ALVAR – A Library for Virtual and Augmented Reality User's Manual (v.2.0)

Change history

Version	**Date**	**Status** (draft/proposal/updated /to be reviewed/ approved)	**Author(s)**	**Remarks**
0.1	21.11.2011	Draft	TTERAK	First draft created.
0.2	12.12.2011	Draft	TTERAK	Background info added to Ch 9.3
0.3	15.12.2011	Draft	TTERAK	New screen shots from doc, FAQs, installation step-by-step
0.4, 0.5	25.01.2012	Draft	TTERAK	Content to sections 9.3.x
0.6	29.02.2012	Draft	TTERAK	GNU LGPL license
0.7	25.05.2012	Draft	TTERAK	Changes caused by the new ALVAR distribution packages
1.0	30.05.2012	Final	TTERAK	Screen dumps from v.2.0.0
1.1	13.12.2013	Update	ABALAIN	Add FAQ entries.

	Name	**Position**	**Date**	**Signatures**
Written by	Kari Rainio	TK8036	30.05.2012	
Reviewed by	Alain Boyer	TK8035	30.05.2012	
Approved by				

Contents

1 Introduction

1.1 Purpose of the Document

This document contains instructions for the users of the ALVAR open-source software library, published by VTT Technical Research Centre of Finland.

1.2 Other Documents

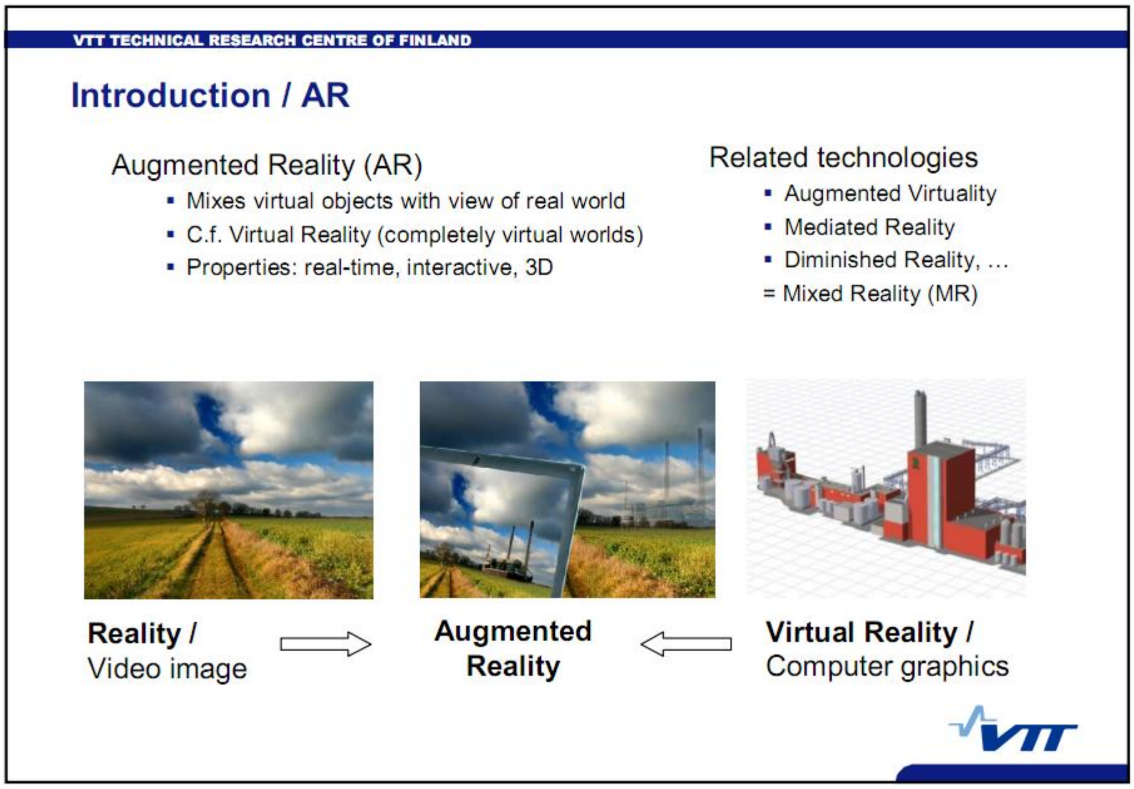
The VTT AR Team web page: http://www.vtt.fi/multimedia

AR Team web pages contain many useful documents of Augmented Reality in general and ALVAR in particular:

- Description of AR (slide show, pdf),
- Videos of team's AR applications,
- Downloadable AR demo applications,

- PowerPoint presentation of ALVAR, ALVAR User's Manual (this document),
- ALVAR downloads (bin, sdk, src – see later).

1.3 Introduction to Augmented Reality (AR) and VTT AR Team



VTT TECHNICAL RESEARCH CENTRE OF FINLAND

Introduction / AR

Augmented Reality (AR)

- Mixes virtual objects with view of real world
- C.f. Virtual Reality (completely virtual worlds)
- Properties: real-time, interactive, 3D

Related technologies

- Augmented Virtuality
- Mediated Reality
- Diminished Reality, …

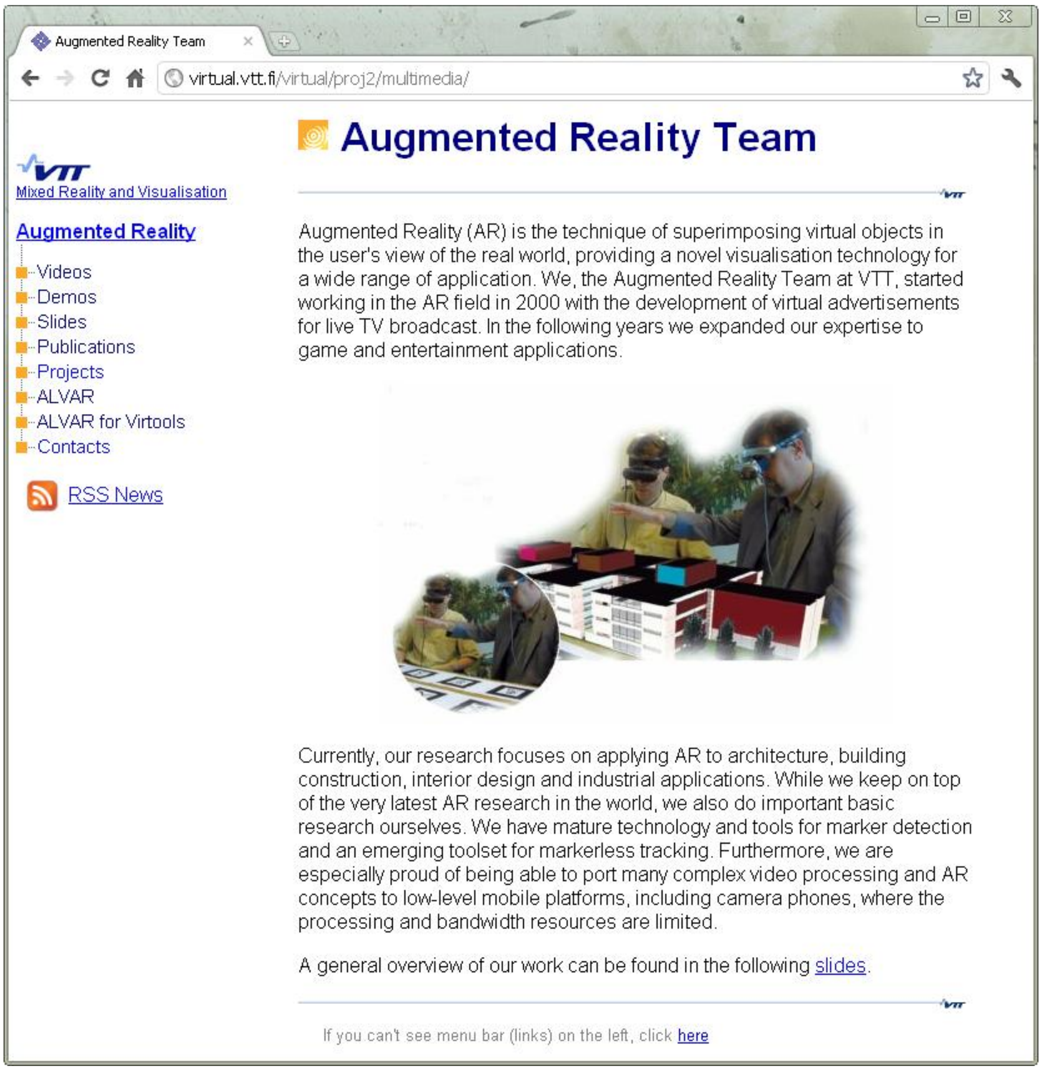
= Mixed Reality (MR)

Reality / Video image
Augmented Reality
Virtual Reality / Computer graphics

In Augmented Reality the user looks at a live video image of reality. The video image is real-time and interactive, i.e. the user can choose where (s)he looks and move around. Often head-up displays are employed for this, or tablet PCs (with cameras), or smart mobile phones.

The live video in analyzed by a computer, which deduces from the image features where the user is and which way (s)he looks. This way the computer can insert virtual objects to the scene so that they are displayed in the correct size and angle, i.e. the virtual objects seem to be "glued" to the reality.

There are lesser forms of Augmented Reality, e.g. displaying informative texts near the real objects of the scene, but VTT AR team in conserned with the most demanding form of Augmented Reality, where the goal is to insert virtual 3D objects to the real scene seamlessly.



Augmented Reality Team

Augmented Reality (AR) is the technique of superimposing virtual objects in the user's view of the real world, providing a novel visualisation technology for a wide range of application. We, the Augmented Reality Team at VTT, started working in the AR field in 2000 with the development of virtual advertisements for live TV broadcast. In the following years we expanded our expertise to game and entertainment applications.

Currently, our research focuses on applying AR to architecture, building construction, interior design and industrial applications. While we keep on top of the very latest AR research in the world, we also do important basic research ourselves. We have mature technology and tools for marker detection and an emerging toolset for markerless tracking. Furthermore, we are especially proud of being able to port many complex video processing and AR concepts to low-level mobile platforms, including camera phones, where the processing and bandwidth resources are limited.

A general overview of our work can be found in the following slides.

If you can't see menu bar (links) on the left, click here

AR Team web pages (http://www.vtt.fi/multimedia) contain many useful documents of Augmented Reality:

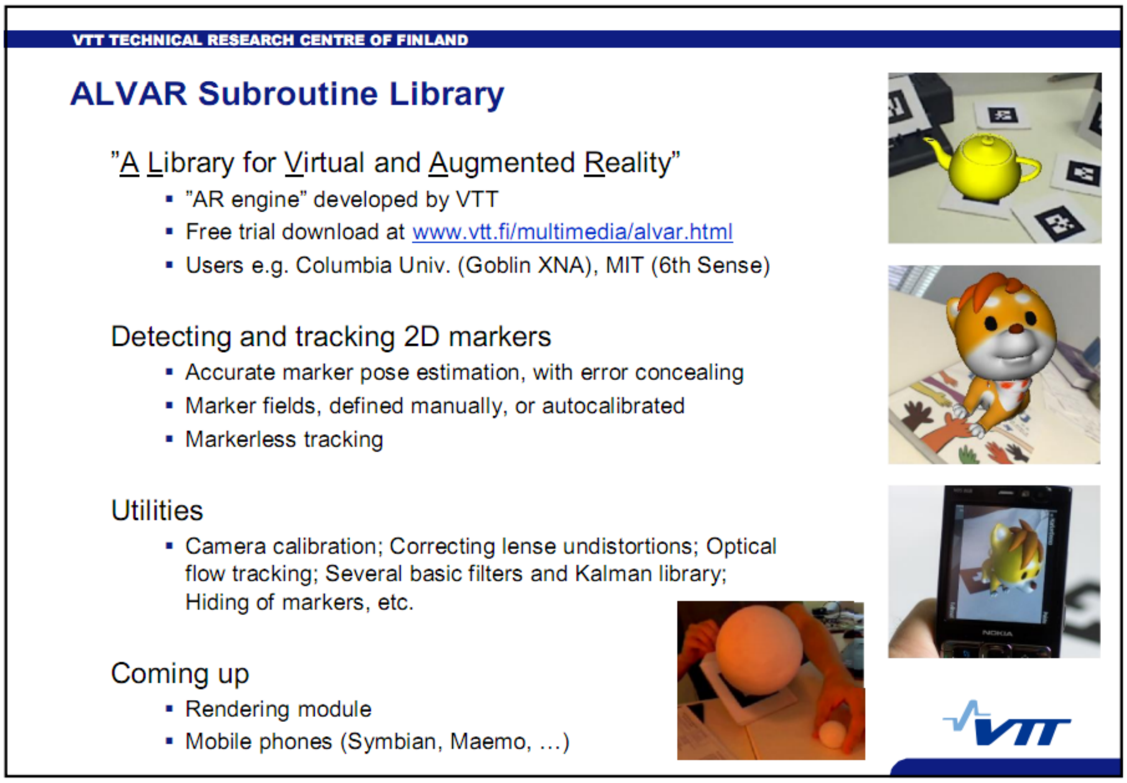
- Description of AR (slide show, pdf),
- Videos of team's AR applications, and
- Downloadable AR demo applications.

A large part of the team's AR know-how (but not all of it) is embodied in the Alvar open-source AR library: A Library for Virtual and Augmented Reality.

1.4 Introduction to ALVAR – A Library for Virtual and Augmented Reality

ALVAR can be downloaded from the AR Team web pages:
http://www.vtt.fi/multimedia/alvar.html

ALVAR is a software library for creating virtual and augmented reality applications. ALVAR has been developed by the VTT Technical Research Centre of Finland.



VTT TECHNICAL RESEARCH CENTRE OF FINLAND

ALVAR Subroutine Library

"A Library for Virtual and Augmented Reality"

- "AR engine" developed by VTT
- Free trial download at www.vtt.fi/multimedia/alvar.html
- Users e.g. Columbia Univ. (Goblin XNA), MIT (6th Sense)

Detecting and tracking 2D markers

- Accurate marker pose estimation, with error concealing
- Marker fields, defined manually, or autocalibrated
- Markerless tracking

Utilities

- Camera calibration; Correcting lense undistortions; Optical flow tracking; Several basic filters and Kalman library; Hiding of markers, etc.

Coming up

- Rendering module
- Mobile phones (Symbian, Maemo, …)

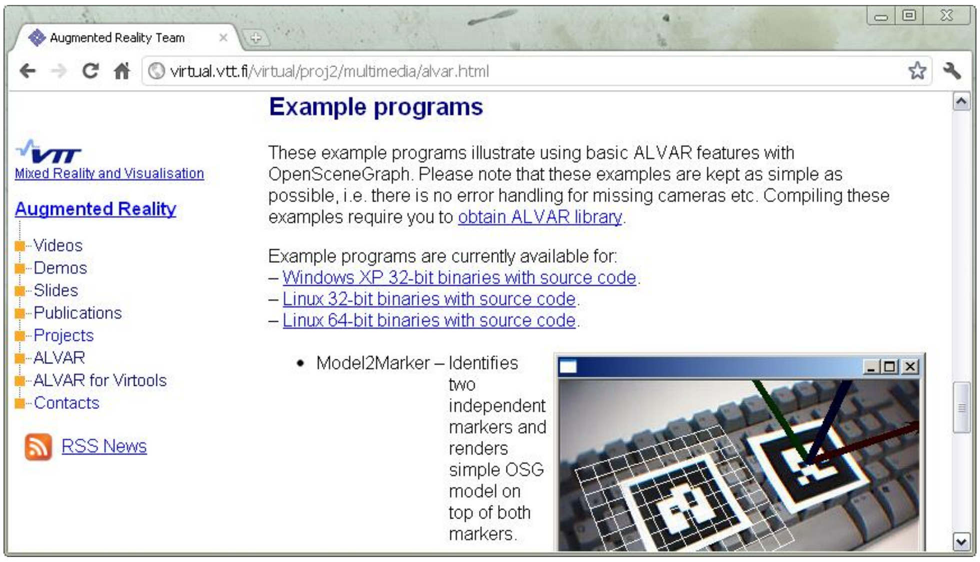
The current version of the library mainly supports marker-based augmented reality applications, but also includes tools for markerless augmented reality.

ALVAR is designed to be as flexible as possible. It offers high-level tools and methods for creating augmented reality applications with just a few lines of code. The library also includes interfaces for all of the low-level tools and methods, which makes it possible for the user to develop their own solutions using alternative approaches or completely new algorithms.

ALVAR is currently provided on Windows and Linux operating systems and requires only one third party library (OpenCV). ALVAR is independent of any graphical libraries and can be easily integrated.

On the other hand, this implies that ALVAR itself contains no support for 3D graphics or 3D models – these must be implemented using other software libraries.

There are separate demo programs (binaries and source code) that use OpenSceneGraph for 3D graphics.



Example programs

These example programs illustrate using basic ALVAR features with OpenSceneGraph. Please note that these examples are kept as simple as possible, i.e. there is no error handling for missing cameras etc. Compiling these examples require you to obtain ALVAR library.

Example programs are currently available for:
– Windows XP 32-bit binaries with source code.
– Linux 32-bit binaries with source code.
– Linux 64-bit binaries with source code.

- Model2Marker – Identifies two independent markers and renders simple OSG model on top of both markers.

1.4.1 ALVAR Desktop License

ALVAR Desktop license is GNU LGPL v.2.1. Thus ALVAR Desktop is free for both commercial and non-commercial use.

ALVAR 2.0 is distributed under the terms of the GNU Lesser General Public License (LGPL) version 2.1 or later. See http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html for the license terms. By downloading Alvar 2.0, you agree to be bound by the terms of the GNU LGPL version 2.1 or later.

Other versions of ALVAR (e.g. ALVAR Mobile) are commercial.

2 ALVAR Features

Detecting and tracking 2D markers. Currently two types of square matrix markers are supported. Custom marker types can easily be added. ALVAR keeps the marker pose estimation as accurate as possible. Furthermore, ALVAR uses some tracking heuristics to identify markers that are "too far" and to recover from occlusions in the multimarker case for example.

Using a setup of multiple markers for pose detection. The marker setup coordinates can be set manually or they can be automatically deduced using various methods.

Tools for calibrating a camera. Distorting and undistorting points, projecting points and finding exterior orientation using point-sets.

Hiding markers from the view.

Several basic filters: average, median, running average, double exponential smoothing. Kalman filters for sensor fusion: Kalman filter, extended Kalman filter and unscented Kalman filter.

Several methods for tracking using optical flow.

Markerless tracking using the SfM and Fern's algorithms.

Summary of ALVAR main core features:

- Capture video from USB camera, Firewire camera or AVI file (using plugins).
- Detecting Markers and predefined MultiMarkers. Marker types: ALVAR, ARToolkit, custom
- Filters for data sequences
- Tracking image features
- Camera/Homography methods: Calibrate, Distort, Undistort, CalcExteriorOrientation, ProjectPoints
- Further utils: Threads, Mutex, Histogram, Serialization, Image Labeling, Drawing, HideTexture, …
- Types: Point, Line, Rotation, Pose, Bitset

Summary of ALVAR main advanced features:

- Methods to deduce/optimize MultiMarker setups
- SimpleSfM: Structure for motion to use features in addition to markers. Pose update optimization.
- External container versions of several methods
- Non-linear optimization using Gauss-Newton, Levenberg-Marquardt and Tukey m-estimator
- Kalman filter, EKF, Unscented Kalman filter
- More methods for tracking image features
- Further utils: Container3d, Ransac, TrifocalTensor, IntegralImage, IntegralGradient, …
- Fern's classification framework to enable markerless tracking

2.1 Other AR Technologies, NOT in ALVAR (yet…)

Other Demonstrations of the AR Team (not yet in ALVAR):

- 3D-model based tracking
- Image database e.g. for tracking init/recovery
- Photorealistic rendering
- Plugin interface for external sensors (e.g. inertial measurement unit).

3 ALVAR Packages and Requirements

3.1 The 3 ALVAR Packages

There are three different ALVAR distribution packages, which all can be freely downloaded:

- **Bin** – for those who only want to test AR and ALVAR; contains precompiled binary versions of the ALVAR samples and OSG demos; no C++ compiler nor 3rd-party libraries are required

- **Sdk** – for those who want to build their own AR applications using ALVAR; contains ALVAR header files, precompiled ALVAR libraries, and HTML documentation; a C++ compiler and 3rd-party libraries are required
- **Src** – for those who want to compile ALVAR themselves; contains the raw source code of ALVAR; a C++ compiler and 3rd-party libraries are required

Please note that the 3 distributions are NOT subsets of each other, e.g. HTML documentation is in the **Sdk** distribution only. The ALVAR Presentation and the User’s Manual (this document) must be downloaded separately.

3.2 ALVAR Requirements

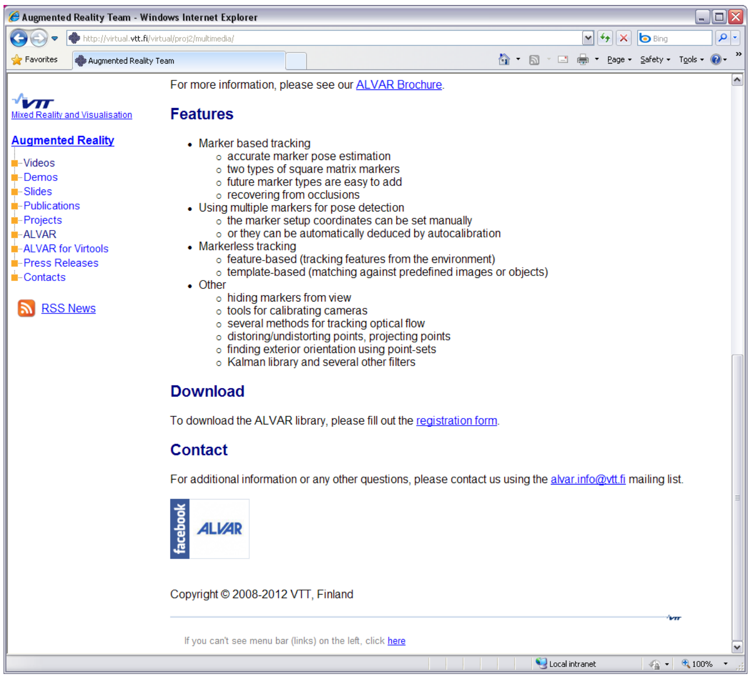
ALVAR has been tested with the following environments: Windows XP 32-bit, Microsoft Visual Studio 2005, 2008 and 2010 (versions 8, 9 and 10). Linux

ALVAR core library requires the following 3rd party library: OpenCV 2.4.0

ALVAR sample code requires: GLUT 3.7.6, CMake 2.8.3

The separate demo programs require: OpenSceneGraph 2.8.4

4 Istalling and Using Alvar



For more information, please see our ALVAR Brochure.

Features

- Marker based tracking
  - accurate marker pose estimation
  - two types of square matrix markers
  - future marker types are easy to add
  - recovering from occlusions
- Using multiple markers for pose detection
  - the marker setup coordinates can be set manually
  - or they can be automatically deduced by autocalibration
- Markerless tracking
  - feature-based (tracking features from the environment)
  - template-based (matching against predefined images or objects)
- Other
  - hiding markers from view
  - tools for calibrating cameras
  - several methods for tracking optical flow
  - distoring/undistorting points, projecting points
  - finding exterior orientation using point-sets
  - Kalman library and several other filters

Download

To download the ALVAR library, please fill out the registration form.

Contact

For additional information or any other questions, please contact us using the alvar.info@vtt.fi mailing list.

Copyright © 2008-2012 VTT, Finland

If you can't see menu bar (links) on the left, click here

4.1 Test ALVAR Only (Bin package)

If you only want to find out what AR and ALVAR is all about, download and install the **Bin** distribution package for your operating system. You can execute the binary versions of the ALVAR Samples and Demos without having to compile anything.

4.2 Basic ALVAR Usage (Sdk package)

Those users who want to develop their own C++ applications using the ALVAR libraries should download and install the ALVAR **Sdk** distribution package.

Ensure that you have a suitable development environment. Currently the library has been used with the following.

- Microsoft Visual Studio 2005, 2008 and 2010
- Linux 32-bit and 64-bit; gcc versions 4.3, 4.4 and 4.5

Install the required 3rd party libraries: OpenCV 2.4.0, CMake 2.8.3, GLUT 3.7.6 (needed only in Samples), OSG 2.8.4 (needed only in Demos).

Install the ALVAR library (**Sdk** package).

Develop your application. Include the needed ALVAR headers in your source directory and link to the ALVAR library matching your development environment (e.g. bin/msvc90/alvar.lib).

