.4)
CfgParameter.h	This class reads the configuration parameters from the initialisation file.
ColourTracker.h	This node implements the statistical colour tracking method described in §11.
EdgeDetector.h	Used by the calibration process for detecting the mirror boundary (see §7.4.2.3).

EM.h	Implements the Expectation-Maximisation algorithm.
HistoryNode.h	Performs high-level operations like depth estimation and saving tracking results in HTML format to disk, as described in §12.1 and §12.3.
IEM.h	Implements an incremental version of the Expectation-Maximisation (EM) algorithm (§11.5).
IterativeThreshold.hpp	Used by the calibration process to split an omnidirectional image into scene pixels and blank pixels (see §7.4.2.2).
Log.h	This class implements an interface for a trace handling objects.
MatchScores.h	The table object used to compare similarity criteria and keep the scores in the table for the blob-tracking method (§10.4.1).
MixtureModel.h	A Gaussian mixture model implementation
MixtureModelMap.h	A sub-class of MixtureModel that keeps an LUT with pre-calculated probability values.
Object.h	The data structure that keeps information about detected objects. Used by both blob tracking and colour tracking methods.
ObjectsInformation.h	The global data structures that keep information about objects being tracked across frames.
PanoramicDewarper.h	This node generates panoramic views from the omnidirectional image, as well as implementing the virtual camera control mechanisms.
PerspectiveDewarper.h	This node generates perspective views from the omnidirectional image, as well as implementing the virtual camera control mechanisms.

Pipeline.h	The main class that creates and controls the application processing pipeline depicted in Figure 5.4. It controls all the nodes in the pipeline and allows the pipeline to be stopped/paused/started.
SyncNodeWithStatistics.h	This class extends SyncNode and adds some functionality for reporting information such as fps.
Tracker.h	This file contains code common to both object tracking methods.
Version.h	Application version information.
VirtualCameraController	This node creates virtual cameras that automatically track the objects detected by the motion detection process (see §12.2).

Tools package

Source file	Description
CvUtils	Miscellaneous functions that enhance some functions or the usage of OpenCV. (Most of these functions use OpenCV's data structures; like IplImage, CvRect, CvPoint, etc.).
DistinctColours.h	A support class that can be used to get a list of distinct colours. Used by the program to assign a different colour to each object/virtual camera for GUI purposes.
float_cast.h	A fast floating-point truncation operation - see comments in this file for more details.
hsv.h	Fast RGB to HSV conversion functions using only integer arithmetic. See end of §8.7.3 for reasons why these functions have been implemented.

Ijl_Ipl.h	Provides functions that read/write JPEG files from disk (by using the IJL library) and returns the image data as an IplImage data structure (as defined by IPL/OpenCV). Some functions allow parts of the JPEG disk file to be uncompressed and read (by using a ROI).
LogHandler.h	A generic interface for trace handling objects.
LWGraph.h	A lightweight graph that uses two underlying LWList objects and keeps the set of edges between elements of these lists. Intended to be used for bipartite graphs, but can also be used for more generic graphs by setting the two lists to the same object.
LWList.h	A lightweight list that can also be used as a vector. This list grows on demand and space is allocated in blocks of elements.
MemInfo.h	A class that displays the amount of free memory and disk space. Gets the information from the Windows OS.
Singleton.h	Classes that should have only one object instance running throughout the life of the program are derived from this template class. It enforces a single-instance property for classes and a lazy evaluation mechanism for the instance creation.
SingletonDestroyer.h	A template class that is used to cleanup and deallocate singleton objects upon program termination.
ThreadSafeHighgui.h	The HighGUI library of OpenCV is not thread-safe. This file provides a global mutex object that is used for accessing highgui functions.

Trig.h	A function that calculates the sine and cosine of a vector of numbers. Uses fast LUTs for trigonometric functions. This is a modified version of a similar function in OpenCV, but adapted to work using float data types.
Timer.h	A class that implements a timing functionality based on the high performance clock of the processor. Used for synching nodes, reporting frame rates and also used for timing purposes during debugging and optimisation of the application.

GUI package

Source file	Description
BkgView.h	The background GUI window view, against which floating windows are normally positioned.
CameraViewList.h	The GUI window for the side-bar that contains the icons.
FloatingImageView.h	The class representing a generic floating window.
FloatingOmniImageView.h	A sub-class of FloatingImageView used specifically to show omnidirectional images. This class makes changes to the window toolbar.
FloatingPanoramaView.h	A sub-class of FloatingImageView used specifically to show panoramic images.
FloatingPerspectiveView.h	A sub-class of FloatingImageView used specifically to show perspective images.
Ipl2Bmp.h	This class converts IplImages to a memory bitmapped structure as required by the Windows GUI.

wIpl2DecimatedBmp.h	Same as Ipl2Bmp, but also reduces size of image; used for icons.
LogView.h	The window serving as a logging console
mainframe.h	The main GUI window frame.
OmniApp2.h	The main GUI application class and the program entry point for the Windows OS.
resource.h	GUI resource constants.
SplashWindow.h	Title pop-up window.
StaticImageView.h	The image icon class (called static controls in GUI terminology).
stdafx.h	Main header file for the GUI classes.