Copy the OpenCV (and GLUT) runtime libraries where your application can find them (e.g. the exe-directory).

4.3 Compiling the Alvar Samples (Sdk package)

The most up-to-date instructions of how to compile ALVAR applications are in the 'compiling.txt' file in the ***doc*** *folder. Please refer to that file for more thorough instructions, this section presents just a brief summary.*

Ensure that you have a suitable development environment.

Install the required 3rd party libraries: OpenCV, GLUT, CMake.

Install the ALVAR library.

Note that the CMake binary directory must be in your system's PATH environment variable.

Generate the development environment to build the samples by running the generate-script of your choice (e.g. build/msvc90/generate.bat).

Fill in the missing information for CMake (e.g. GLUT_ROOT_PATH).

Do NOT change the 'where to build binaries' directory.

Use the generated development environment to compile the samples.

(The Demo programs using OpenSceneGraph for 3D graphics are compiled the same way, but in addition OSG 2.8.4 must be downloaded.)

4.4 Advanced: Compile the ALVAR Source Code (Src package)

For those (advanced) users who want to compile the ALVAR source code themselves there is the **Src** distribution package. The HTML Help is generated from the source code using the Doxygen 3rd-party tool.

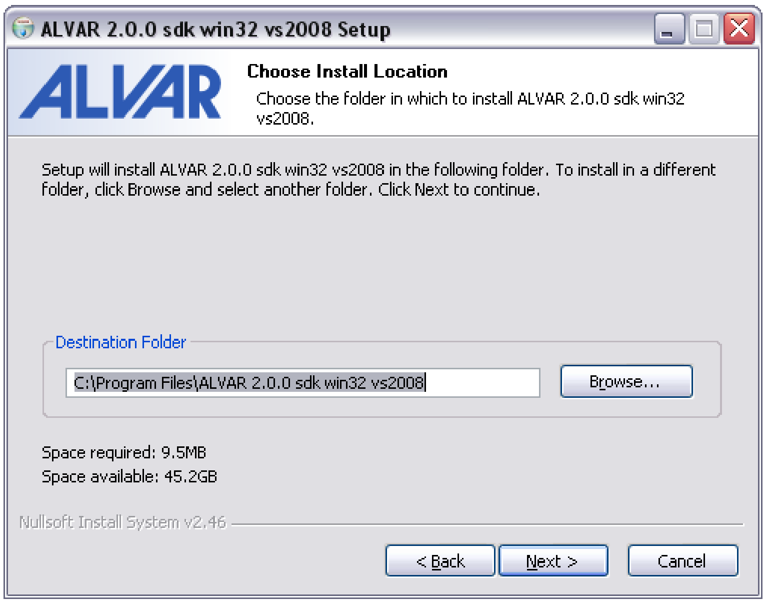
4.5 ALVAR Directories

- **bin** - The compiled binaries will appear in a subdirectory matching the selected build subdirectory
- **build** - The building environment is in a matching subdirectory. See the 'compiling.txt' file in the **doc** folder.
- **data** – Data files used by some of the Samples and Demos (however, the markers are in the *Alvar.pdf* file, which is in the **doc** directory).
- **demo** – Demo applications using OpenSceneGraph for 3D graphics.
- **doc** - Documentation. Generated using Doxygen (e.g. "make doc"). Also contains sample markers in the *Alvar.pdf* file.
- **include** - contains the ALVAR C++ header files (**Sdk** package only).
- **sample** - Samples that demonstrate how to use the library.
- **src** - Sources for the ALVAR library (**Src** package only). Note that Alvar.h is different from the others; it is generated separately for each build environment based on Alvar.h.cmake.

4.6 Installing ALVAR: Step by Step

The installation package **alvar-2.0.0-sdk-win32-vs2008.exe** was executed.

First there is a welcome screen, then a license acceptance screen, and then the user is prompted for installation location:


ALVAR 2.0.0 sdk win32 vs2008 Setup

Choose Install Location

Choose the folder in which to install ALVAR 2.0.0 sdk win32 vs2008.

Setup will install ALVAR 2.0.0 sdk win32 vs2008 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

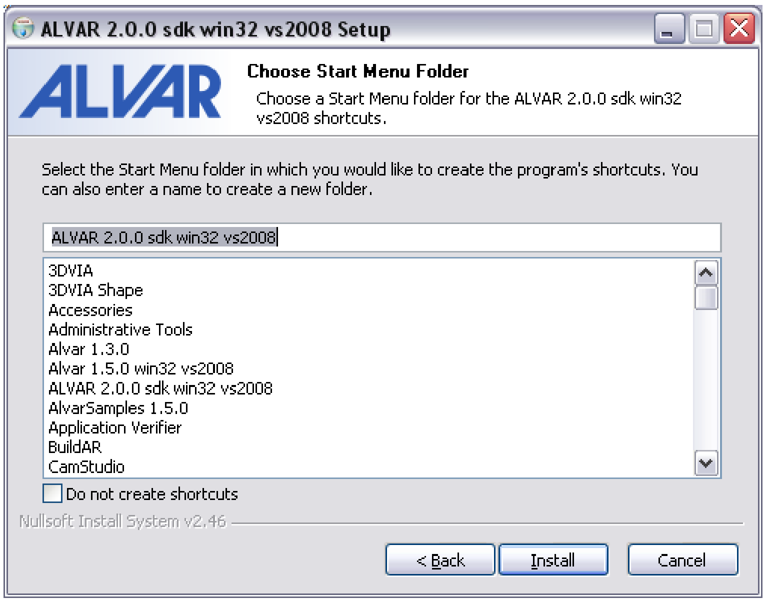
Destination Folder

C:\Program Files\ALVAR 2.0.0 sdk win32 vs2008

Space required: 9.5MB

Space available: 45.2GB

Nullsoft Install System v2.46


ALVAR 2.0.0 sdk win32 vs2008 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the ALVAR 2.0.0 sdk win32 vs2008 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

ALVAR 2.0.0 sdk win32 vs2008

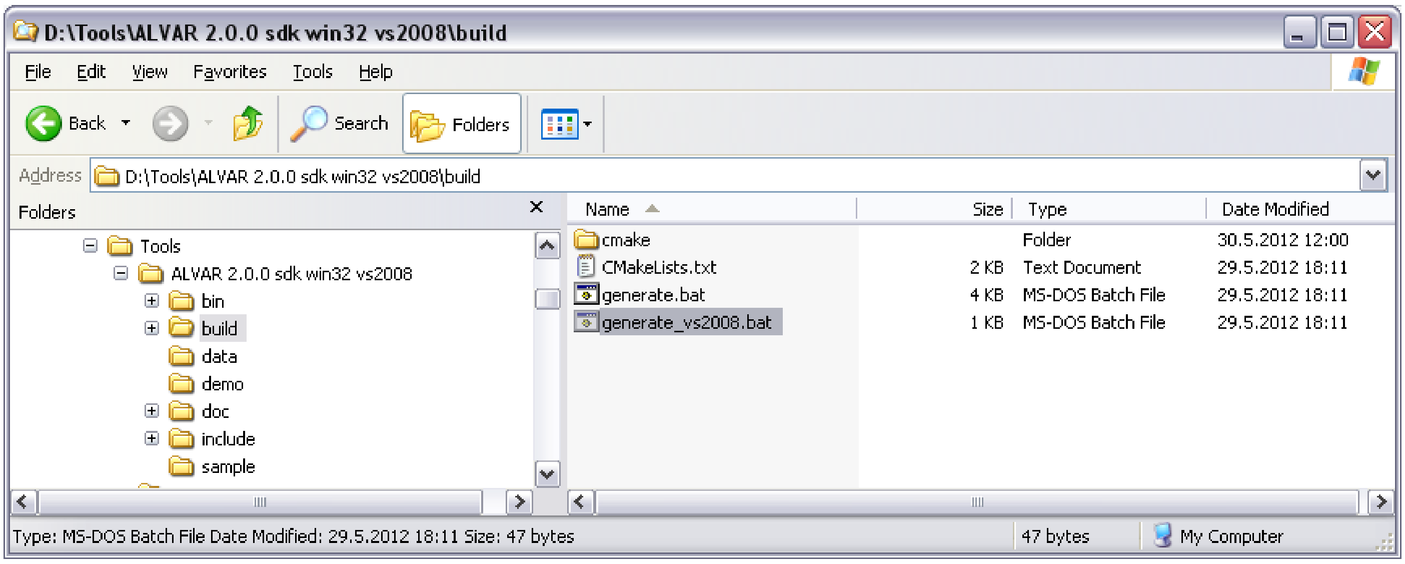
3DVIA
3DVIA Shape
Accessories
Administrative Tools
Alvar 1.3.0
Alvar 1.5.0 win32 vs2008
ALVAR 2.0.0 sdk win32 vs2008
AlvarSamples 1.5.0
Application Verifier
BuildAR
CamStudio

☐ Do not create shortcuts

Nullsoft Install System v2.46

In the following screen dumps, the Destination Folder of the installation was set to: D:\Tools\ALVAR 2.0.0 sdk win32 vs2008

In subfolder **build** the script **generate_vs2008.bat** was executed.



D:\Tools\ALVAR 2.0.0 sdk win32 vs2008\build

Address: D:\Tools\ALVAR 2.0.0 sdk win32 vs2008\build

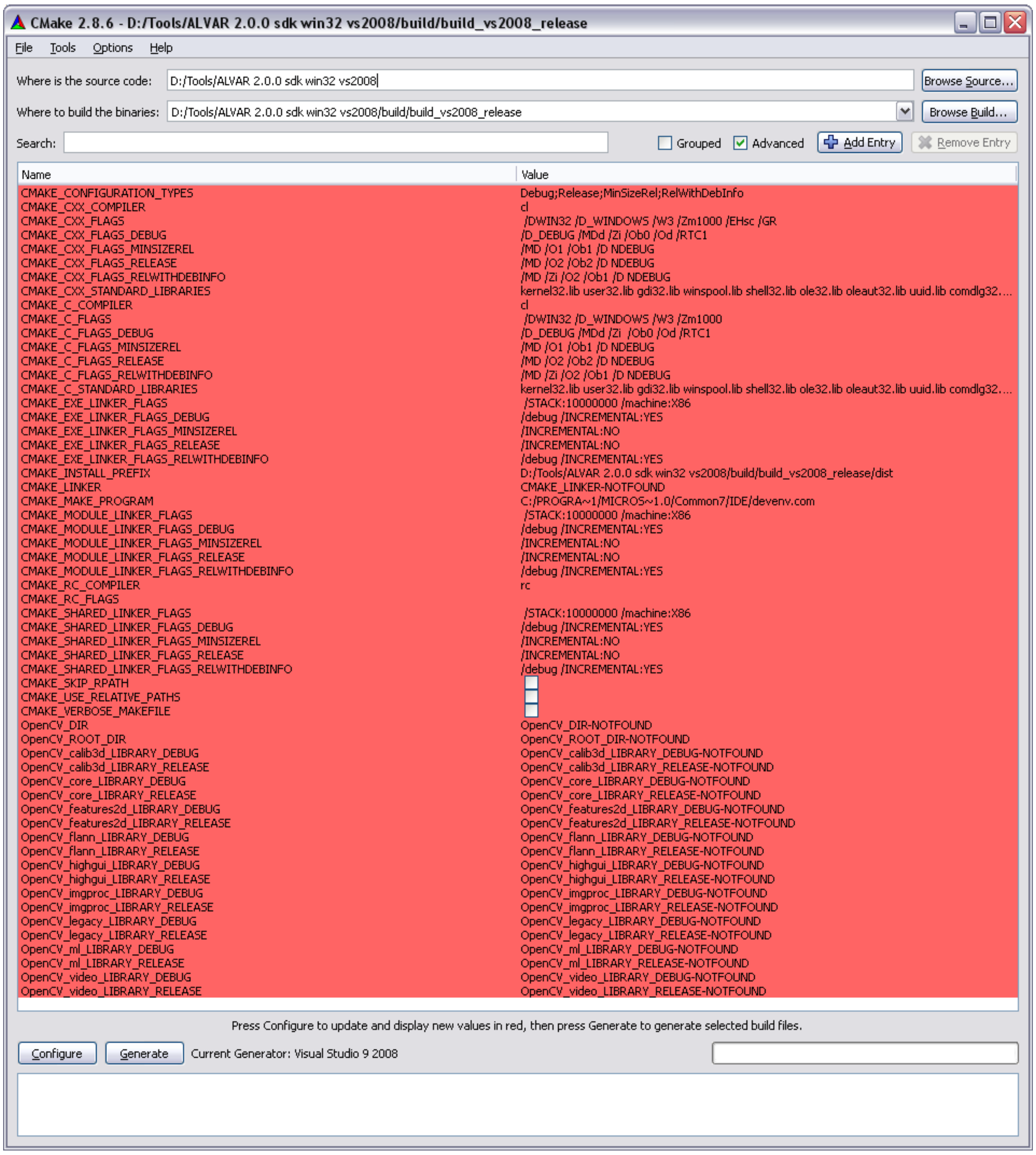
Folders:

- Tools
  - ALVAR 2.0.0 sdk win32 vs2008
    - bin
    - build
    - data
    - demo
    - doc
    - include
    - sample

Name	Size	Type	Date Modified
cmake		Folder	30.5.2012 12:00
CMakeLists.txt	2 KB	Text Document	29.5.2012 18:11
generate.bat	4 KB	MS-DOS Batch File	29.5.2012 18:11
generate_vs2008.bat	1 KB	MS-DOS Batch File	29.5.2012 18:11

Type: MS-DOS Batch File Date Modified: 29.5.2012 18:11 Size: 47 bytes   47 bytes   My Computer

The *.bat starts the CMake 2.8 program. The user must enter the proper paths. The initial values are usually good, so the first thing to do is to press the button **Configure**.



CMake 2.8.6 - D:/Tools/ALVAR 2.0.0 sdk win32 vs2008/build/build_vs2008_release

Where is the source code: D:/Tools/ALVAR 2.0.0 sdk win32 vs2008

Where to build the binaries: D:/Tools/ALVAR 2.0.0 sdk win32 vs2008/build/build_vs2008_release

Search: ☐ Grouped ☑ Advanced

Name	Value
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_CXX_COMPILER	cl
CMAKE_CXX_FLAGS	/DWIN32 /D_WINDOWS /W3 /Zm1000 /EHsc /GR
CMAKE_CXX_FLAGS_DEBUG	/D_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_CXX_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /D NDEBUG
CMAKE_CXX_FLAGS_RELEASE	/MD /O2 /Ob2 /D NDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /D NDEBUG
CMAKE_CXX_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg32....
CMAKE_C_COMPILER	cl
CMAKE_C_FLAGS	/DWIN32 /D_WINDOWS /W3 /Zm1000
CMAKE_C_FLAGS_DEBUG	/D_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_C_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /D NDEBUG
CMAKE_C_FLAGS_RELEASE	/MD /O2 /Ob2 /D NDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /D NDEBUG
CMAKE_C_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg32....
CMAKE_EXE_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_EXE_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_EXE_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_INSTALL_PREFIX	D:/Tools/ALVAR 2.0.0 sdk win32 vs2008/build/build_vs2008_release/dist
CMAKE_LINKER	CMAKE_LINKER-NOTFOUND
CMAKE_MAKE_PROGRAM	C:/PROGRA~1/MICROS~1.0/Common7/IDE/devenv.com
CMAKE_MODULE_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_MODULE_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_MODULE_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_RC_COMPILER	rc
CMAKE_RC_FLAGS	
CMAKE_SHARED_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_SHARED_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_SHARED_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_SHARED_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_SHARED_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_SKIP_RPATH	☐
CMAKE_USE_RELATIVE_PATHS	☐
CMAKE_VERBOSE_MAKEFILE	☐
OpenCV_DIR	OpenCV_DIR-NOTFOUND
OpenCV_ROOT_DIR	OpenCV_ROOT_DIR-NOTFOUND
OpenCV_calib3d_LIBRARY_DEBUG	OpenCV_calib3d_LIBRARY_DEBUG-NOTFOUND
OpenCV_calib3d_LIBRARY_RELEASE	OpenCV_calib3d_LIBRARY_RELEASE-NOTFOUND
OpenCV_core_LIBRARY_DEBUG	OpenCV_core_LIBRARY_DEBUG-NOTFOUND
OpenCV_core_LIBRARY_RELEASE	OpenCV_core_LIBRARY_RELEASE-NOTFOUND
OpenCV_features2d_LIBRARY_DEBUG	OpenCV_features2d_LIBRARY_DEBUG-NOTFOUND
OpenCV_features2d_LIBRARY_RELEASE	OpenCV_features2d_LIBRARY_RELEASE-NOTFOUND
OpenCV_flann_LIBRARY_DEBUG	OpenCV_flann_LIBRARY_DEBUG-NOTFOUND
OpenCV_flann_LIBRARY_RELEASE	OpenCV_flann_LIBRARY_RELEASE-NOTFOUND
OpenCV_highgui_LIBRARY_DEBUG	OpenCV_highgui_LIBRARY_DEBUG-NOTFOUND
OpenCV_highgui_LIBRARY_RELEASE	OpenCV_highgui_LIBRARY_RELEASE-NOTFOUND
OpenCV_imgproc_LIBRARY_DEBUG	OpenCV_imgproc_LIBRARY_DEBUG-NOTFOUND
OpenCV_imgproc_LIBRARY_RELEASE	OpenCV_imgproc_LIBRARY_RELEASE-NOTFOUND
OpenCV_legacy_LIBRARY_DEBUG	OpenCV_legacy_LIBRARY_DEBUG-NOTFOUND
OpenCV_legacy_LIBRARY_RELEASE	OpenCV_legacy_LIBRARY_RELEASE-NOTFOUND
OpenCV_ml_LIBRARY_DEBUG	OpenCV_ml_LIBRARY_DEBUG-NOTFOUND
OpenCV_ml_LIBRARY_RELEASE	OpenCV_ml_LIBRARY_RELEASE-NOTFOUND
OpenCV_video_LIBRARY_DEBUG	OpenCV_video_LIBRARY_DEBUG-NOTFOUND
OpenCV_video_LIBRARY_RELEASE	OpenCV_video_LIBRARY_RELEASE-NOTFOUND

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

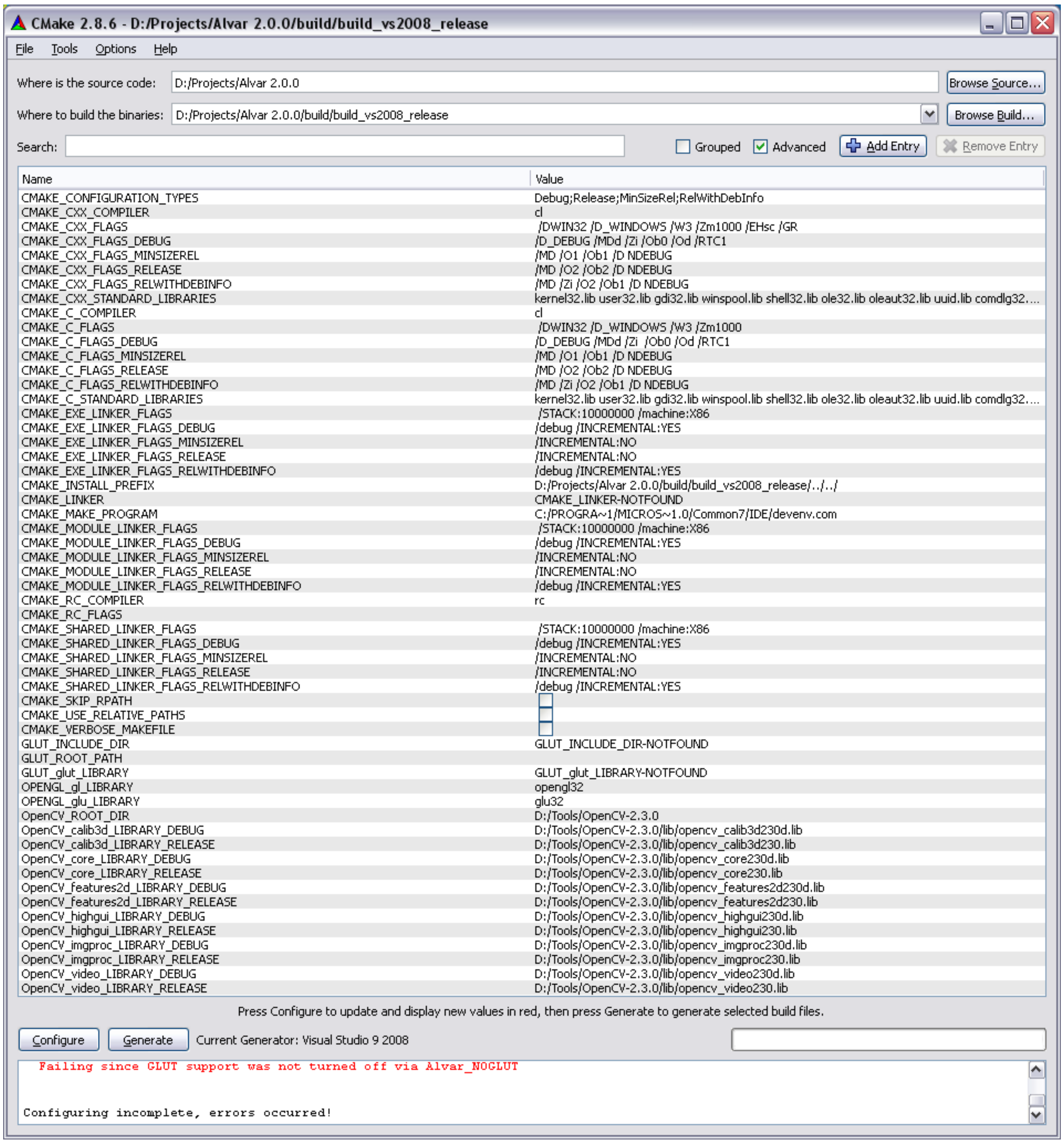
The following sceen dumps of this section were generated with an earlier VTT internal development version called ***2.0.0.794e832****. This ALVAR version used earlier versions of the 3rd-party libraries, namely CMake 2.8, OpenCV 2.3, GLUT 3.7, and OpenSceneGraph 2.8.0.*

The example PC had Windows XP operating system, MS Visual Studio 2008 SP1. CMake 2.8, OpenCV 2.3, GLUT 3.7, and OpenSceneGraph 2.8.0 had been installed earlier.

Also the Destination Folder of the installation was different, it was set to:

D:\Projects\Alvar 2.0.0

(Please remember that the current version of the 3rd-party libraries are: CMake 2.8.3, OpenCV 2.4.0, GLUT 3.7.6, and OpenSceneGraph 2.8.4.)



Where is the source code: D:/Projects/Alvar 2.0.0

Where to build the binaries: D:/Projects/Alvar 2.0.0/build/build_vs2008_release

Search:

[ ] Grouped [✓] Advanced

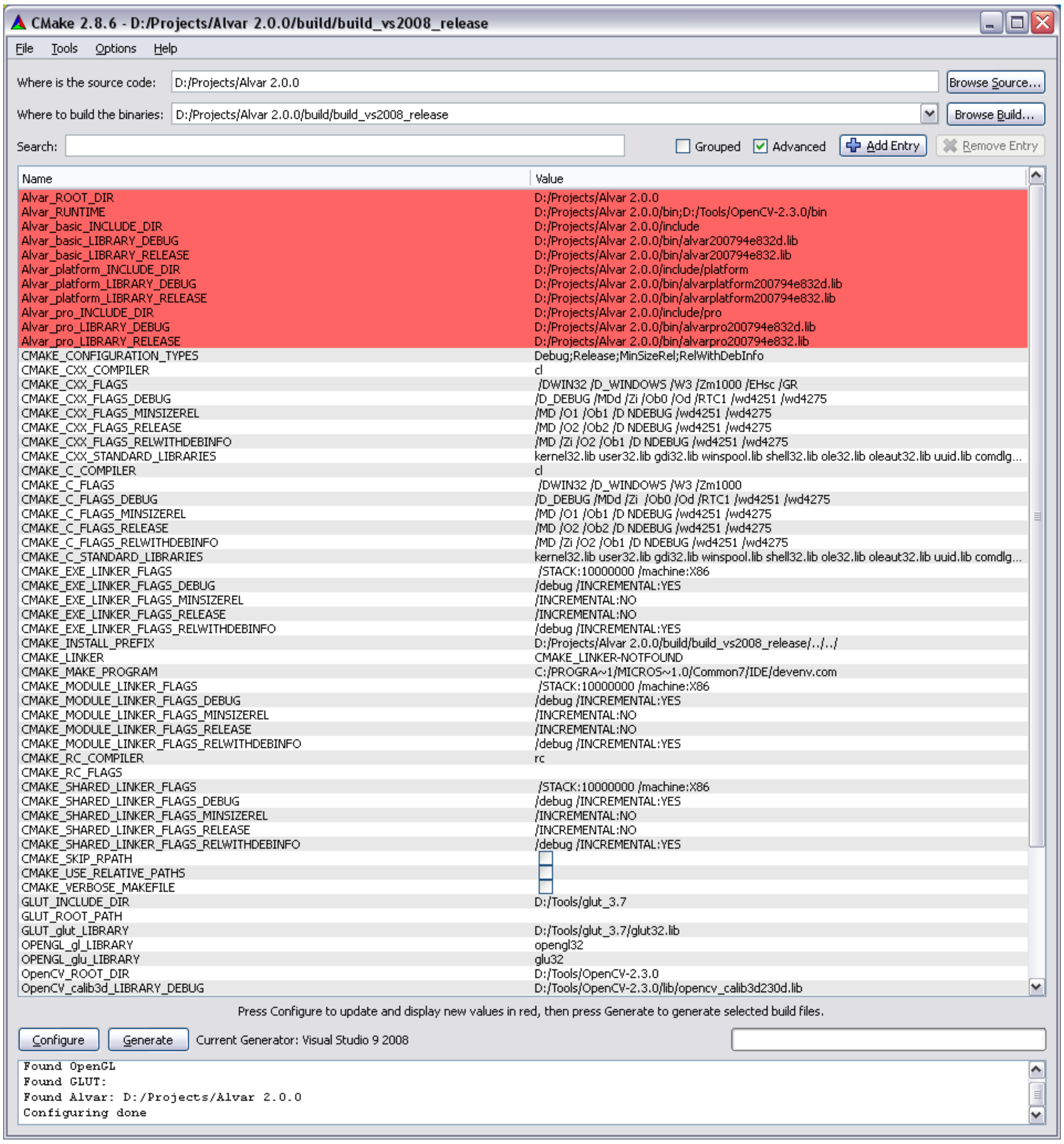
Name	Value
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_CXX_COMPILER	cl
CMAKE_CXX_FLAGS	/DWIN32 /D_WINDOWS /W3 /Zm1000 /EHsc /GR
CMAKE_CXX_FLAGS_DEBUG	/D_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_CXX_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /D NDEBUG
CMAKE_CXX_FLAGS_RELEASE	/MD /O2 /Ob2 /D NDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /D NDEBUG
CMAKE_CXX_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg32....
CMAKE_C_COMPILER	cl
CMAKE_C_FLAGS	/DWIN32 /D_WINDOWS /W3 /Zm1000
CMAKE_C_FLAGS_DEBUG	/D_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_C_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /D NDEBUG
CMAKE_C_FLAGS_RELEASE	/MD /O2 /Ob2 /D NDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /D NDEBUG
CMAKE_C_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg32....
CMAKE_EXE_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_EXE_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_EXE_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_INSTALL_PREFIX	D:/Projects/Alvar 2.0.0/build/build_vs2008_release/../../
CMAKE_LINKER	CMAKE_LINKER-NOTFOUND
CMAKE_MAKE_PROGRAM	C:/PROGRA~1/MICROS~1.0/Common7/IDE/devenv.com
CMAKE_MODULE_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_MODULE_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_MODULE_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_RC_COMPILER	rc
CMAKE_RC_FLAGS	
CMAKE_SHARED_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_SHARED_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_SHARED_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_SHARED_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_SHARED_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_SKIP_RPATH	☐
CMAKE_USE_RELATIVE_PATHS	☐
CMAKE_VERBOSE_MAKEFILE	☐
GLUT_INCLUDE_DIR	GLUT_INCLUDE_DIR-NOTFOUND
GLUT_ROOT_PATH	
GLUT_glut_LIBRARY	GLUT_glut_LIBRARY-NOTFOUND
OPENGL_gl_LIBRARY	opengl32
OPENGL_glu_LIBRARY	glu32
OpenCV_ROOT_DIR	D:/Tools/OpenCV-2.3.0
OpenCV_calib3d_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_calib3d230d.lib
OpenCV_calib3d_LIBRARY_RELEASE	D:/Tools/OpenCV-2.3.0/lib/opencv_calib3d230.lib
OpenCV_core_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_core230d.lib
OpenCV_core_LIBRARY_RELEASE	D:/Tools/OpenCV-2.3.0/lib/opencv_core230.lib
OpenCV_features2d_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_features2d230d.lib
OpenCV_features2d_LIBRARY_RELEASE	D:/Tools/OpenCV-2.3.0/lib/opencv_features2d230.lib
OpenCV_highgui_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_highgui230d.lib
OpenCV_highgui_LIBRARY_RELEASE	D:/Tools/OpenCV-2.3.0/lib/opencv_highgui230.lib
OpenCV_imgproc_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_imgproc230d.lib
OpenCV_imgproc_LIBRARY_RELEASE	D:/Tools/OpenCV-2.3.0/lib/opencv_imgproc230.lib
OpenCV_video_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_video230d.lib
OpenCV_video_LIBRARY_RELEASE	D:/Tools/OpenCV-2.3.0/lib/opencv_video230.lib

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
  Failing since GLUT support was not turned off via Alvar_NOGLUT


Configuring incomplete, errors occurred!
```

GLUT 3.7 path had to be entered (search for entries **…-NOTFOUND** in the right-hand column). The entries were clicked and the proper paths were entered. After that a new click of button **Configure**.



Where is the source code: D:/Projects/Alvar 2.0.0

Where to build the binaries: D:/Projects/Alvar 2.0.0/build/build_vs2008_release

Search:

☐ Grouped ✓ Advanced

Name	Value
Alvar_ROOT_DIR	D:/Projects/Alvar 2.0.0
Alvar_RUNTIME	D:/Projects/Alvar 2.0.0/bin;D:/Tools/OpenCV-2.3.0/bin
Alvar_basic_INCLUDE_DIR	D:/Projects/Alvar 2.0.0/include
Alvar_basic_LIBRARY_DEBUG	D:/Projects/Alvar 2.0.0/bin/alvar200794e832d.lib
Alvar_basic_LIBRARY_RELEASE	D:/Projects/Alvar 2.0.0/bin/alvar200794e832.lib
Alvar_platform_INCLUDE_DIR	D:/Projects/Alvar 2.0.0/include/platform
Alvar_platform_LIBRARY_DEBUG	D:/Projects/Alvar 2.0.0/bin/alvarplatform200794e832d.lib
Alvar_platform_LIBRARY_RELEASE	D:/Projects/Alvar 2.0.0/bin/alvarplatform200794e832.lib
Alvar_pro_INCLUDE_DIR	D:/Projects/Alvar 2.0.0/include/pro
Alvar_pro_LIBRARY_DEBUG	D:/Projects/Alvar 2.0.0/bin/alvarpro200794e832d.lib
Alvar_pro_LIBRARY_RELEASE	D:/Projects/Alvar 2.0.0/bin/alvarpro200794e832.lib
CMAKE_CONFIGURATION_TYPES	Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_CXX_COMPILER	cl
CMAKE_CXX_FLAGS	/DWIN32 /D_WINDOWS /W3 /Zm1000 /EHsc /GR
CMAKE_CXX_FLAGS_DEBUG	/D_DEBUG /MDd /Zi /Ob0 /Od /RTC1 /wd4251 /wd4275
CMAKE_CXX_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /D NDEBUG /wd4251 /wd4275
CMAKE_CXX_FLAGS_RELEASE	/MD /O2 /Ob2 /D NDEBUG /wd4251 /wd4275
CMAKE_CXX_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /D NDEBUG /wd4251 /wd4275
CMAKE_CXX_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg...
CMAKE_C_COMPILER	cl
CMAKE_C_FLAGS	/DWIN32 /D_WINDOWS /W3 /Zm1000
CMAKE_C_FLAGS_DEBUG	/D_DEBUG /MDd /Zi /Ob0 /Od /RTC1 /wd4251 /wd4275
CMAKE_C_FLAGS_MINSIZEREL	/MD /O1 /Ob1 /D NDEBUG /wd4251 /wd4275
CMAKE_C_FLAGS_RELEASE	/MD /O2 /Ob2 /D NDEBUG /wd4251 /wd4275
CMAKE_C_FLAGS_RELWITHDEBINFO	/MD /Zi /O2 /Ob1 /D NDEBUG /wd4251 /wd4275
CMAKE_C_STANDARD_LIBRARIES	kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.lib uuid.lib comdlg...
CMAKE_EXE_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_EXE_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_EXE_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_EXE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_INSTALL_PREFIX	D:/Projects/Alvar 2.0.0/build/build_vs2008_release/../../
CMAKE_LINKER	CMAKE_LINKER-NOTFOUND
CMAKE_MAKE_PROGRAM	C:/PROGRA~1/MICROS~1.0/Common7/IDE/devenv.com
CMAKE_MODULE_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_MODULE_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_MODULE_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_MODULE_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_RC_COMPILER	rc
CMAKE_RC_FLAGS	
CMAKE_SHARED_LINKER_FLAGS	/STACK:10000000 /machine:X86
CMAKE_SHARED_LINKER_FLAGS_DEBUG	/debug /INCREMENTAL:YES
CMAKE_SHARED_LINKER_FLAGS_MINSIZEREL	/INCREMENTAL:NO
CMAKE_SHARED_LINKER_FLAGS_RELEASE	/INCREMENTAL:NO
CMAKE_SHARED_LINKER_FLAGS_RELWITHDEBINFO	/debug /INCREMENTAL:YES
CMAKE_SKIP_RPATH	☐
CMAKE_USE_RELATIVE_PATHS	☐
CMAKE_VERBOSE_MAKEFILE	☐
GLUT_INCLUDE_DIR	D:/Tools/glut_3.7
GLUT_ROOT_PATH	
GLUT_glut_LIBRARY	D:/Tools/glut_3.7/glut32.lib
OPENGL_gl_LIBRARY	opengl32
OPENGL_glu_LIBRARY	glu32
OpenCV_ROOT_DIR	D:/Tools/OpenCV-2.3.0
OpenCV_calib3d_LIBRARY_DEBUG	D:/Tools/OpenCV-2.3.0/lib/opencv_calib3d230d.lib

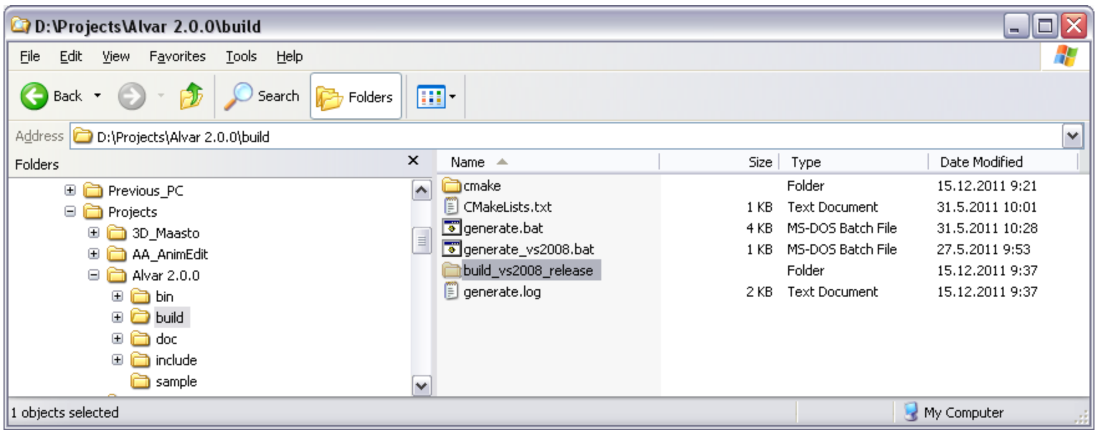
Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 9 2008

```
Found OpenGL
Found GLUT:
Found Alvar: D:/Projects/Alvar 2.0.0
Configuring done
```

And now we are this far, the Alvar installation folder was detected correctly. So now the button **Generate** can be pressed.

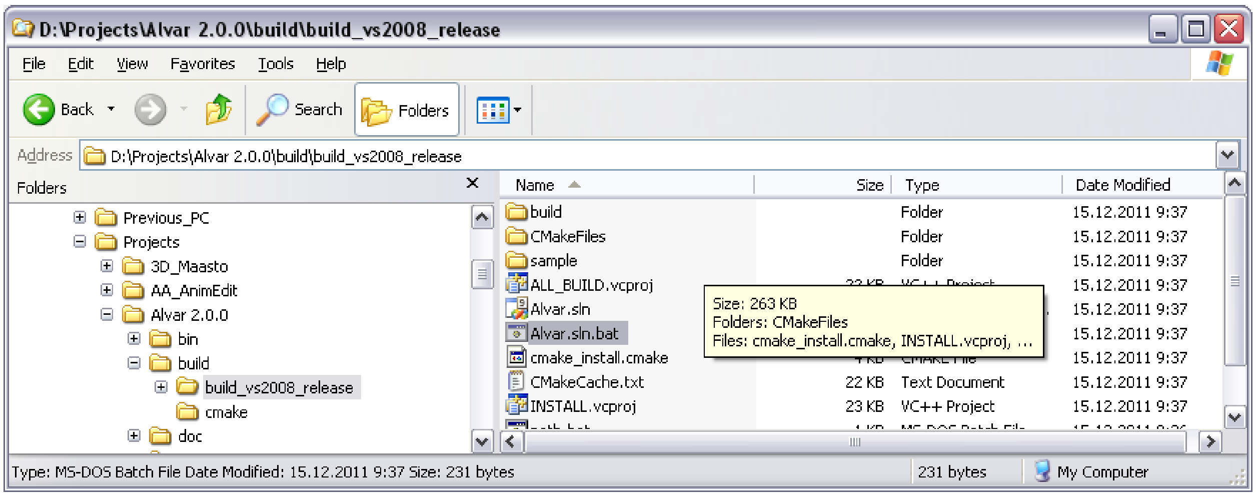
A new subfolder to folder **build** was generated: **build_vs2008_release**



D:\Projects\Alvar 2.0.0\build

Name	Size	Type	Date Modified
cmake		Folder	15.12.2011 9:21
CMakeLists.txt	1 KB	Text Document	31.5.2011 10:01
generate.bat	4 KB	MS-DOS Batch File	31.5.2011 10:28
generate_vs2008.bat	1 KB	MS-DOS Batch File	27.5.2011 9:53
build_vs2008_release		Folder	15.12.2011 9:37
generate.log	2 KB	Text Document	15.12.2011 9:37

In this subfolder there is the batch file **Alvar.sln.bat**, which is used to start Visual Studio 2008. Do not start **Alvar.sln** directly, or the correct environment variables are not set up.



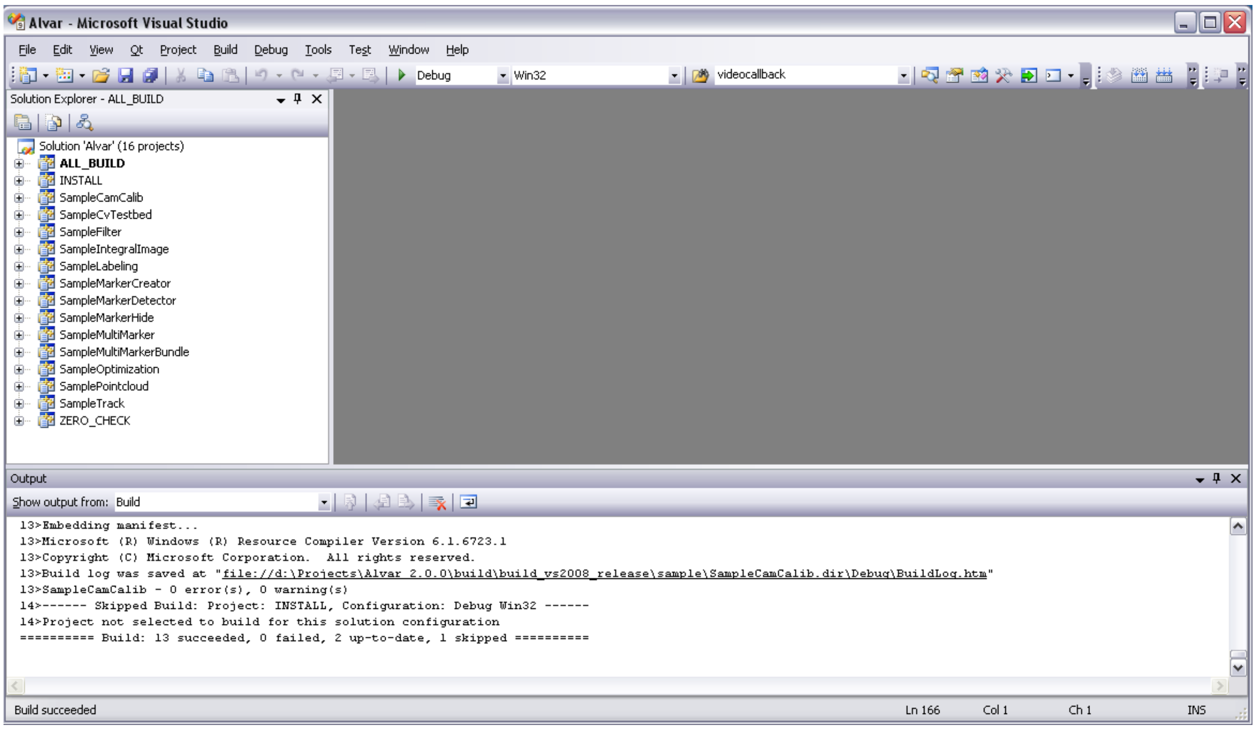
D:\Projects\Alvar 2.0.0\build\build_vs2008_release

Name	Size	Type	Date Modified
build		Folder	15.12.2011 9:37
CMakeFiles		Folder	15.12.2011 9:37
sample		Folder	15.12.2011 9:37
ALL_BUILD.vcproj			15.12.2011 9:37
Alvar.sln			15.12.2011 9:37
Alvar.sln.bat			15.12.2011 9:37
cmake_install.cmake			15.12.2011 9:37
CMakeCache.txt	22 KB	Text Document	15.12.2011 9:37
INSTALL.vcproj	23 KB	VC++ Project	15.12.2011 9:37

Size: 263 KB
Folders: CMakeFiles
Files: cmake_install.cmake, INSTALL.vcproj, ...

Type: MS-DOS Batch File Date Modified: 15.12.2011 9:37 Size: 231 bytes

Then enter the menu command **Build** – **Build Solution**.

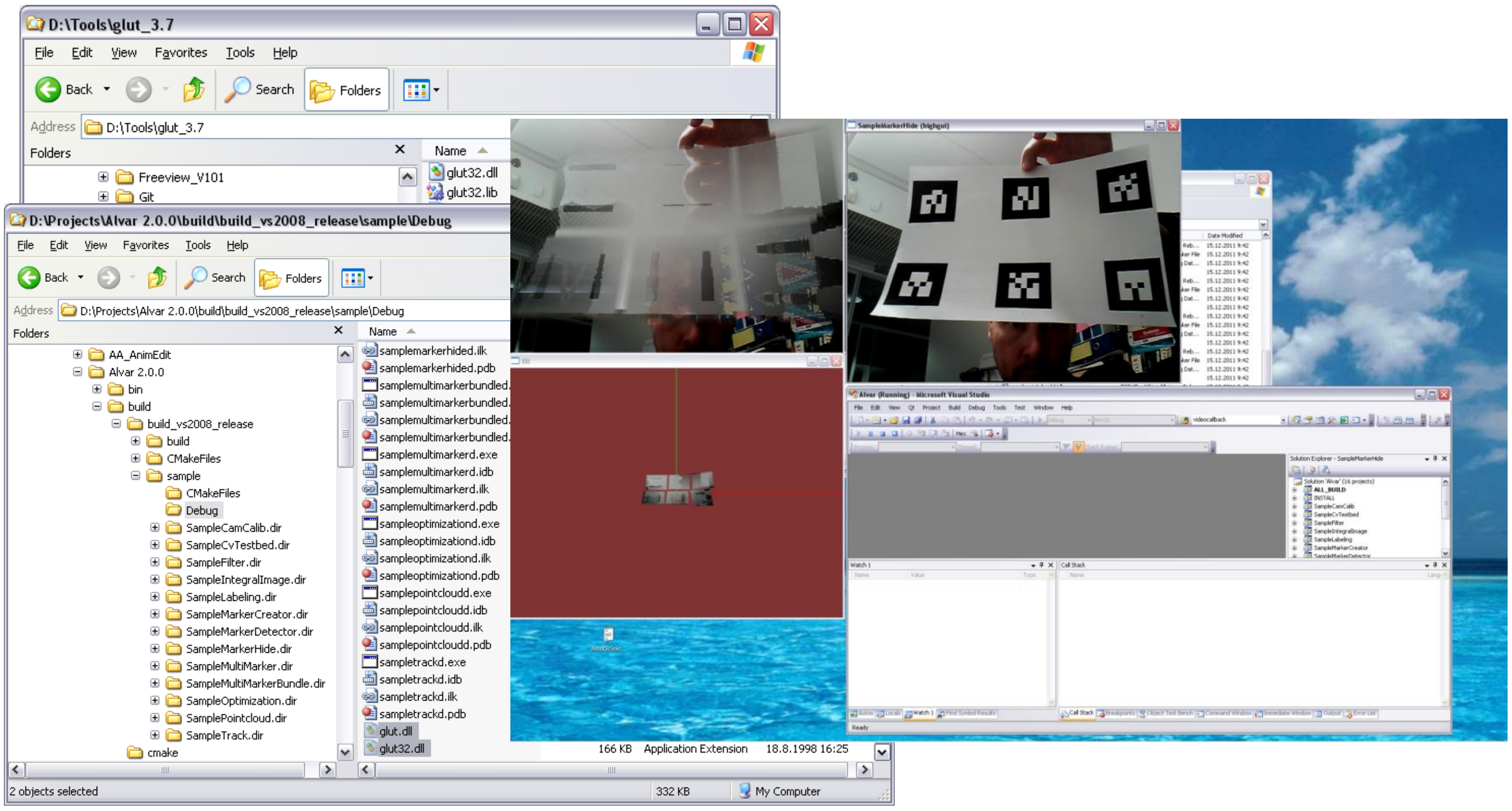


Solution 'Alvar' (16 projects)
- **ALL_BUILD**
- INSTALL
- SampleCamCalib
- SampleCvTestbed
- SampleFilter
- SampleIntegralImage
- SampleLabeling
- SampleMarkerCreator
- SampleMarkerDetector
- SampleMarkerHide
- SampleMultiMarker
- SampleMultiMarkerBundle
- SampleOptimization
- SamplePointcloud
- SampleTrack
- ZERO_CHECK

```
13>Embedding manifest...
13>Microsoft (R) Windows (R) Resource Compiler Version 6.1.6723.1
13>Copyright (C) Microsoft Corporation.  All rights reserved.
13>Build log was saved at "file://d:\Projects\Alvar 2.0.0\build\build_vs2008_release\sample\SampleCamCalib.dir\Debug\BuildLog.htm"
13>SampleCamCalib - 0 error(s), 0 warning(s)
14>------ Skipped Build: Project: INSTALL, Configuration: Debug Win32 ------
14>Project not selected to build for this solution configuration
========== Build: 13 succeeded, 0 failed, 2 up-to-date, 1 skipped ==========
```

Build succeeded

Print markers, try to execute e.g. **SampleMarkerHide**. In this installation the execution failed because VS2008 could not find **glut32.dll**. These DLLs were copied from the GLUT folder to the **Debug** folder.



D:\Tools\glut_3.7

Address D:\Tools\glut_3.7

Folders: Freeview_V101, Git

Name: glut32.dll, glut32.lib

D:\Projects\Alvar 2.0.0\build\build_vs2008_release\sample\Debug

Address D:\Projects\Alvar 2.0.0\build\build_vs2008_release\sample\Debug

Folders:
- AA_AnimEdit
- Alvar 2.0.0
  - bin
  - build
    - build_vs2008_release
      - build
      - CMakeFiles
      - sample
        - CMakeFiles
        - Debug
        - SampleCamCalib.dir
        - SampleCvTestbed.dir
        - SampleFilter.dir
        - SampleIntegralImage.dir
        - SampleLabeling.dir
        - SampleMarkerCreator.dir
        - SampleMarkerDetector.dir
        - SampleMarkerHide.dir
        - SampleMultiMarker.dir
        - SampleMultiMarkerBundle.dir
        - SampleOptimization.dir
        - SamplePointcloud.dir
        - SampleTrack.dir
  - cmake

Name:
- samplemarkerhided.ilk
- samplemarkerhided.pdb
- samplemultimarkerbundled.
- samplemultimarkerbundled.
- samplemultimarkerbundled.
- samplemultimarkerbundled.
- samplemultimarkerd.exe
- samplemultimarkerd.idb
- samplemultimarkerd.ilk
- samplemultimarkerd.pdb
- sampleoptimizationd.exe
- sampleoptimizationd.idb
- sampleoptimizationd.ilk
- sampleoptimizationd.pdb
- samplepointcloudd.exe
- samplepointcloudd.idb
- samplepointcloudd.ilk
- samplepointcloudd.pdb
- sampletrackd.exe
- sampletrackd.idb
- sampletrackd.ilk
- sampletrackd.pdb
- glut.dll
- glut32.dll

166 KB Application Extension 18.8.1998 16:25

2 objects selected 332 KB My Computer

File Edit View Favorites Tools Help
Back
Search
Folders
SampleMarkerHide (highgui)
Alvar (Running) - Microsoft Visual Studio
Solution Explorer - SampleMarkerHide
Watch 1
Call Stack

After that the sample executed correctly.

5 ALVAR Sample Code

ALVAR sample code requires GLUT 3.7.6 and CMake 2.8.3.

The following table presents a summary of the sample code:

Sample	Description
SampleCamCalib	This is an example of how to use *ProjPoints* and *Camera* classes to perform camera calibration using a chessboard pattern.
SampleCvTestbed	This is an example of how to use the *CvTestbed*, *CaptureFactory* and *Capture* classes in order to make quick OpenCV prototype applications.
SampleFilter	This is an example of how to use various filters: *FilterAverage*, *FilterMedian*, *FilterRunningAverage*, *FilterDoubleExponentialSmoothing*, *Kalman*, *KalmanEkf* and *FilterArray*.
SampleIntegralImage	This is an example of how to use the *IntegralImage* and *IntegralGradient* classes for image gradient analysis.
SampleLabeling	This is an example of how to label images using *LabelingCvSeq*.
SampleMarkerCreator	This is an example that demonstrates the generation of *MarkerData* (or *MarkerArtoolkit*) markers and saving the image using *SaveMarkerImage*.
SampleMarkerDetector	This is an example that shows how to detect *MarkerData* markers and visualize them using *GlutViewer*.
SampleMarkerHide	This is an example that shows how to detect *MarkerData* markers, visualize them using *GlutViewer* and hide them with *BuildHideTexture* and *DrawTexture*.
SampleMarkerlessCreator	This is an example of how to use the *FernImageDetector* class to train a Fern classifier for markerless image-based tracking.
SampleMarkerlessDetector	This is an example of how to use the *FernImageDetector* and *FernPoseEstimator* classes to detect and track an image and visualize it using *GlutViewer*.
SampleMultiMarker	This is an example that demonstrates the use of a preconfigured *MultiMarker* setup.
SampleMultiMarkerBundle	This is an example that automatically recognises and optimizes *MultiMarker* setups using *MultiMarkerBundle*.
SampleOptimization	This is an example of how to use the *Optimization* class by fitting curves of increasing degree to random data.
SamplePointcloud	This is an example showing how to use *SimpleSfM* for tracking the environment using features in addition to *MultiMarker*.
SampleTrack	This is an example that shows how to perform tracking of the optical flow using *TrackerPsa*, *TrackerPsaRot*, *TrackerFeatures*, *TrackerStat* or *TrackerStatRot*.

All 15 sample programs use the header file Shared.h, which defines the functions outputEnumeratedPlugins (print plugins to console), outputEnumeratedDevices (print devices to console), and defaultDevice (find index of highgui device). Many of the sample program projects contain files GlutViewer.h and GlutViewer.cpp, which define the class Drawable and 20 functions in namespace GlutViewer.

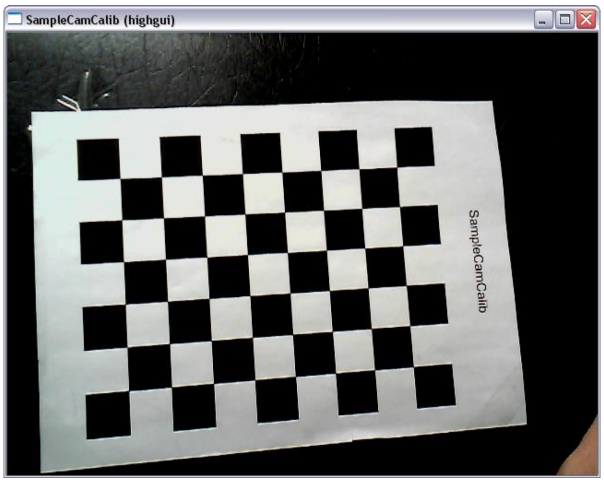
Many of the sample programs also define function videocallback which is called for each video frame by GLUT. Some sample programs also define function keycallback to handle the keyboard presses.

The main program of most samples prints the usage instructions, processes the command-line parameters, sets the videocallback function to be called, enumerates the available plugins, enumerates the available capture devices (and selects one of them to be used), and initializes the capture. After this the camera calibration is loaded, if available, and the video processing is started. The simplest sample program demonstrating this structure is SampleCvTestbed.

Next there is a more thorough presentation of each sample program with screen shots.

5.1 SampleCamCalib

This is an example of how to use *ProjPoints* and *Camera* classes to perform camera calibration using a chessboard pattern. Point the camera to the chessboard calibration pattern (see *Alvar.pdf*) from several directions until 50 calibration images are collected. A '*calib.xml*' file that contains the internal parameters of the camera is generated and can be used by other applications that require a calibrated camera.



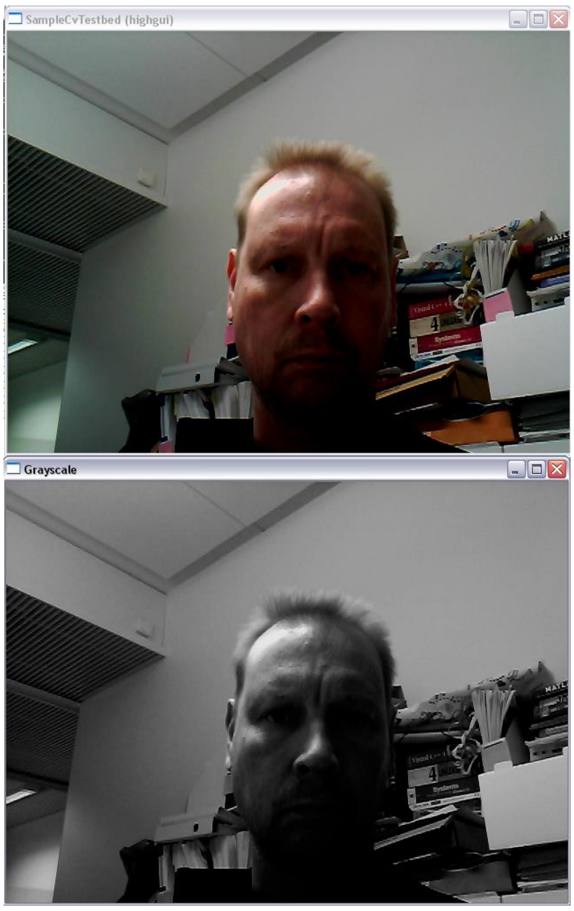
Usage:
  samplecamcalib.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered

Keyboard Shortcuts:
  q: quit

This application is discussed further in section 7.1.

5.2 SampleCvTestbed

This is an example of how to use the *CvTestbed*, *CaptureFactory* and *Capture* classes in order to make quick OpenCV prototype applications. The *CaptureFactory* can create *Capture* objects from many different backends (see SampleCvTestbed.cpp). You can also show/hide the first ten images created using *CvTestbed* using the number keys. In this example you can use key '0' to show/hide a grayscale version of the captured image.

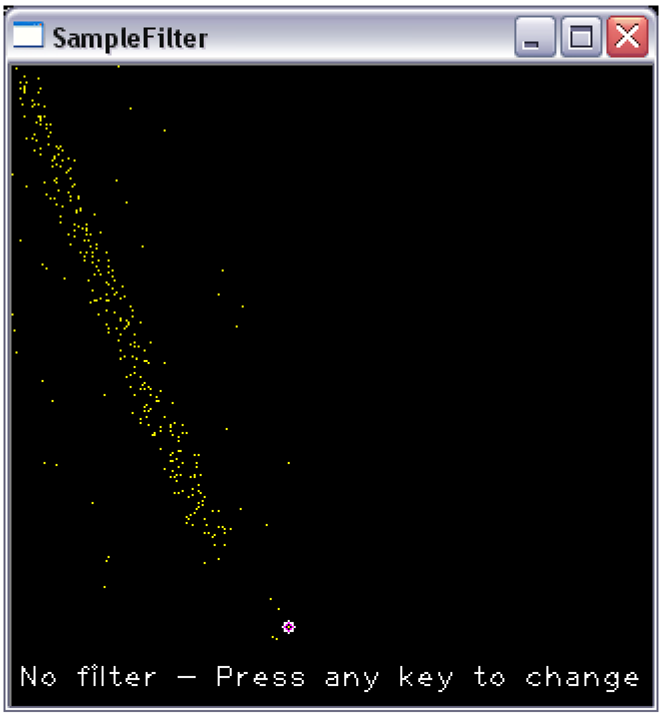


Usage:
  samplecvtestbed.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered

Keyboard Shortcuts:
  0: show/hide grayscale image
  q: quit

5.3 SampleFilter

This is an example of how to use various filters: *FilterAverage*, *FilterMedian*, *FilterRunningAverage*, *FilterDoubleExponentialSmoothing*, *Kalman*, *KalmanEkf* and *FilterArray*. First the example shows unfiltered test data with outliers. The data is then filtered using the various filters. Press any key to cycle through the filters.



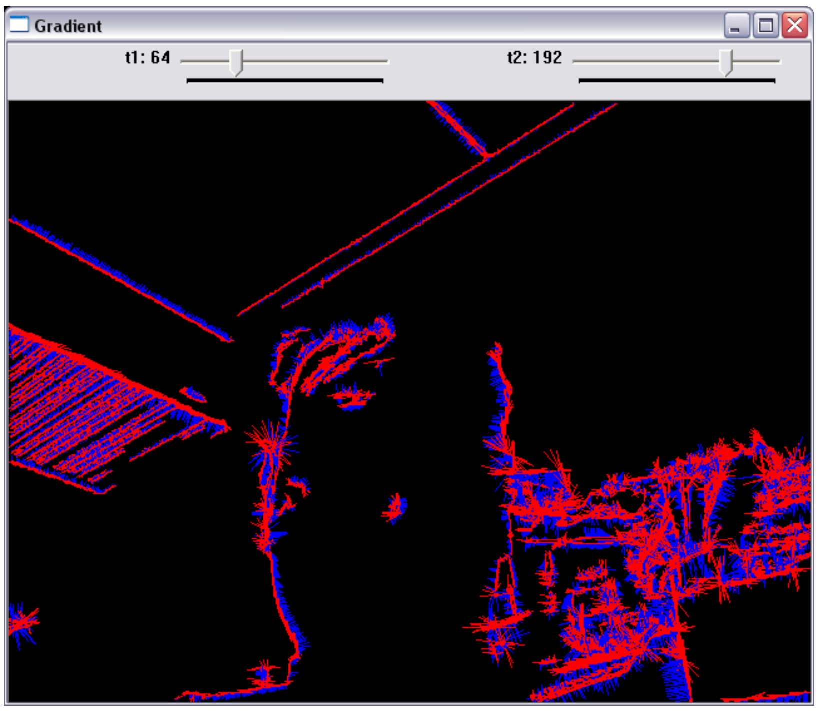
Usage:
  samplefilter.exe

Keyboard Shortcuts:
  any key: cycle through filters
  q: quit

5.4 SampleIntegralImage

This is an example of how to use the *IntegralImage* and *IntegralGradient* classes for image gradient analysis. The vertical (green) and horizontal (red) whole image projections are computed using *IntegralImage::GetSubimage* and shown in the SampleIntegralImage window. The gradients of the image edges are shown in the Gradient window. The edges are detected using the Canny edge detector where t1 and t2 are parameters for the Canny algorithm. The gradients are drawn in red and their local normals are drawn in blue.

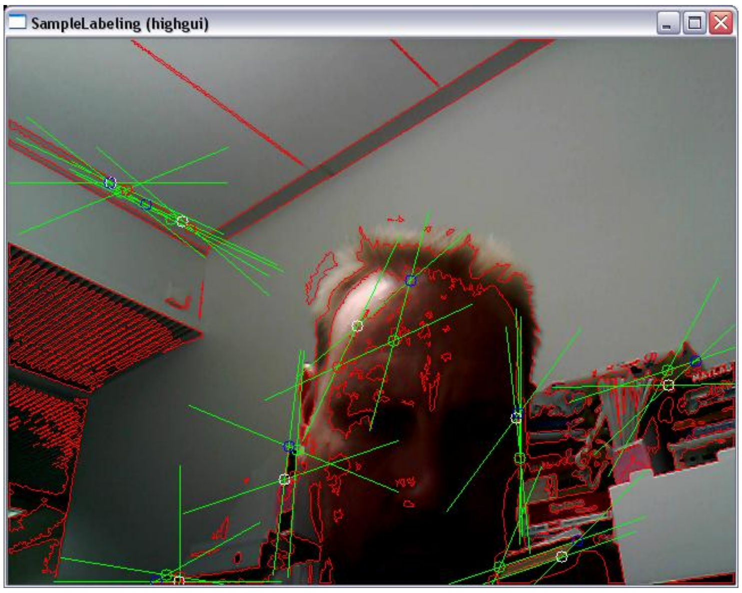
Usage:
  sampleintegralimage.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered



Keyboard Shortcuts:
0: show/hide gradient image
1: show/hide grayscale image
2: show/hide vertical image
3: show/hide horizontal image
4: show/hide canny image
q: quit

5.5 SampleLabeling

This is an example of how to label images using *LabelingCvSeq*. Blobs are detected in the image and if the blobs have four corners, the edges between the corners are visualized.



```
Usage:
  samplelabeling.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture
devices are prefered

Keyboard Shortcuts:
  +: Increase adaptive threshold block size.
  -: Decrease adaptive threshold block size.
  q: quit
```

5.6 SampleMarkerCreator

This is an example that demonstrates the generation of *MarkerData* (or *MarkerArtoolkit*) markers and saving the image using *SaveMarkerImage*. This application can be used to generate markers and multimarker setups that can be used with *SampleMarkerDetector* and *SampleMultiMarker*.

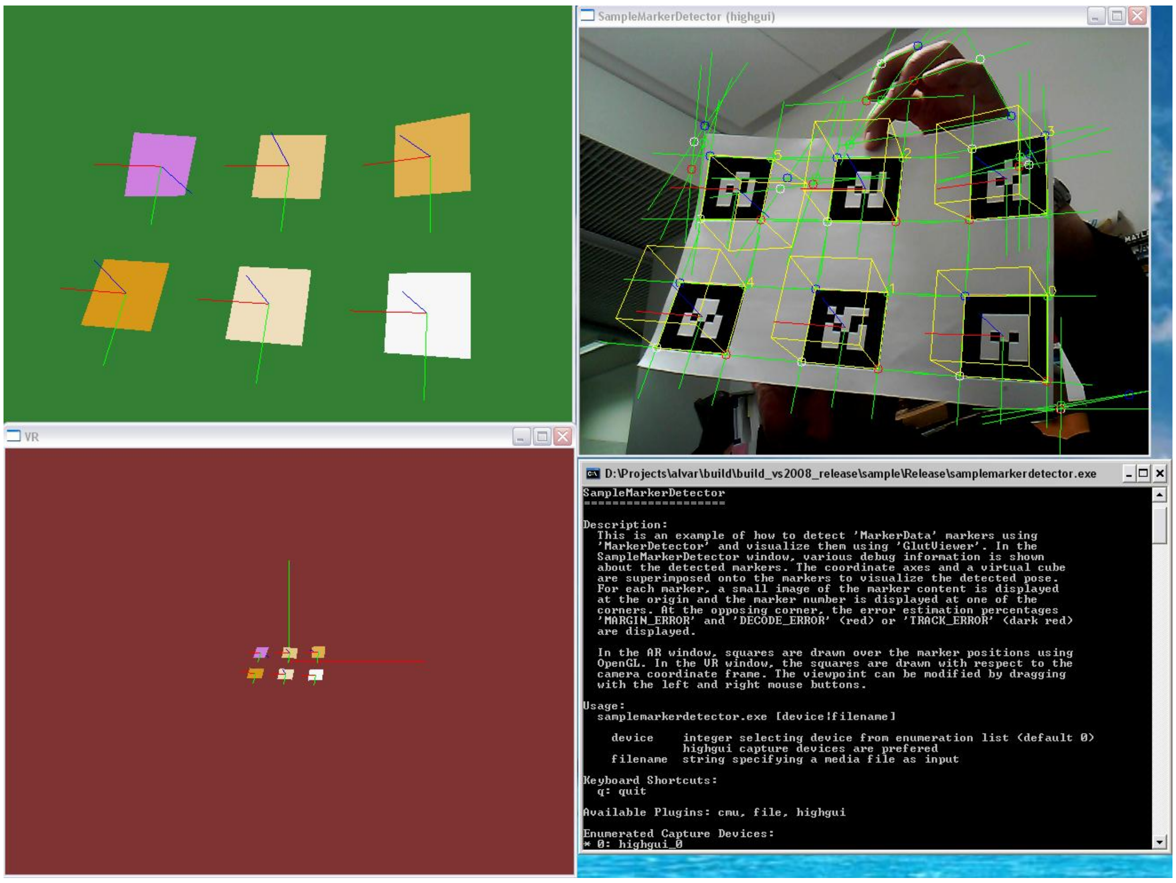
```
Usage:
  samplemarkercreator.exe [options] argument
    65535                  marker with number 65535
    -f 65535               force hamming(8,4) encoding
    -1 "hello world"       marker with string
    -2 catalog.xml         marker with file reference
    -3 www.vtt.fi          marker with URL
    -u 96                  use units corresponding to 1.0 unit per 96 pixels
    -uin                   use inches as units (assuming 96 dpi)
    -ucm                   use cm's as units (assuming 96 dpi) <default>
    -s 5.0                 use marker size 5.0x5.0 units (default 9.0x9.0)
    -r 5                   marker content resolution -- 0 uses default
    -m 2.0                 marker margin resolution -- 0 uses default
    -a                     use ArToolkit style matrix markers
    -p                     prompt marker placements interactively from the user
```

This application is discussed further in section 7.1.

5.7 SampleMarkerDetector

This is an example that shows how to detect *MarkerData* markers and visualize them using *GlutViewer*. In the SampleMarkerDetector window, various debug information is shown about the detected markers. The coordinate axes and a virtual cube are superimposed onto the markers to visualize the detected pose. For each marker, a small image of the marker content is displayed at the origin and the marker number is displayed at one of the corners. At the opposing corner, the error estimation percentages 'MARGIN_ERROR' and 'DECODE_ERROR' (red) or 'TRACK_ERROR' (dark red) are displayed.

In the AR window, squares are drawn over the marker positions using OpenGL. In the VR window, the squares are drawn with respect to the camera coordinate frame. The viewpoint can be modified by dragging with the left and right mouse buttons.


SampleMarkerDetector (highgui)
VR
D:\Projects\alvar\build\build_vs2008_release\sample\Release\samplemarkerdetector.exe

```
SampleMarkerDetector
====================

Description:
  This is an example of how to detect 'MarkerData' markers using
  'MarkerDetector' and visualize them using 'GlutViewer'. In the
  SampleMarkerDetector window, various debug information is shown
  about the detected markers. The coordinate axes and a virtual cube
  are superimposed onto the markers to visualize the detected pose.
  For each marker, a small image of the marker content is displayed
  at the origin and the marker number is displayed at one of the
  corners. At the opposing corner, the error estimation percentages
  'MARGIN_ERROR' and 'DECODE_ERROR' (red) or 'TRACK_ERROR' (dark red)
  are displayed.

  In the AR window, squares are drawn over the marker positions using
  OpenGL. In the VR window, the squares are drawn with respect to the
  camera coordinate frame. The viewpoint can be modified by dragging
  with the left and right mouse buttons.

Usage:
  samplemarkerdetector.exe [device|filename]

    device    integer selecting device from enumeration list (default 0)
              highgui capture devices are prefered
    filename  string specifying a media file as input

Keyboard Shortcuts:
  q: quit

Available Plugins: cmu, file, highgui

Enumerated Capture Devices:
* 0: highgui_0
```

Usage:
  samplemarkerdetector.exe [device|filename]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered
    filename = string specifying a media file as input

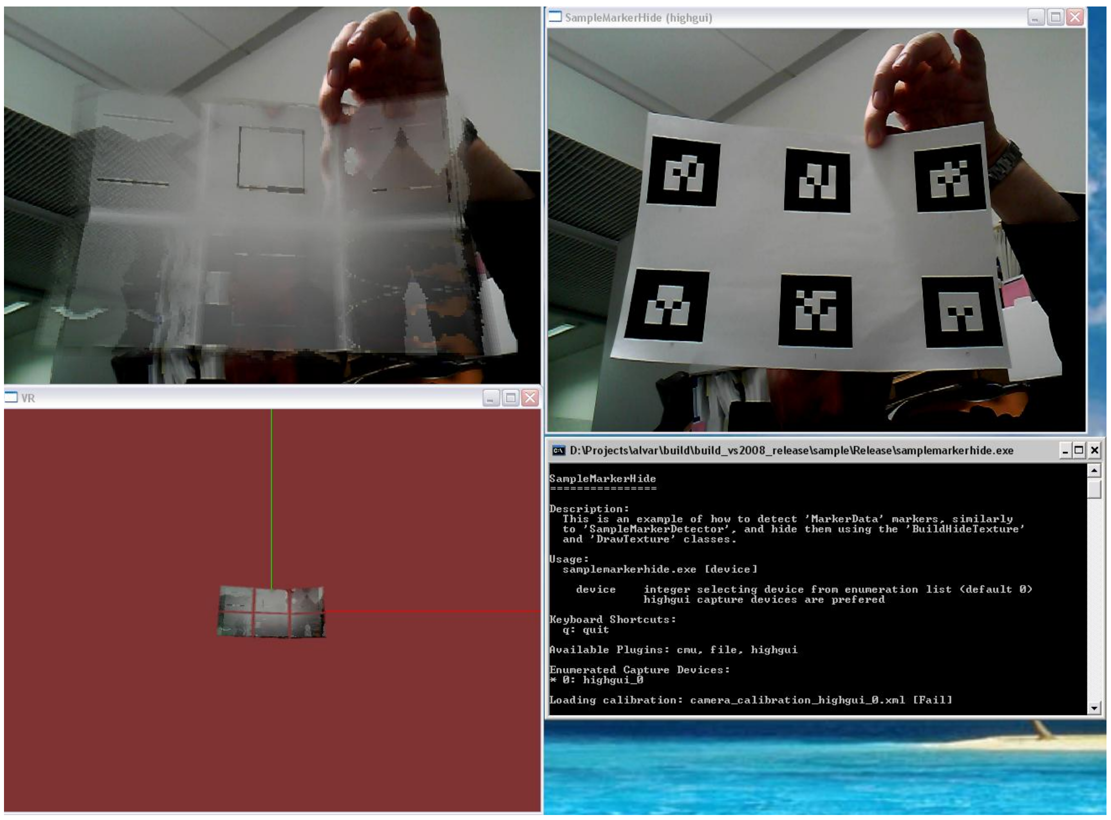
Keyboard Shortcuts:
  q: quit

5.8 SampleMarkerHide

This is an example that shows how to detect *MarkerData* markers, visualize them using *GlutViewer* and hide them with *BuildHideTexture* and *DrawTexture*.

Usage:
  samplemarkerhide.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered

Keyboard Shortcuts:
  q: quit


SampleMarkerHide (highgui)
VR
D:\Projects\alvar\build\build_vs2008_release\sample\Release\samplemarkerhide.exe

```
SampleMarkerHide
================

Description:
  This is an example of how to detect 'MarkerData' markers, similarly
  to 'SampleMarkerDetector', and hide them using the 'BuildHideTexture'
  and 'DrawTexture' classes.

Usage:
  samplemarkerhide.exe [device]

    device    integer selecting device from enumeration list (default 0)
              highgui capture devices are prefered

Keyboard Shortcuts:
  q: quit

Available Plugins: cmu, file, highgui

Enumerated Capture Devices:
* 0: highgui_0

Loading calibration: camera_calibration_highgui_0.xml [Fail]
```

5.9 SampleMarkerlessCreator

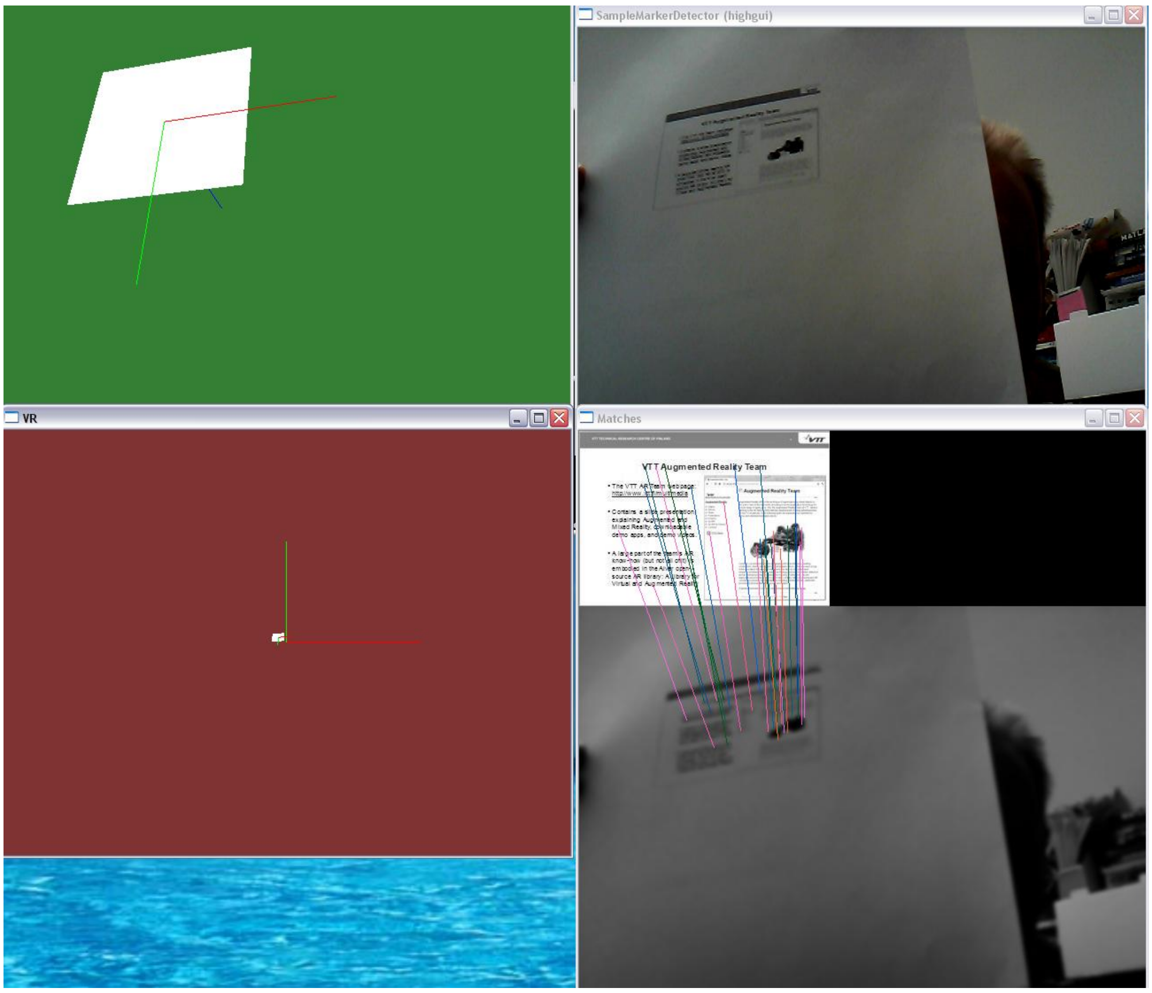
This is an example of how to use the *FernImageDetector* class to train a Fern classifier for markerless image-based tracking. The image should contain many unique features and be in the range of 200x200 to 500x500 pixels. A '**.dat*' file will be saved in the same directory as the image and can be used with the *SampleMarkerlessDetector* sample. Training will take about one minute with a 200x200 sample.

```
Usage:
  samplemarkerlesscreator filename
    filename = filename of image to train
```

5.10 SampleMarkerlessDetector

This is an example of how to use the *FernImageDetector* and *FernPoseEstimator* classes to detect and track an image and visualize it using *GlutViewer*. The classification must first be trained with the *SampleMarkerlessCreator* sample and the resulting file passed as an argument to this sample.

For optimal results, a high quality USB camera or a Firewire camera is necessary. It is also advised to calibrate the camera using the SampleCamCalib sample. It should be noted that the size of the trained image will affect the optimal distance for detection.



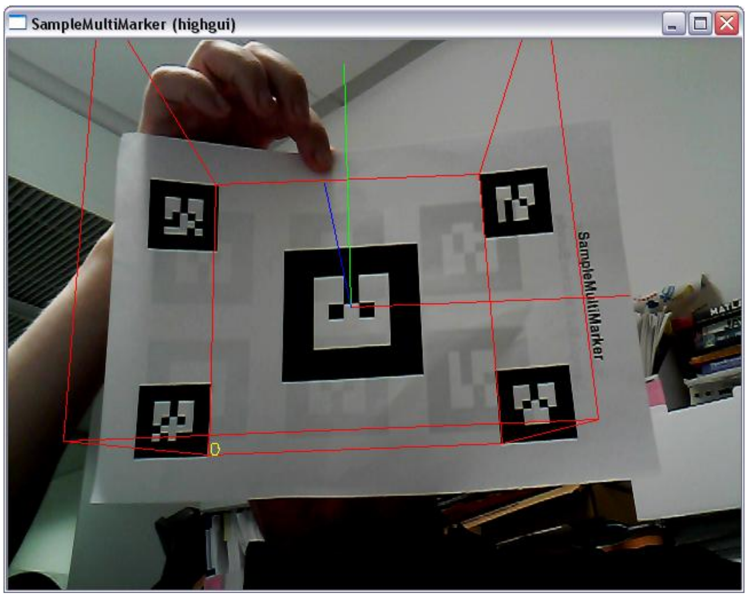
Usage:
  samplemarkerlessdetector filename [device]
    filename = the filename of classifier (.dat)
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered

Keyboard Shortcuts:
  q: quit

5.11 SampleMultiMarker

This is an example of how to use the *MultiMarker* class to detect preconfigured multi-marker setup (see *multimarker.pdf*). A large cube is drawn over the detected multi-marker even when only some of the markers are visible.

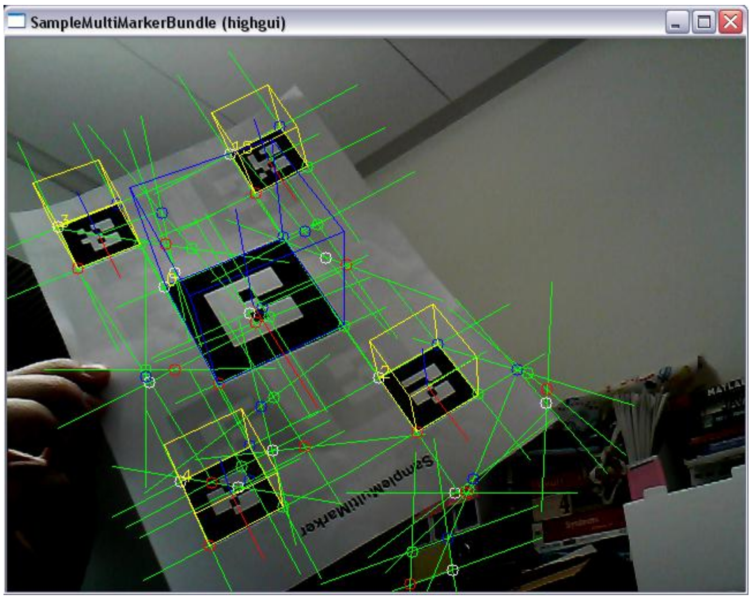
Usage:
  samplemultimarker.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered



Keyboard Shortcuts:
  v: switch between three debug visualizations
  l: load marker configuration from mmarker.txt
  d: toggle the detection of non-readable markers
  q: quit

5.12 SampleMultiMarkerBundle

This is an example that automatically recognises and optimizes *MultiMarker* setups using *MultiMarkerBundle*.

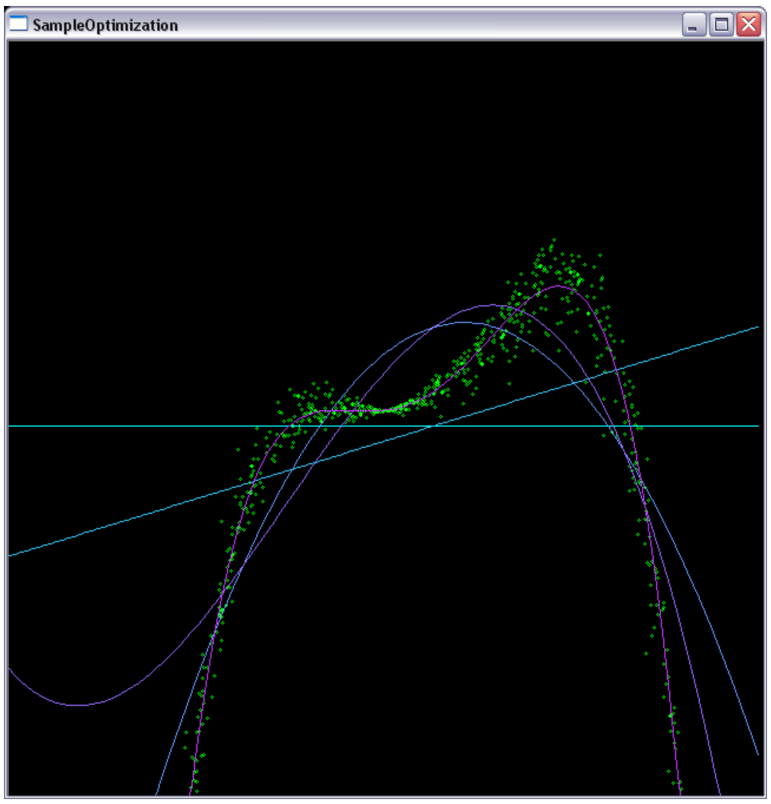


Usage:
  samplemultimarkerbundle.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are preferred

Keyboard Shortcuts:
  v: switch between three debug visualizations
  l: load marker configuration from mmarker.txt
  s: save marker configuration to mmarker.txt
  r: reset marker configuration
  p: add measurement
  o: optimize bundle
  q: quit

5.13 SampleOptimization

This is an example of how to use the *Optimization* class by fitting curves of increasing degree to random data.



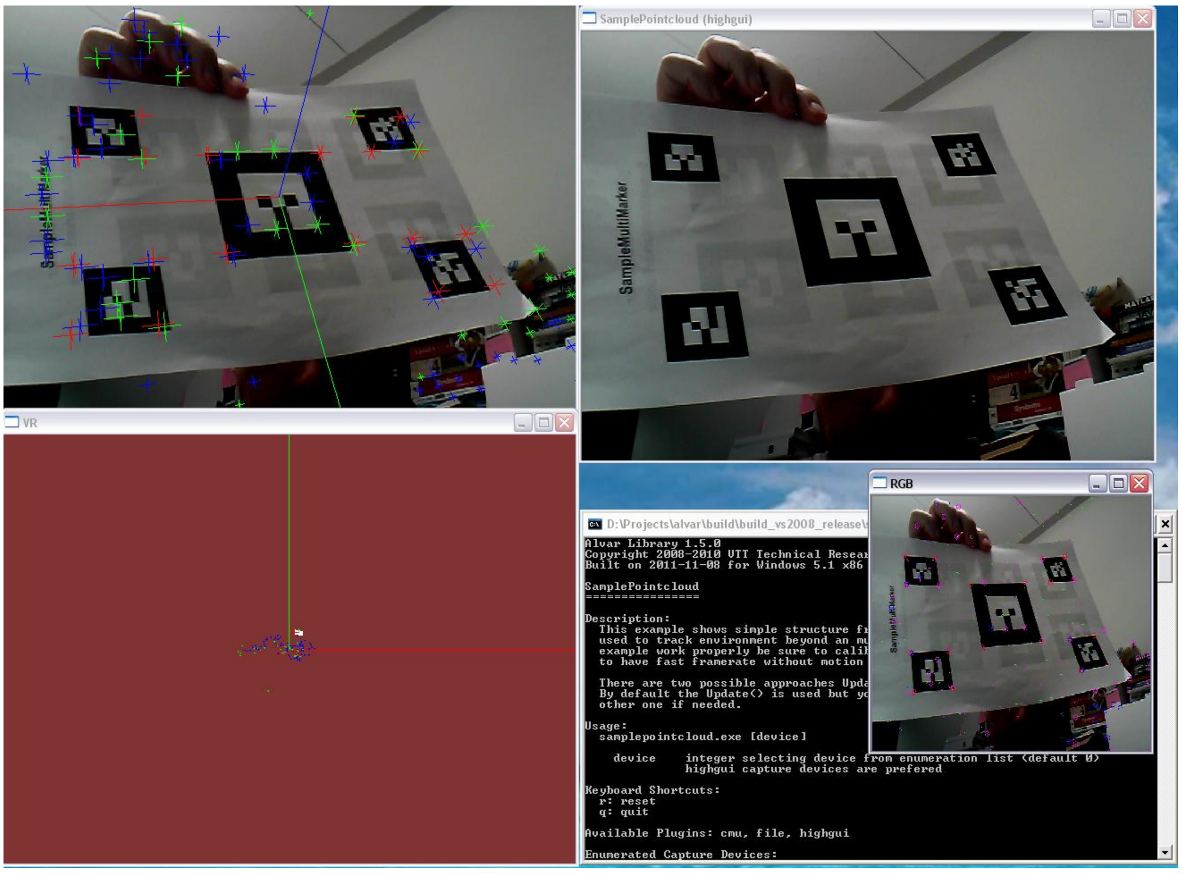
Usage:
  sampleoptimization.exe

Keyboard Shortcuts:
  any key: cycle through datasets
  q: quit

5.14 SamplePointcloud

This is an example showing how to use *SimpleSfM* for tracking the environment using features in addition to *MultiMarker*. To get this example work properly be sure to calibrate your camera and tune it to have fast framerate without motion blur.

There are two possible approaches: *Update()* and *UpdateRotationsOnly()*. By default the *Update()* is used but you can easily uncomment the other one if needed.


SamplePointcloud (highgui)
VR
RGB
SampleMultiMarker
D:\Projects\alvar\build\build_vs2008_release\s

```
Alvar Library 1.5.0
Copyright 2008-2010 VTT Technical Resear
Built on 2011-11-08 for Windows 5.1 x86

SamplePointcloud
================

Description:
  This example shows simple structure fr
  used to track environment beyond an mu
  example work properly be sure to calib
  to have fast framerate without motion

  There are two possible approaches Upda
  By default the Update() is used but yo
  other one if needed.

Usage:
  samplepointcloud.exe [device]

    device    integer selecting device from enumeration list (default 0)
              highgui capture devices are prefered

Keyboard Shortcuts:
  r: reset
  q: quit

Available Plugins: cmu, file, highgui

Enumerated Capture Devices:
```

Usage:
  samplepointcloud.exe [device]
    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered

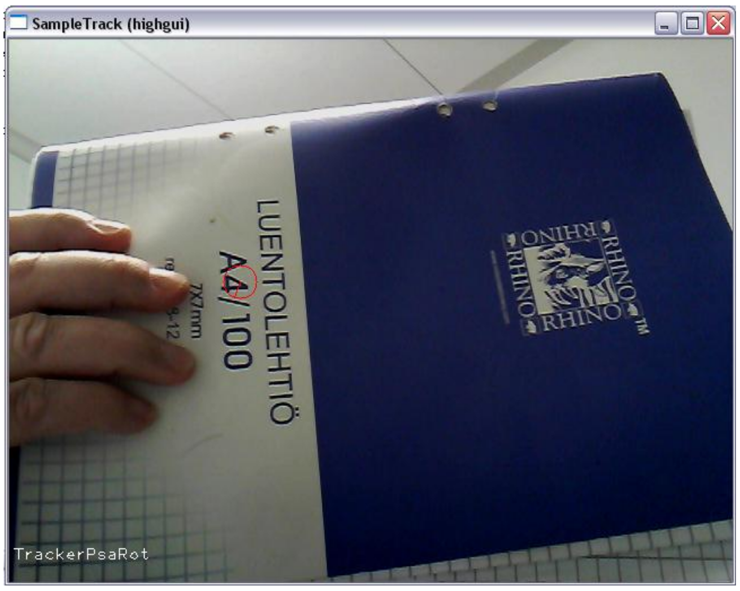
Keyboard Shortcuts:
  r: reset
  q: quit

5.15 SampleTrack

This is an example that shows how to perform tracking of the optical flow using *TrackerPsa*, *TrackerPsaRot*, *TrackerFeatures*, *TrackerStat* or *TrackerStatRot*.

Usage:
  sampletrack.exe [device]

    device = integer selecting device from enumeration list (default 0); highgui capture devices are prefered



Keyboard Shortcuts:

  r,t: reset tracker

  n,space: cycle through tracking algorithms

  q: quit

6 Demo Programs: Using ALVAR with OpenSceneGraph

ALVAR **Sdk** distribution package contains C++ source code only, the users must compile their own programs.

ALVAR itself contains no support for 3D graphics or 3D models.

There are separate demo programs, which use OpenSceneGraph for 3D graphics.

There are three demo programs that demonstrate the core ALVAR features, and three demonstaring the advanced features.

The core demos use only HighGui for acquisition and refer to hard-coded data files (no command-line parameters).

The advanced demos use all installed acquisition plug-ins, and while they also refer to hard-coded data file names, some settings can be overridden with command-line parameters.

The command-line parameters of the advanced demos are: webcam index, camera calibration file, and app-specific option.

6.1 Demo Programs for ALVAR Core Features

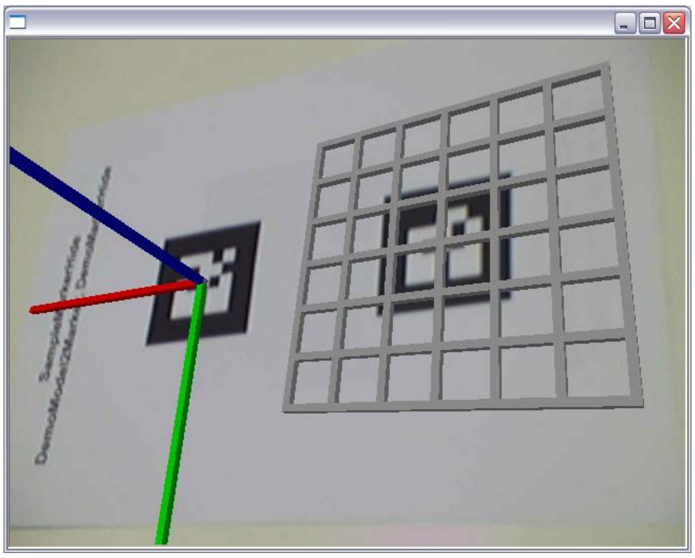
With all three programs, if file ‘*calib.xml*’ is found, it is used as your camera calibration. This file can be created with the utility program ‘*SampleCamCalib.exe*’.

All three core demo programs use the class CVideoBG (defined in header file CommonUtils.h), which draws the current video frame (i.e. the picture from the webcam) as the background, so the desired OSG 3D model(s) can be drawn (i.e. augmented) on top of it.

In the main program of each demo the OSG render loop is entered. And for each frame the program calls the ALVAR image acquisition (to get a new video frame) and checks whether the tracked markers are detected. If they are, the marker pose (i.e. position and orientation) is computed for each marker, and the corresponding OSG model is set to the same pose and drawn; this way the OSG models are augmented to the video frame on top of the corresponding markers. However, if a tracked marker is not detected, the corresponding OSG model is hidden.

6.1.1 Model2Marker

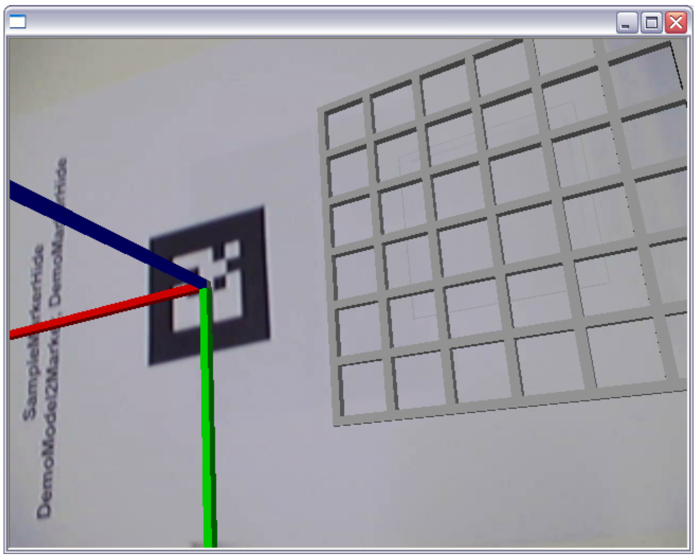
Identifies two independent markers and renders simple OSG model on top of both markers.



Print markers 5 and 10 in the *Alvar.pdf* file in the **doc** directory and show them to your webcam.

6.1.2 MarkerHide

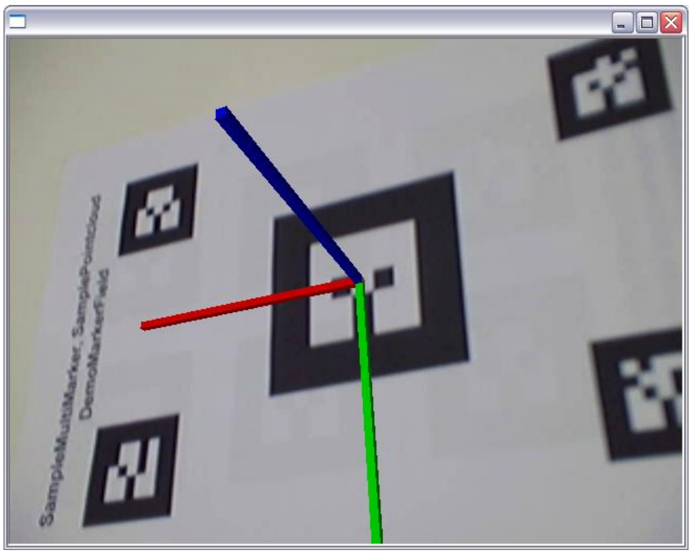
Identifies two independent markers, renders simple OSG model on top of both markers and hides one of the markers using ALVAR hide-texture generation.



Print markers 5 and 10 in the *Alvar.pdf* file in the **doc** directory and show them to your webcam.

6.1.3 MarkerField

Identifies a set of markers in predefined configuration and renders a single OSG model as long as any one of the markers is visible.



Print the multimarker in the *Alvar.pdf* file in the **doc** directory and show it to your webcam.

6.2 Demo Programs for ALVAR Advanced Features

With all three programs, if file '*calib.xml*' is found, it is used as your camera calibration. This file can be created with the utility program '*SampleCamCalib.exe*'.

All three advanced demo programs use the class ViewWithBackGroundImage (defined in header file CommonUtils.h), which draws the current video frame (i.e. the picture from the webcam) as the background, so the desired OSG 3D model(s) can be drawn (i.e. augmented) on top of it.

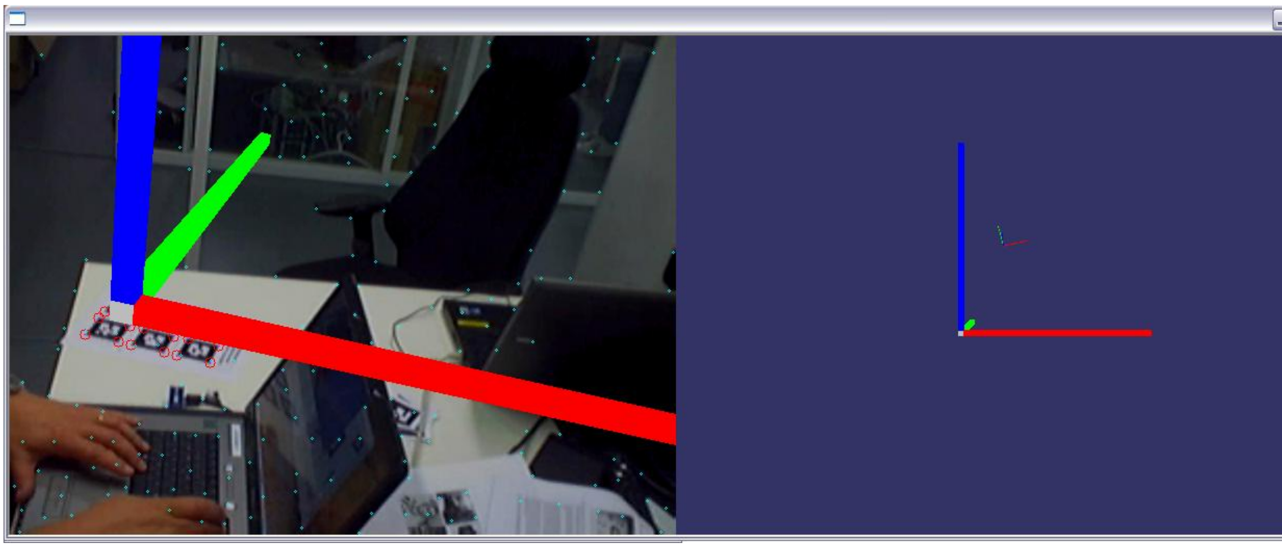
All three advanced demo programs also define functions PickHandler (to handle keyboard commands), InitVideoCapture (to initialize video capture), InitImages (to allocate memory for one RGB image and one grayscale image), InitOSG (to initialize OSG and to create a ViewWithBackGroundImage object), CleanUp (to free memory allocated for the RGB and grayscale images and stop capture at program end), and Process (to process each captured video frame). Besides, each of the three advanced demo programs has a function that is specific to that program: InitALVAR, InitFernClassifier, or GetMultiMarkerPose.

The main program of each demo prints the usage instructions, processes the command-line parameters, and calls the Init* functions. If the initializations succeed, the Process function is called, which captures a video frame, calls the relevant ALVAR functions, and calls the OSG rendering function "viewer->frame()". Before the OSG rendering the return values of the relevant ALVAR functions have determined whether the OSG model should be hidden or shown; and if shown, what is the correct pose of the OSG model so that it is correctly augmented to the video frame.

The markers used by these programs are in the *Alvar.pdf* file in the **doc** subdirectory; other data files that these programs use are in the **data** subdirectory of the ALVAR installation directory.

6.2.1 OsgSfM

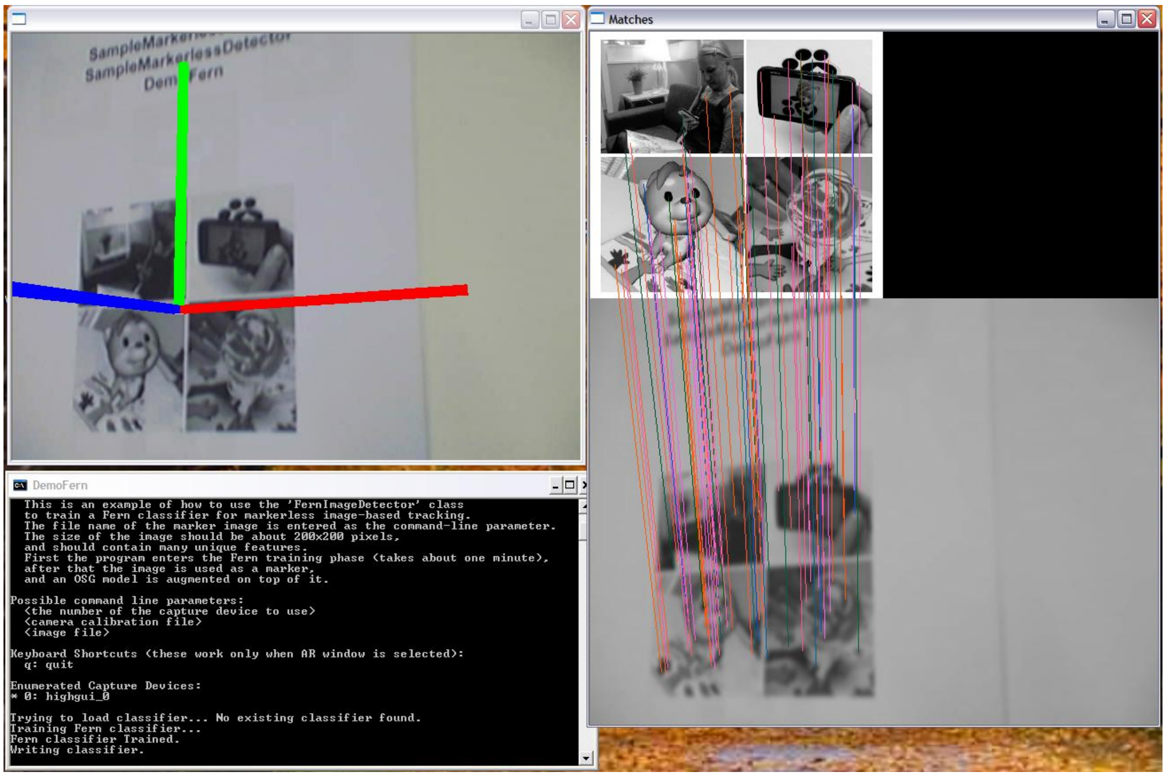
Identifies a predefined marker configuration and renders an OSG model. Reconstructs features with two alternative methods for tracking in the surroundings.



The default multimarker to be tracked is in file '*mmarker.xml*' (3rd cmd-line parameter can be used to override this), which refers to multimarker in the *Alvar.pdf* file. Some keyboard commands are available, see the command window.

6.2.2 OsgFern

Uses an image as a marker and renders an OSG model. Uses the Fern's algorithm.



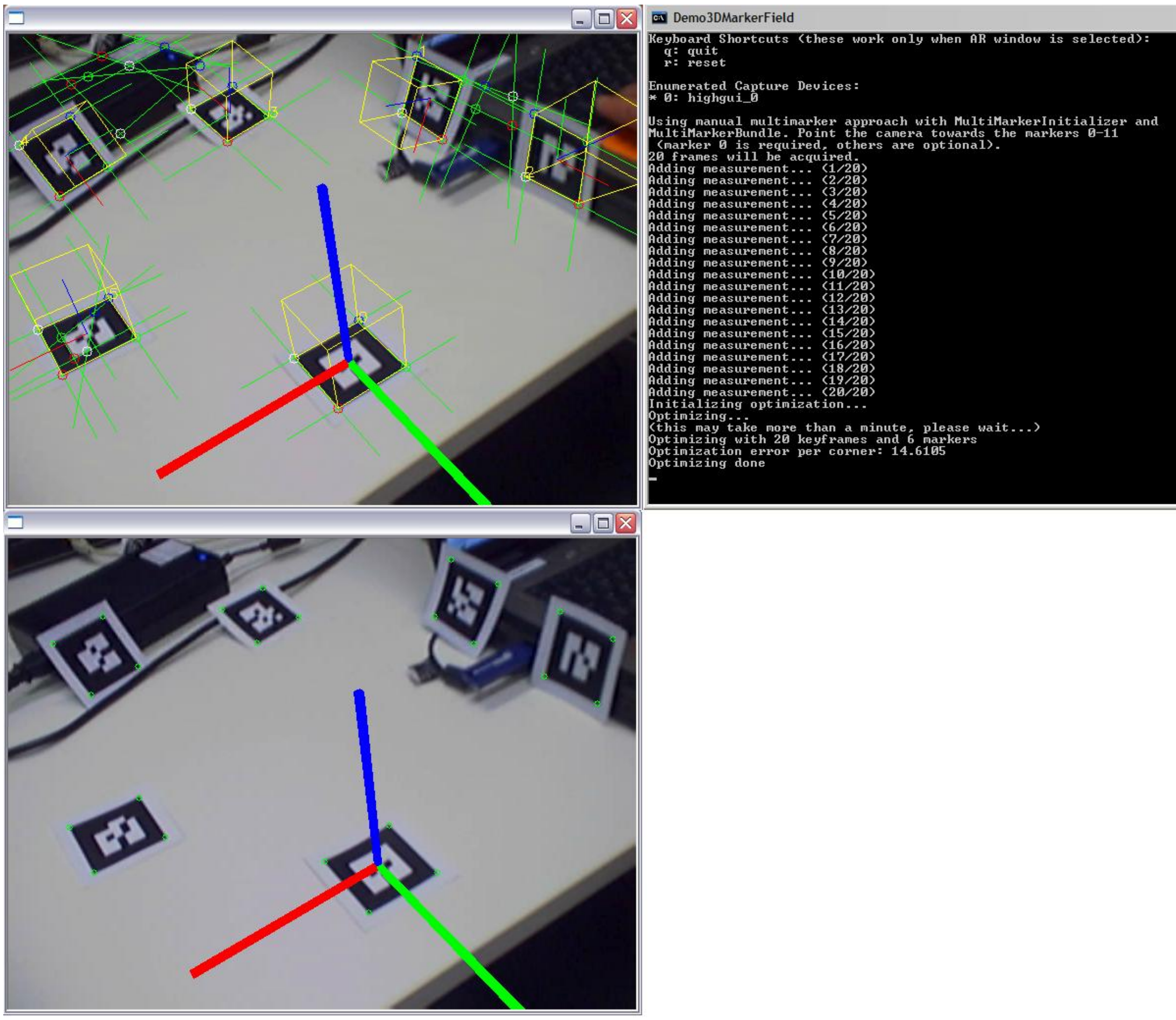
```
  This is an example of how to use the 'FernImageDetector' class
  to train a Fern classifier for markerless image-based tracking.
  The file name of the marker image is entered as the command-line parameter.
  The size of the image should be about 200x200 pixels,
  and should contain many unique features.
  First the program enters the Fern training phase (takes about one minute),
  after that the image is used as a marker,
  and an OSG model is augmented on top of it.

Possible command line parameters:
  <the number of the capture device to use>
  <camera calibration file>
  <image file>

Keyboard Shortcuts (these work only when AR window is selected):
  q: quit

Enumerated Capture Devices:
* 0: highgui_0

Trying to load classifier... No existing classifier found.
Training Fern classifier...
Fern classifier Trained.
Writing classifier.
```

The default image file to be tracked is in '*AlvarSlide.jpg*'. Use the 3rd cmd-line parameter to enter some other image to be used as a marker, its size should be about 200x200 pixels.

First time the program is executed for a new image, it must train the Fern classifier. This takes about one minute, after which the program saves the classifier training results. For subsequent executions the program uses the results data file, so no training period is needed.

6.2.3 Osg3DMarkerField

Deduces the spatial configuration of a 3D marker field, then renders an OSG model on top of the selected marker in this 3D marker field.



```
Keyboard Shortcuts (these work only when AR window is selected):
  q: quit
  r: reset

Enumerated Capture Devices:
* 0: highgui_0

Using manual multimarker approach with MultiMarkerInitializer and
MultiMarkerBundle. Point the camera towards the markers 0-11
 (marker 0 is required, others are optional).
20 frames will be acquired.
Adding measurement... (1/20)
Adding measurement... (2/20)
Adding measurement... (3/20)
Adding measurement... (4/20)
Adding measurement... (5/20)
Adding measurement... (6/20)
Adding measurement... (7/20)
Adding measurement... (8/20)
Adding measurement... (9/20)
Adding measurement... (10/20)
Adding measurement... (11/20)
Adding measurement... (12/20)
Adding measurement... (13/20)
Adding measurement... (14/20)
Adding measurement... (15/20)
Adding measurement... (16/20)
Adding measurement... (17/20)
Adding measurement... (18/20)
Adding measurement... (19/20)
Adding measurement... (20/20)
Initializing optimization...
Optimizing...
(this may take more than a minute, please wait...)
Optimizing with 20 keyframes and 6 markers
Optimization error per corner: 14.6105
Optimizing done
_
```

Create a 3D marker field using Alvar markers 0 - 11. Marker 0 is required (its pose is reported as the marker field pose), others are optional. Enter another marker index (1-11) as the 3rd cmd-line parameter to use another pose marker. The size of all the used markers 0 – 11 should be the same.

7 ALVAR Utility Programs

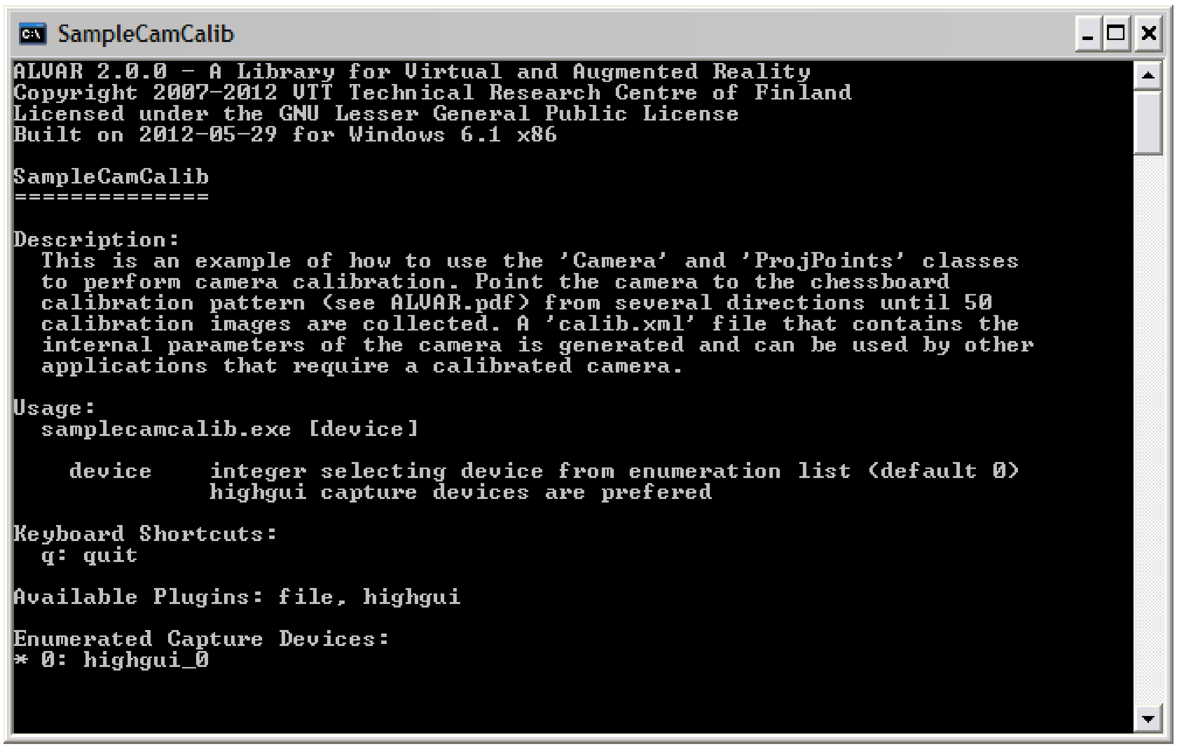
In the **Bin** distribution package of Alvar you'll find two ALVAR utilities: *SampleCamCalib.exe* and *SampleMarkerCreator.exe*.

Actually, *SampleCamCalib.exe* is the SampleCamCalib sample of ALVAR library, while *SampleMarkerCreator.exe* is the SampleMarkerCreator sample of ALVAR library.

SampleCamCalib.exe is used to calibrate your webcam, using the chessboard pattern.

SampleMarkerCreator.exe is used to create marker image files which you can print. It can also create multimarkers, i.e. 2D marker fields.

7.1 SampleCamCalib.exe



```
ALVAR 2.0.0 - A Library for Virtual and Augmented Reality
Copyright 2007-2012 VTT Technical Research Centre of Finland
Licensed under the GNU Lesser General Public License
Built on 2012-05-29 for Windows 6.1 x86

SampleCamCalib
==============

Description:
  This is an example of how to use the 'Camera' and 'ProjPoints' classes
  to perform camera calibration. Point the camera to the chessboard
  calibration pattern (see ALVAR.pdf) from several directions until 50
  calibration images are collected. A 'calib.xml' file that contains the
  internal parameters of the camera is generated and can be used by other
  applications that require a calibrated camera.

Usage:
  samplecamcalib.exe [device]

    device    integer selecting device from enumeration list (default 0)
              highgui capture devices are prefered

Keyboard Shortcuts:
  q: quit

Available Plugins: file, highgui

Enumerated Capture Devices:
* 0: highgui_0
```

SampleCamCalib.exe is used to calibrate your webcam, using the chessboard pattern (in the *Alvar.pdf* file in the **doc** directory).

Print the slide containing the chessboard pattern. Attach the print to a level surface.

Start the program, and show the chessboard pattern to your webcam in different positions, distances, and orientations.

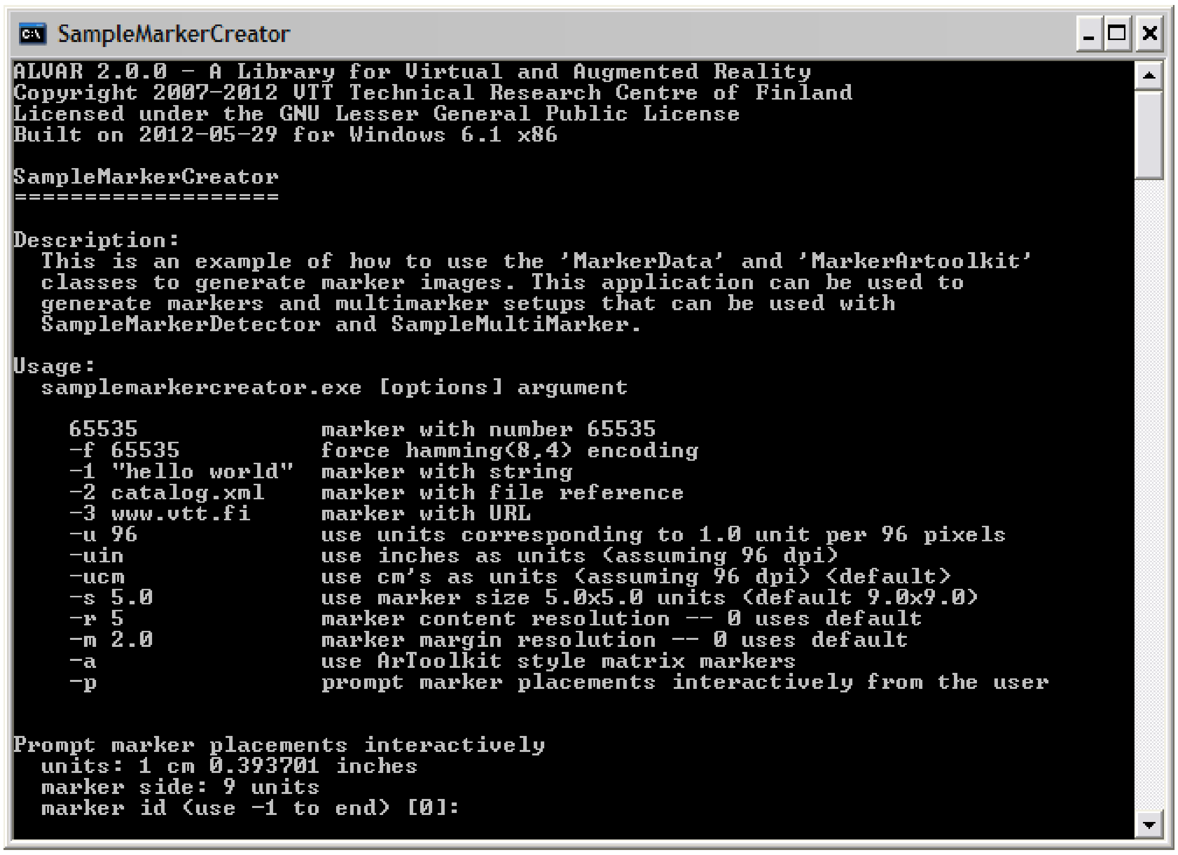
The program acquires 50 samples, about once a second. When a sample is acquired (the chessboard pattern is visible), the found pattern is shortly shown in red.

Calibration result is written to file '*calib.xml*'.

By default, *SampleCamCalib.exe* calibrates the default camera using the default resolution. However, if either of these is not the one you intend to use, enter command-line argument

"1" for your first camera (resolution and other options will be prompted), "2" for your 2nd camera, etc.

7.2 SampleMarkerCreator.exe


SampleMarkerCreator

```
ALVAR 2.0.0 - A Library for Virtual and Augmented Reality
Copyright 2007-2012 VTT Technical Research Centre of Finland
Licensed under the GNU Lesser General Public License
Built on 2012-05-29 for Windows 6.1 x86

SampleMarkerCreator
===================

Description:
  This is an example of how to use the 'MarkerData' and 'MarkerArtoolkit'
  classes to generate marker images. This application can be used to
  generate markers and multimarker setups that can be used with
  SampleMarkerDetector and SampleMultiMarker.

Usage:
  samplemarkercreator.exe [options] argument

    65535              marker with number 65535
    -f 65535           force hamming(8,4) encoding
    -1 "hello world"   marker with string
    -2 catalog.xml     marker with file reference
    -3 www.vtt.fi      marker with URL
    -u 96              use units corresponding to 1.0 unit per 96 pixels
    -uin               use inches as units (assuming 96 dpi)
    -ucm               use cm's as units (assuming 96 dpi) <default>
    -s 5.0             use marker size 5.0x5.0 units (default 9.0x9.0)
    -r 5               marker content resolution -- 0 uses default
    -m 2.0             marker margin resolution -- 0 uses default
    -a                 use ArToolkit style matrix markers
    -p                 prompt marker placements interactively from the user


Prompt marker placements interactively
  units: 1 cm 0.393701 inches
  marker side: 9 units
  marker id (use -1 to end) [0]:
```

SampleMarkerCreator.exe is used to create marker image files which you can print. It can also create multimarkers, i.e. 2D marker fields.

To create an image file containing a single marker, enter the marker number (=ID) as a command-line parameter.

If no command-line parameters are entered, the program goes to the interactive mode; in this mode you can enter the IDs and relative positions of several markers, creating a multimarker. Defaults are sensible, so you can just keep pressing <Enter> until your multimarker is big enough.

The output image of the program is written into file '*markerdata_**0**.png*' (where **0** is replaced with the marker id). Default marker size is 9x9 cm.

The output image of a multimarker employing e.g. markers 0, 1, 2, and 3 is written into file '*markerdata_0_1_2_3.png*', whereas the XML definition of this multimarker is written into '*markerdata_0_1_2_3.xml*' (very long file names are truncated).

SampleMarkerCreator.exe usage options and additional instructions:

- The program prompts for marker id and its position in the used units (3 questions per marker). The default unit is 'cm' and the default marker size is 9x9 cm

- If the user just presses <Enter> to all prompts, the program creates a square marker field which keeps growing (2x2 => … => 3x3 => … => 4x4 etc.)
- When enough markers have been added to the marker field, the user should enter -1 for the ID of the next marker, which ends the prompting
- If the user wishes to use other units or a different marker size, these can be entered as command-line parameters before the interactive mode, e.g.:

```
> samplemarkercreator.exe -uin -s 1.0 –p
```

- If the user wishes to enter a marker field design into a separate text file, it is also possible (contents of ‘test.txt’ is shown to the right):

```
> samplemarkercreator.exe -p < test.txt
```

- If the user wishes to use markers of different sizes, the data for all the markers must be entered as command-line parameters:

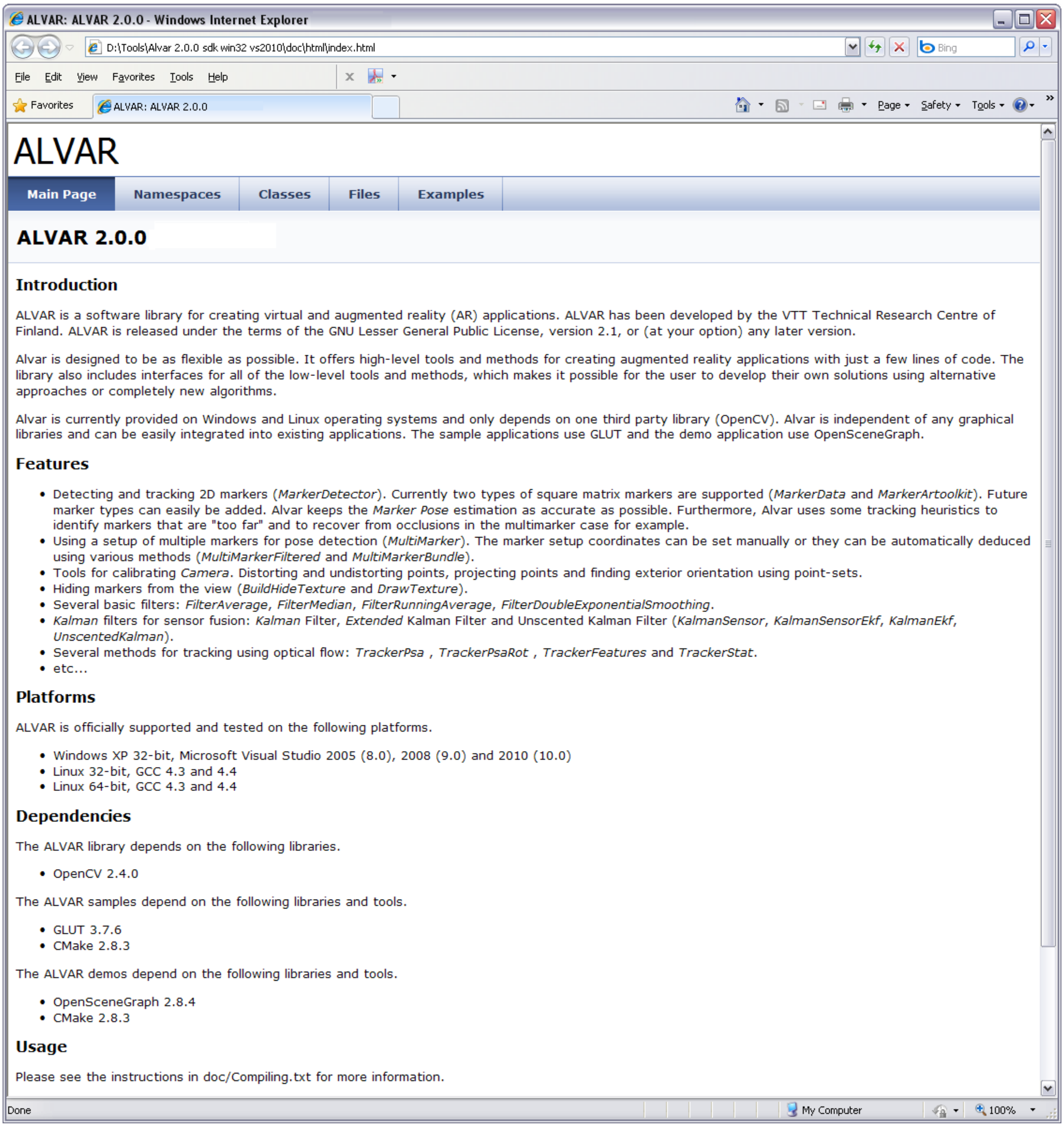
```
> samplemarkercreator.exe -s 18 -xy -36 0 255 -xy -36 -36 254 -xy 0 -36 253 -s 9
–p
```

- This example sets 18x18 cm markers 255, 254 and 253 to the left, up-left, and up from the origin. Then the program enters the interactive mode to prompt for additional 9x9 cm markers.

8 ALVAR HTML Help

This chapter presents a summary of ALVAR HTML Help, which have been generated automatically from the source code comments using Doxygen. The documentation entry is the file \doc\html\index.html, there should be a shortcut for this in the Start menu.

The most important locations in the ALVAR HTML Help are the Classes list (click text 'Classes') and the header Files list (click text 'Files'). In both the Classes list and Files list you can get more detailed information by clicking the class/header file name. Through the detailed information you can navigate from a class name to the corresponding header file and from a header file to the class(es) defined in that file.


ALVAR: ALVAR 2.0.0 - Windows Internet Explorer
D:\Tools\Alvar 2.0.0 sdk win32 vs2010\doc\html\index.html
Bing
File Edit View Favorites Tools Help
Favorites
ALVAR: ALVAR 2.0.0
Page Safety Tools
ALVAR
Main Page Namespaces Classes Files Examples
Done
My Computer
100%

ALVAR 2.0.0

Introduction

ALVAR is a software library for creating virtual and augmented reality (AR) applications. ALVAR has been developed by the VTT Technical Research Centre of Finland. ALVAR is released under the terms of the GNU Lesser General Public License, version 2.1, or (at your option) any later version.

Alvar is designed to be as flexible as possible. It offers high-level tools and methods for creating augmented reality applications with just a few lines of code. The library also includes interfaces for all of the low-level tools and methods, which makes it possible for the user to develop their own solutions using alternative approaches or completely new algorithms.

Alvar is currently provided on Windows and Linux operating systems and only depends on one third party library (OpenCV). Alvar is independent of any graphical libraries and can be easily integrated into existing applications. The sample applications use GLUT and the demo application use OpenSceneGraph.

Features

- Detecting and tracking 2D markers (*MarkerDetector*). Currently two types of square matrix markers are supported (*MarkerData* and *MarkerArtoolkit*). Future marker types can easily be added. Alvar keeps the *Marker Pose* estimation as accurate as possible. Furthermore, Alvar uses some tracking heuristics to identify markers that are "too far" and to recover from occlusions in the multimarker case for example.
- Using a setup of multiple markers for pose detection (*MultiMarker*). The marker setup coordinates can be set manually or they can be automatically deduced using various methods (*MultiMarkerFiltered* and *MultiMarkerBundle*).
- Tools for calibrating *Camera*. Distorting and undistorting points, projecting points and finding exterior orientation using point-sets.
- Hiding markers from the view (*BuildHideTexture* and *DrawTexture*).
- Several basic filters: *FilterAverage*, *FilterMedian*, *FilterRunningAverage*, *FilterDoubleExponentialSmoothing*.
- *Kalman* filters for sensor fusion: *Kalman* Filter, *Extended* Kalman Filter and Unscented Kalman Filter (*KalmanSensor*, *KalmanSensorEkf*, *KalmanEkf*, *UnscentedKalman*).
- Several methods for tracking using optical flow: *TrackerPsa* , *TrackerPsaRot* , *TrackerFeatures* and *TrackerStat*.
- etc...

Platforms

ALVAR is officially supported and tested on the following platforms.

- Windows XP 32-bit, Microsoft Visual Studio 2005 (8.0), 2008 (9.0) and 2010 (10.0)
- Linux 32-bit, GCC 4.3 and 4.4
- Linux 64-bit, GCC 4.3 and 4.4

Dependencies

The ALVAR library depends on the following libraries.

- OpenCV 2.4.0

The ALVAR samples depend on the following libraries and tools.

- GLUT 3.7.6
- CMake 2.8.3

The ALVAR demos depend on the following libraries and tools.

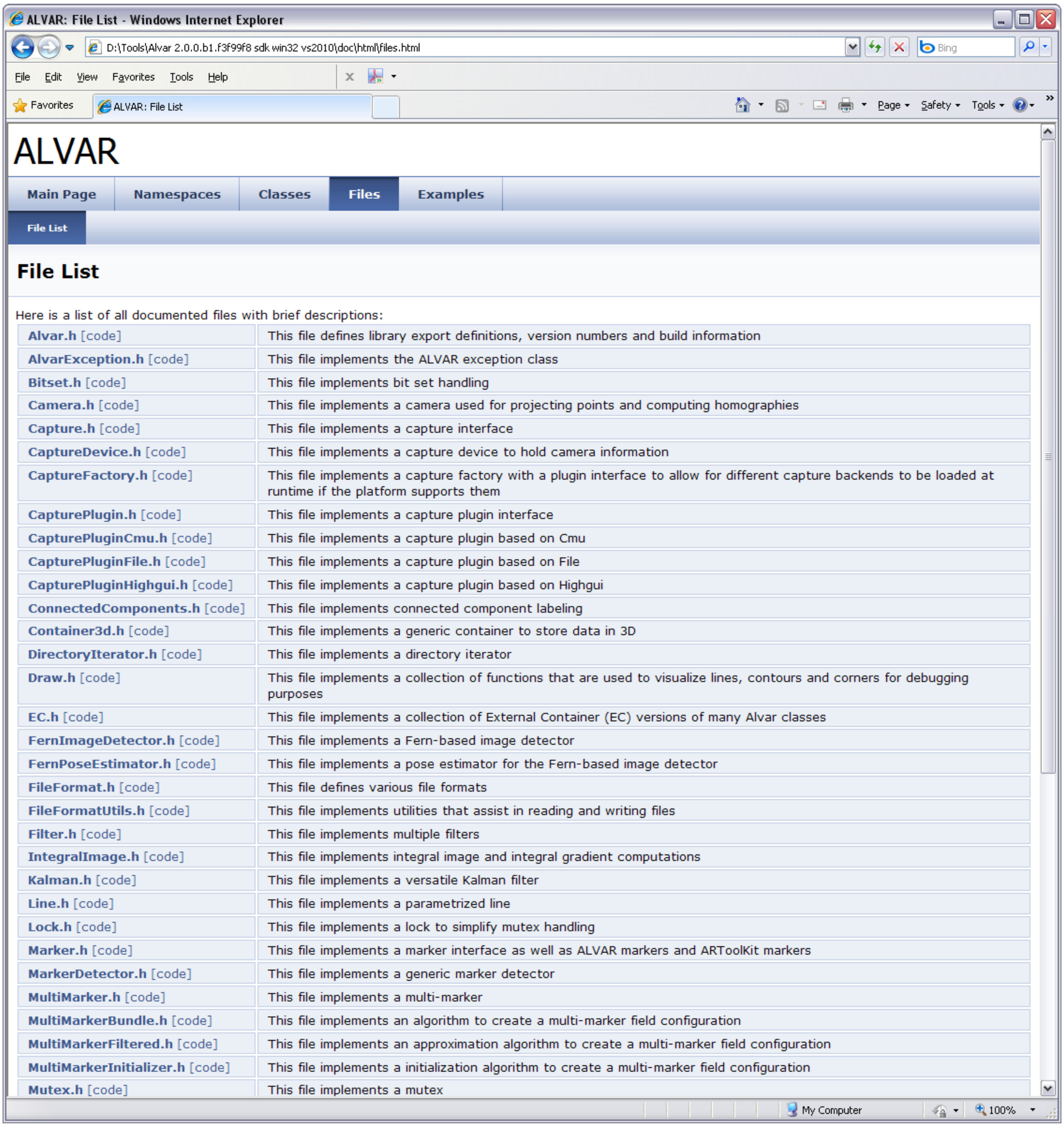
- OpenSceneGraph 2.8.4
- CMake 2.8.3

Usage

Please see the instructions in doc/Compiling.txt for more information.

8.1 File List

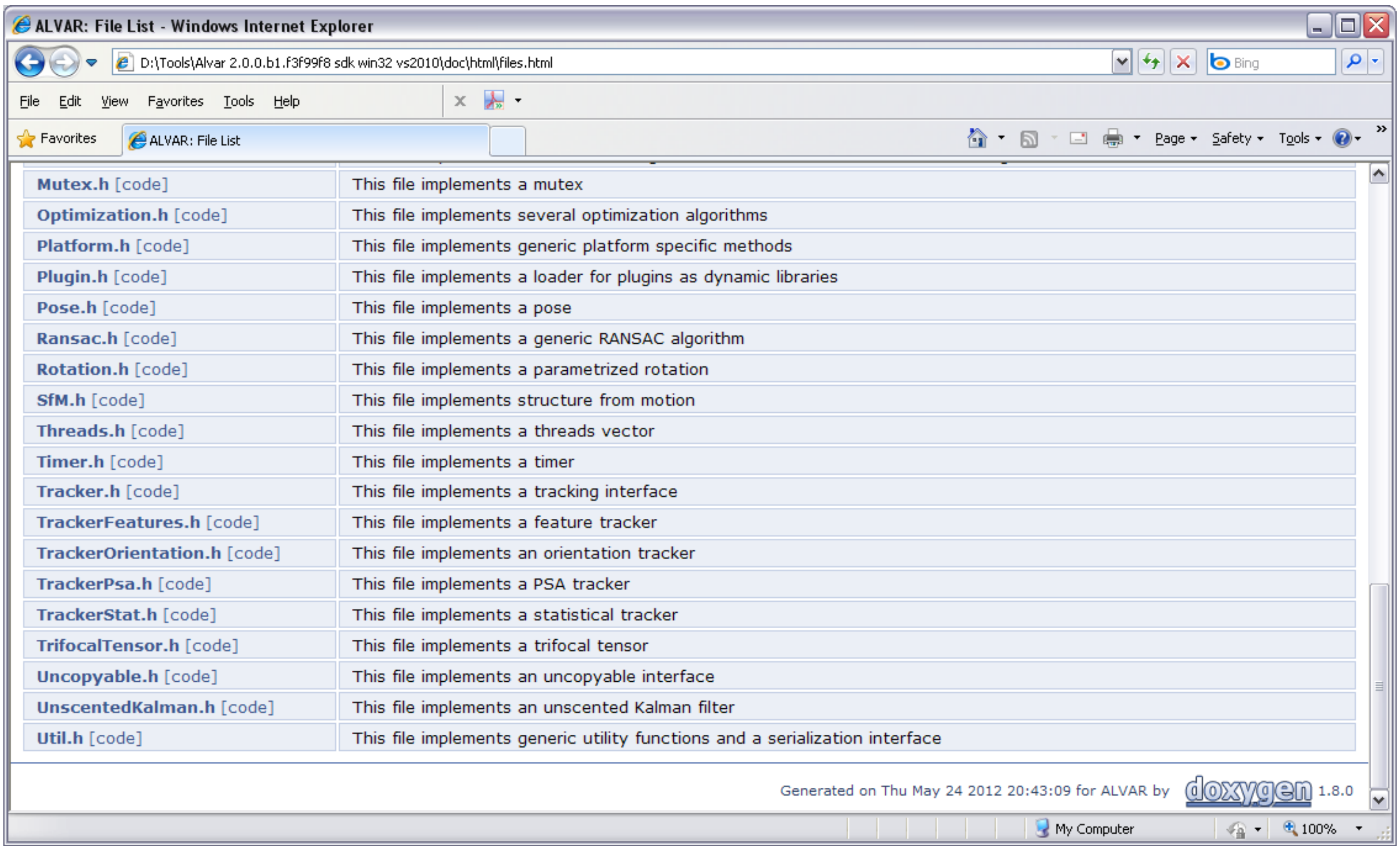
For easy reference, this section presents the header Files list (click text 'Files') of the ALVAR HTML Help, while the next section presents the Classes list (click text 'Classes').


ALVAR: File List - Windows Internet Explorer
D:\Tools\Alvar 2.0.0.b1.f3f99f8 sdk win32 vs2010\doc\html\files.html
Bing
File Edit View Favorites Tools Help
Favorites
ALVAR: File List
Page Safety Tools
ALVAR
Main Page
Namespaces
Classes
Files
Examples
File List
My Computer
100%

File List

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

Alvar.h [code]	This file defines library export definitions, version numbers and build information
AlvarException.h [code]	This file implements the ALVAR exception class
Bitset.h [code]	This file implements bit set handling
Camera.h [code]	This file implements a camera used for projecting points and computing homographies
Capture.h [code]	This file implements a capture interface
CaptureDevice.h [code]	This file implements a capture device to hold camera information
CaptureFactory.h [code]	This file implements a capture factory with a plugin interface to allow for different capture backends to be loaded at runtime if the platform supports them
CapturePlugin.h [code]	This file implements a capture plugin interface
CapturePluginCmu.h [code]	This file implements a capture plugin based on Cmu
CapturePluginFile.h [code]	This file implements a capture plugin based on File
CapturePluginHighgui.h [code]	This file implements a capture plugin based on Highgui
ConnectedComponents.h [code]	This file implements connected component labeling
Container3d.h [code]	This file implements a generic container to store data in 3D
DirectoryIterator.h [code]	This file implements a directory iterator
Draw.h [code]	This file implements a collection of functions that are used to visualize lines, contours and corners for debugging purposes
EC.h [code]	This file implements a collection of External Container (EC) versions of many Alvar classes
FernImageDetector.h [code]	This file implements a Fern-based image detector
FernPoseEstimator.h [code]	This file implements a pose estimator for the Fern-based image detector
FileFormat.h [code]	This file defines various file formats
FileFormatUtils.h [code]	This file implements utilities that assist in reading and writing files
Filter.h [code]	This file implements multiple filters
IntegralImage.h [code]	This file implements integral image and integral gradient computations
Kalman.h [code]	This file implements a versatile Kalman filter
Line.h [code]	This file implements a parametrized line
Lock.h [code]	This file implements a lock to simplify mutex handling
Marker.h [code]	This file implements a marker interface as well as ALVAR markers and ARToolKit markers
MarkerDetector.h [code]	This file implements a generic marker detector
MultiMarker.h [code]	This file implements a multi-marker
MultiMarkerBundle.h [code]	This file implements an algorithm to create a multi-marker field configuration
MultiMarkerFiltered.h [code]	This file implements an approximation algorithm to create a multi-marker field configuration
MultiMarkerInitializer.h [code]	This file implements a initialization algorithm to create a multi-marker field configuration
Mutex.h [code]	This file implements a mutex



ALVAR: File List - Windows Internet Explorer

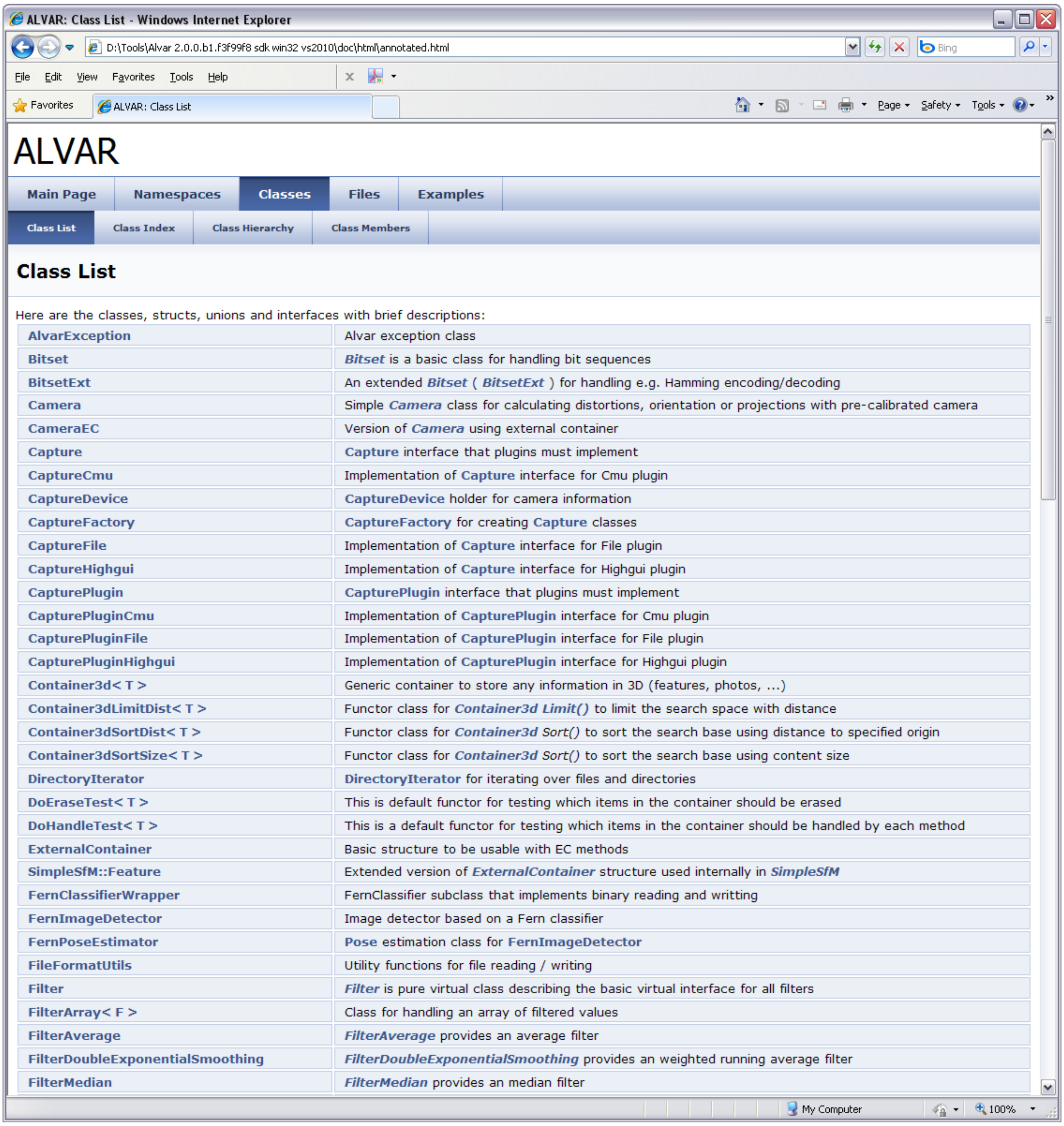
D:\Tools\Alvar 2.0.0.b1.f3f99f8 sdk win32 vs2010\doc\html\files.html

Mutex.h [code]	This file implements a mutex
Optimization.h [code]	This file implements several optimization algorithms
Platform.h [code]	This file implements generic platform specific methods
Plugin.h [code]	This file implements a loader for plugins as dynamic libraries
Pose.h [code]	This file implements a pose
Ransac.h [code]	This file implements a generic RANSAC algorithm
Rotation.h [code]	This file implements a parametrized rotation
SfM.h [code]	This file implements structure from motion
Threads.h [code]	This file implements a threads vector
Timer.h [code]	This file implements a timer
Tracker.h [code]	This file implements a tracking interface
TrackerFeatures.h [code]	This file implements a feature tracker
TrackerOrientation.h [code]	This file implements an orientation tracker
TrackerPsa.h [code]	This file implements a PSA tracker
TrackerStat.h [code]	This file implements a statistical tracker
TrifocalTensor.h [code]	This file implements a trifocal tensor
Uncopyable.h [code]	This file implements an uncopyable interface
UnscentedKalman.h [code]	This file implements an unscented Kalman filter
Util.h [code]	This file implements generic utility functions and a serialization interface

Generated on Thu May 24 2012 20:43:09 for ALVAR by doxygen 1.8.0

8.2 Class List

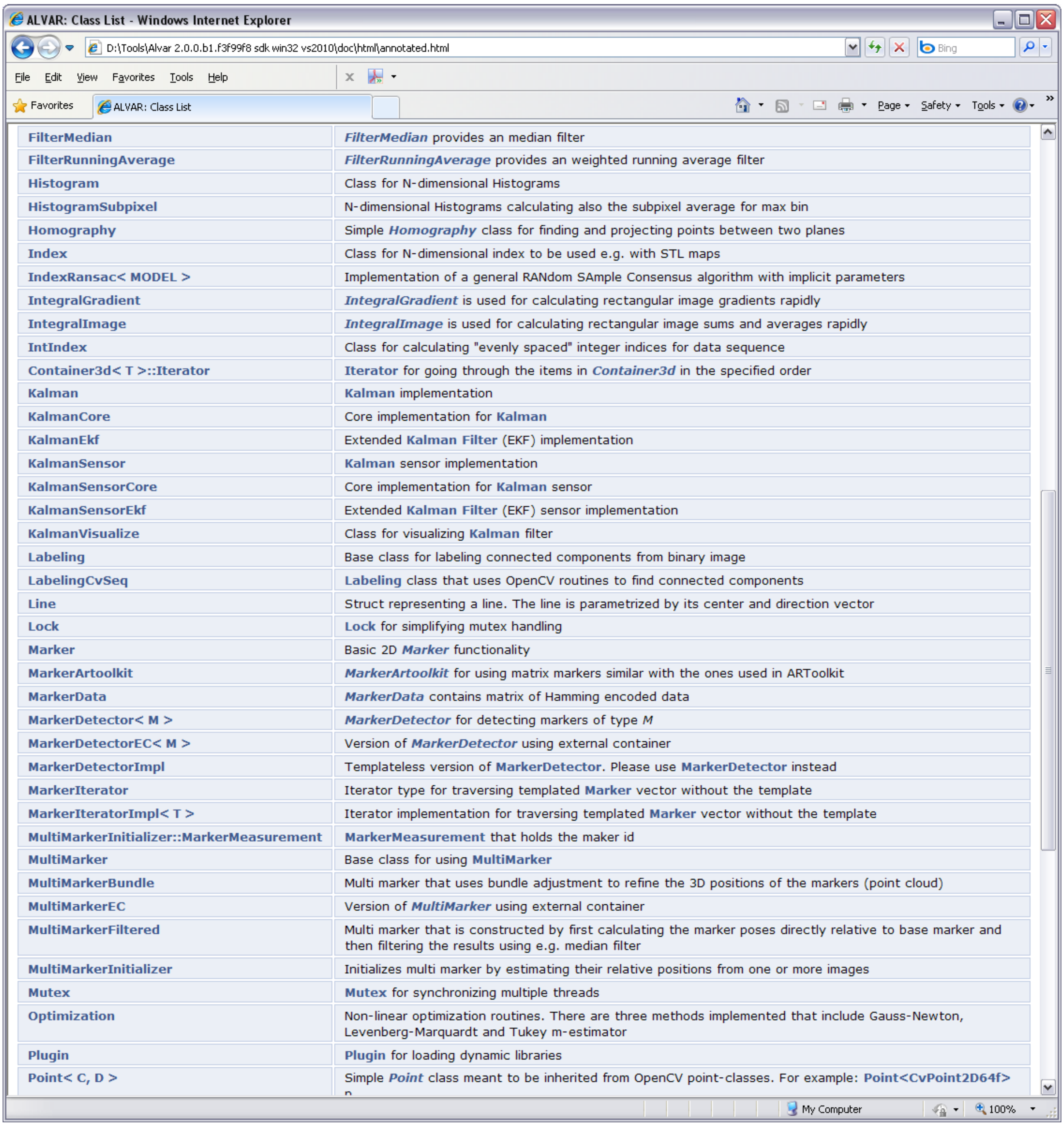
For easy reference, this section presents the Classes list (click text 'Classes') of the ALVAR HTML Help, while the previous section presented the header Files list (click text 'Files').


ALVAR: Class List - Windows Internet Explorer
D:\Tools\Alvar 2.0.0.b1.f3f99f8 sdk win32 vs2010\doc\html\annotated.html
File Edit View Favorites Tools Help
Favorites
ALVAR: Class List
Page Safety Tools
ALVAR
Main Page
Namespaces
Classes
Files
Examples
Class List
Class Index
Class Hierarchy
Class Members
My Computer
100%

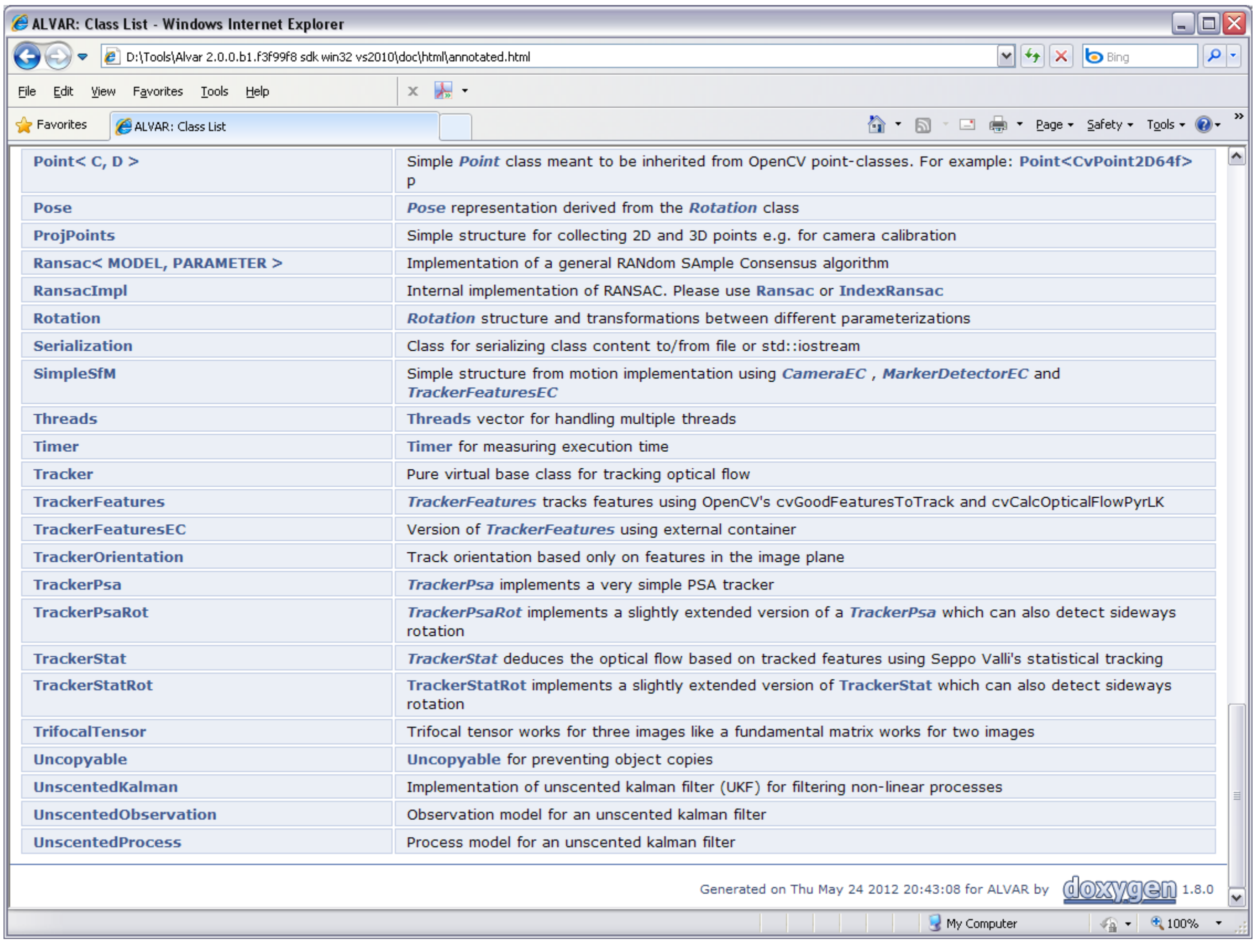
Class List

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

AlvarException	Alvar exception class
Bitset	Bitset is a basic class for handling bit sequences
BitsetExt	An extended Bitset (BitsetExt) for handling e.g. Hamming encoding/decoding
Camera	Simple Camera class for calculating distortions, orientation or projections with pre-calibrated camera
CameraEC	Version of Camera using external container
Capture	Capture interface that plugins must implement
CaptureCmu	Implementation of Capture interface for Cmu plugin
CaptureDevice	CaptureDevice holder for camera information
CaptureFactory	CaptureFactory for creating Capture classes
CaptureFile	Implementation of Capture interface for File plugin
CaptureHighgui	Implementation of Capture interface for Highgui plugin
CapturePlugin	CapturePlugin interface that plugins must implement
CapturePluginCmu	Implementation of CapturePlugin interface for Cmu plugin
CapturePluginFile	Implementation of CapturePlugin interface for File plugin
CapturePluginHighgui	Implementation of CapturePlugin interface for Highgui plugin
Container3d< T >	Generic container to store any information in 3D (features, photos, ...)
Container3dLimitDist< T >	Functor class for Container3d Limit() to limit the search space with distance
Container3dSortDist< T >	Functor class for Container3d Sort() to sort the search base using distance to specified origin
Container3dSortSize< T >	Functor class for Container3d Sort() to sort the search base using content size
DirectoryIterator	DirectoryIterator for iterating over files and directories
DoEraseTest< T >	This is default functor for testing which items in the container should be erased
DoHandleTest< T >	This is a default functor for testing which items in the container should be handled by each method
ExternalContainer	Basic structure to be usable with EC methods
SimpleSfM::Feature	Extended version of ExternalContainer structure used internally in SimpleSfM
FernClassifierWrapper	FernClassifier subclass that implements binary reading and writting
FernImageDetector	Image detector based on a Fern classifier
FernPoseEstimator	Pose estimation class for FernImageDetector
FileFormatUtils	Utility functions for file reading / writing
Filter	Filter is pure virtual class describing the basic virtual interface for all filters
FilterArray< F >	Class for handling an array of filtered values
FilterAverage	FilterAverage provides an average filter
FilterDoubleExponentialSmoothing	FilterDoubleExponentialSmoothing provides an weighted running average filter
FilterMedian	FilterMedian provides an median filter


ALVAR: Class List - Windows Internet Explorer
D:\Tools\Alvar 2.0.0.b1.f3f99f8 sdk win32 vs2010\doc\html\annotated.html
Bing
File Edit View Favorites Tools Help
Favorites
ALVAR: Class List
Page Safety Tools
My Computer
100%

FilterMedian	***FilterMedian*** provides an median filter
FilterRunningAverage	***FilterRunningAverage*** provides an weighted running average filter
Histogram	Class for N-dimensional Histograms
HistogramSubpixel	N-dimensional Histograms calculating also the subpixel average for max bin
Homography	Simple ***Homography*** class for finding and projecting points between two planes
Index	Class for N-dimensional index to be used e.g. with STL maps
IndexRansac< MODEL >	Implementation of a general RANdom SAmple Consensus algorithm with implicit parameters
IntegralGradient	***IntegralGradient*** is used for calculating rectangular image gradients rapidly
IntegralImage	***IntegralImage*** is used for calculating rectangular image sums and averages rapidly
IntIndex	Class for calculating "evenly spaced" integer indices for data sequence
Container3d< T >::Iterator	**Iterator** for going through the items in ***Container3d*** in the specified order
Kalman	**Kalman** implementation
KalmanCore	Core implementation for **Kalman**
KalmanEkf	Extended **Kalman Filter** (EKF) implementation
KalmanSensor	**Kalman** sensor implementation
KalmanSensorCore	Core implementation for **Kalman** sensor
KalmanSensorEkf	Extended **Kalman Filter** (EKF) sensor implementation
KalmanVisualize	Class for visualizing **Kalman** filter
Labeling	Base class for labeling connected components from binary image
LabelingCvSeq	**Labeling** class that uses OpenCV routines to find connected components
Line	Struct representing a line. The line is parametrized by its center and direction vector
Lock	**Lock** for simplifying mutex handling
Marker	Basic 2D ***Marker*** functionality
MarkerArtoolkit	***MarkerArtoolkit*** for using matrix markers similar with the ones used in ARToolkit
MarkerData	***MarkerData*** contains matrix of Hamming encoded data
MarkerDetector< M >	***MarkerDetector*** for detecting markers of type *M*
MarkerDetectorEC< M >	Version of ***MarkerDetector*** using external container
MarkerDetectorImpl	Templateless version of **MarkerDetector**. Please use **MarkerDetector** instead
MarkerIterator	Iterator type for traversing templated **Marker** vector without the template
MarkerIteratorImpl< T >	Iterator implementation for traversing templated **Marker** vector without the template
MultiMarkerInitializer::MarkerMeasurement	**MarkerMeasurement** that holds the maker id
MultiMarker	Base class for using **MultiMarker**
MultiMarkerBundle	Multi marker that uses bundle adjustment to refine the 3D positions of the markers (point cloud)
MultiMarkerEC	Version of ***MultiMarker*** using external container
MultiMarkerFiltered	Multi marker that is constructed by first calculating the marker poses directly relative to base marker and then filtering the results using e.g. median filter
MultiMarkerInitializer	Initializes multi marker by estimating their relative positions from one or more images
Mutex	**Mutex** for synchronizing multiple threads
Optimization	Non-linear optimization routines. There are three methods implemented that include Gauss-Newton, Levenberg-Marquardt and Tukey m-estimator
Plugin	**Plugin** for loading dynamic libraries
Point< C, D >	Simple ***Point*** class meant to be inherited from OpenCV point-classes. For example: **Point<CvPoint2D64f>**


ALVAR: Class List - Windows Internet Explorer
D:\Tools\Alvar 2.0.0.b1.f3f99f8 sdk win32 vs2010\doc\html\annotated.html
File Edit View Favorites Tools Help
Favorites
ALVAR: Class List
Page Safety Tools
My Computer
100%

Point< C, D >	Simple *Point* class meant to be inherited from OpenCV point-classes. For example: Point<CvPoint2D64f> p
Pose	*Pose* representation derived from the *Rotation* class
ProjPoints	Simple structure for collecting 2D and 3D points e.g. for camera calibration
Ransac< MODEL, PARAMETER >	Implementation of a general RANdom SAmple Consensus algorithm
RansacImpl	Internal implementation of RANSAC. Please use Ransac or IndexRansac
Rotation	*Rotation* structure and transformations between different parameterizations
Serialization	Class for serializing class content to/from file or std::iostream
SimpleSfM	Simple structure from motion implementation using *CameraEC* , *MarkerDetectorEC* and *TrackerFeaturesEC*
Threads	Threads vector for handling multiple threads
Timer	Timer for measuring execution time
Tracker	Pure virtual base class for tracking optical flow
TrackerFeatures	*TrackerFeatures* tracks features using OpenCV's cvGoodFeaturesToTrack and cvCalcOpticalFlowPyrLK
TrackerFeaturesEC	Version of *TrackerFeatures* using external container
TrackerOrientation	Track orientation based only on features in the image plane
TrackerPsa	*TrackerPsa* implements a very simple PSA tracker
TrackerPsaRot	*TrackerPsaRot* implements a slightly extended version of a *TrackerPsa* which can also detect sideways rotation
TrackerStat	*TrackerStat* deduces the optical flow based on tracked features using Seppo Valli's statistical tracking
TrackerStatRot	TrackerStatRot implements a slightly extended version of TrackerStat which can also detect sideways rotation
TrifocalTensor	Trifocal tensor works for three images like a fundamental matrix works for two images
Uncopyable	Uncopyable for preventing object copies
UnscentedKalman	Implementation of unscented kalman filter (UKF) for filtering non-linear processes
UnscentedObservation	Observation model for an unscented kalman filter
UnscentedProcess	Process model for an unscented kalman filter

Generated on Thu May 24 2012 20:43:08 for ALVAR by doxygen 1.8.0

9 ALVAR Source Code

This chapter presents a summary of ALVAR source code. The source code files were listed in section 8.1, and the ALVAR classes in section 8.2. In this chapter some of the most important classes and concepts are presented in more detail, as well as some Frequently Asked Questions.

9.1 Code highlights: Most important ALVAR concepts, FAQs

Please note that the best way to learn ALVAR is to examine the ALVAR Sample Code (chapter 5) and the ALVAR Demo Programs (chapter 6). The remaining sections should be read only after those sections.

9.1.1 How the capture system (and plugins) works

ALVAR uses the services of OpenCV libraries for video capture. OpenCV in turn contains three alternative capture libraries: HighGui, CvCam, and CMU. Not all webcams can be accessed with all 3 libraries, and the 3 different libraries have different advantages and disadvantages, so therefore this "embarrassment of riches". HighGui is the easiest to use, so if your webcam can be accessed using HighGui, we recommend using it.

After your webcam has captured a video frame, the captured image data is stored into an image in computer memory. This image is accessed by ALVAR libraries (to determine the poses of the visible markers, for example), and this image is usually used as the background, when the augmented content is drawn.

OpenCV callback functions are often used for video frame acquisition and image processing. The beginning of chapter 5 explains the callback function structure of sample code, whereas sections 6.1 and 6.2 explain the callback functions of the demo programs.

9.1.2 How the marker detection works

Here we assume that a video frame has been captured and stored into a memory image as described in the previous subsection.

First the color image is converted into a grayscale image. Then this image is converted into a bitonal (only two colors: black and white) image using an adaptive threshold. From this image edges are searched, producing a number of lines. Then sets of four intersecting lines (i.e. quadrangles) are searched, and these are potential markers. Then it is verified that the outside of the quadrangle is white, and the inside is black (i.e. we are indeed seeing the border stripe of a marker). And finally the inside of the marker borders can be interpreted as bits, and the bit pattern is a valid marker. From the four corner points of the detected marker the program can compute the pose (i.e. location and orientation) of the marker in camera coordinates.

The best position accuracy is achieved when the marker is close to the camera, because then the relative error caused by one pixel inaccurary in the image is smaller. However, the detection algorithm has some built-in limits, so very large targets (i.e. markers very close to the camera) are eliminated, i.e. not detected at all. And conversely, very small targets (i.e. markers very far from the camera) are also eliminated.

The best orientation accuracy is achieved when the marker plane is in a 45 degree angle with the camera line of sight. When the marker plane normal is parallel with the camera line of sight the orientation accuracy is lower, because small marker angle changes do not cause significant changes in the detected marker image. And when the marker plane normal is almost perpendicular with the camera line of sight the orientation accuracy is lower, because some of the marker corners are seen in almost same position of the image.

For further background information about this subject one can turn to ARToolkit documentation. The detection principles of ARToolkit are similar than that of ALVAR, so many descriptions of ARToolkit principles apply to ALVAR also. A web search of "ARToolkit marker detection" is a good place to start. Here are a few links that this web search produces:

http://www.hitl.washington.edu/artoolkit/Papers/ART02-Tutorial.pdf

http://www.hitl.washington.edu/artoolkit/documentation/

http://www.artoolworks.com/support/library/Category:ARToolKit_Professional

9.1.3 How the margins work

There must be enough white (paper) around the black marker edges, so edges can be detected in the beginning of the algorithm.

9.1.4 How the multimarkers (i.e. marker fields) work

Several markers working together. Pose can be computed even if only one of the markers is visible.

2D multimarkers: a XML file is required, which specifies the relative positions and sizes of the component markers; the component markers can have different sizes, but they must all be in the same plane.

3D multimarkers: relative positions and orientations of the component markers are automatically computed, when the configuration has been seen from various angles. All component markers must have the same size.

The advantage of the 3D multimarker over the 2D is that it is more robust: there is a higher probability that at least one of the component markers is seen in an advantageous angle (i.e. high orientation accuracy, close to 45 degrees with the camera line of sight). The disadvantage is that a separate initialization stage is needed, and the camera must be moved around during this stage.

9.1.5 Encoding data (integer, text, URL) into ALVAR markers

A number of bits are available in the ALVAR markers depending on the size of the marker. The larger the marker size, the more bits it can contain, but the more unreliable the detection (i.e. distinguishing between the different markers) becomes. Some bits must be fixed, so that marker orientation can be deduced, but other bits can be used for any purpose.

Usually the bits are interpreted as numbers, but this is not necessary. E.g. groups of seven bits can be used to encode ASCII characters, and groups of eight bits can be used to encode the extended ASCII character set. If there is no need for punctuation, numbers or uppercase letters, five bits are sufficient for all English letters.

See also subsection 9.3.16.

9.1.6 FAQ: I compiled my ALVAR program with VC++, but I cannot run the program in another computer

All recent versions of VC++ require that the run-time libraries (specifit to that version of VC++) are installed in the system directory. Otherwise the programs compiled with VC++ will not run. Naturally the Visual Studio installation program installs these libraries; for computers without Visual Studio one must run the so-called "Microsoft Visual C++ <version> Redistributable Package (x86)", which installs the required libraries. This is true for ALVAR also.

So if you are using Visual Studio 2008 SP1 for example, do a web search of "visual c++ 2008 sp1 redistributable package", which should take you to a Microsoft web page, where you can download the Redistributable Package. Download the program and execute it once in the computer you intend to use your ALVAR program. Then your program should work.

Also if you use a newer version of Visual Studio than what ALVAR 2.0 supports, you may be able to compile your ALVAR programs with the new version of VS, but you must first install the Redistributable Package of the VS version that was used to compile the ALVAR DLLs.

9.1.7 FAQ: Using PsaTracker, i.e. motion flow tracking

First, we are basically talking about image plane tracking. That is the movement of the image in the horizontal and vertical directions. We can use this image plane tracking to implement some quick hacks that produce a better experience when doing AR.

When is this needed?

The image plane tracking is meant to kick in when the marker is not detected for a few frames or when the marker starts to move out of the field of view of the camera (moves to the edge of the image when panning). Usually, you have two options: leave the model where it is or stop rendering it completely. This leads to flickering and weird behaviour when the marker moves out of the view of the camera. This motion flow tracking solves this.

How is it done?

The hard part is already done. You can find the TrackerPsa class in ALVAR that implements image plane tracking. To see how to use this class, you can look at the SampleTrack sample. Now the tricky part is using the horizontal and vertical motion to update the pose of the marker. I will describe how this is done using pseudo code.

```
Camera camera;
camera.SetCalib(...);

MarkerDetector markerDetector;
// markerDetector initialization

TrackerPsa trackerPsa;
// no initialization required

double fovX = camera.GetFovX();
double fovY = camera.GetFovY();

...

IplImage *frame;
// get a new frame from the camera

Pose pose;
// this pose must persist across frames, make it a class member

markerDetector.Detect(frame, camera, ...);
// iterate over markers, when id matches update the pose

trackerPsa.Track(frame);

// only update the pose with the image plane movement when no marker is
found
if (numberOfMarkersDetected == 0) {
  double x = -trackerPsa.xd * fovX / frame->width;
  double y = -trackerPsa.yd * fovY / frame->height;

  CvMat *p;
  CvMat *r;
  // create p and r, 3x3

  pose.GetMatrix(p);

  // initialize r as a rotation matrix with x rotation around x-axis and y
rotation around y-axis
  // p = r * p

  pose.SetMatrix(p);

  // release p and r
}

...

// render model using pose as usual
```

9.1.8 FAQ: Contents of the ALVAR Sdk distribution package

The ALVAR **Sdk** library is distributed in binary form only. This means that the source files of the library (files in src directory) are not distributed. Only the binary libraries (DLLs in bin directory) are distributed, along with the headers (files in include directory).

In order to compile the sample applications, you must follow each of the steps in doc/Compiling.txt.

ALVAR **Sdk** installation is organized in the following way:

* bin – The compiled binaries appear in a subdirectory matching the selected build environment (e.g. /bin/msvc80).

* build – The build environment tools, also in a matching subdirectory.

* data – Data files used by some of the Sample and Demo applications.

* demo – Demo applications that demonstrate how to use the library with OSG.

* doc – Documentation as well as API Reference, sample markers in *Alvar.pdf*.

* include – Headers for the Alvar, AlvarPlatform and AlvarPro libraries.

* sample – Sample applications that demonstrate how to use the library.

* src – Sources for the Alvar core library (This directory is not present in binary distributions).

* src/platform – Sources for the AlvarPlatform library (This directory is not present in binary distributions).

* src/pro – Sources for the AlvarPro library (This directory is not present in binary distributions).

9.1.9 FAQ: Cannot complete the first steps of doc/compiling.txt

Question:

The generate.bat file in the msvc90 folder doesn't seem to do anything special for me. I have installed opencv, glut, cmake and alvar. Also, cmake is located in the system's path environment variable.

Answer:

Check the generate.log file that is generated when you run the generate.bat script. It should be located in the same directory. (C:\program files\alvar 2.0.0\build\msvc90\generate.log).

Also verify the paths in Cmakegui. They should be set correctly by default and not be changed.

```
Where is the source code:
C:/Program Files/Alvar 2.0.0
Where to build the binaries:
C:/Program Files/Alvar 2.0.0/build/msvc90/build
```

9.1.10 FAQ: Cannot follow the last steps of doc/compiling.txt

Question:

I completed the 6 steps of compiling.text file but I cannot understand the following step.

```
7. Open ./build/[target]/build/Alvar.sln.bat and build the solution.

   The batch file will ensure that paths to DLLs are properly configured.
```

If it shows the the ALVAR path then there is no such folders and solution file. My path is C:\Program Files\ Alvar 2.0.0\build and it has no Alvar.sln.bat file.

Answer:

You need to go down into a few more directories. For example, if you used ./build/msvc90/generate.bat to generate the project files, then the solution batch file will be in ./build/msvc90/build/Alvar.sln.bat.

You need to replace [target] (in the instructions) by the platform that you have selected. You can select either msvc, msvc80 or msvc90 depending on the compiler you are using.

To recap, if you used:

```
[1] C:\Program Files\Alvar 2.0.0\build\msvc90\generate.bat
```

to generate the project, then the solution batch file will be in:

```
[2] C:\Program Files \Alvar 2.0.0\build\msvc90\build\Alvar.sln.bat
```

You can then double click on [2] and then build the solution in debug or release mode.

Step 8 requires step 7 do be done. When you have the visual studio solution opened, you can right click on the INSTALL project and select 'Build'. This will install all of the samples and dependencies in:

```
[3] C:\Program Files\Alvar 2.0.0\bin\msvc90\
```

Then you can run the samples by double-clicking on them in the Explorer.

9.1.11 FAQ: ALVAR sample reports "Could not find any capture plugins"

This error message means that ALVAR can't find the capture plugins. These are the DLLs found at the following location (assuming default installation folder):

C:\Program Files\Alvar 2.0.0\bin\alvarplugins

If you are running the sample from inside Visual Studio (via F5), you need to make sure that you have launched VS with Alvar.sln.bat (notice the 'bat' extension). This sets up the paths properly so that VS can find the plugins.

If you are running the sample from the command line, make sure that alvarplugins is in the same directory as the executable.

You can also force the search path for the plugins by setting the following environment variable:

```
set ALVAR_PLUGIN_PATH=C:\Program Files\Alvar 2.0.0\bin\alvarplugins
```

9.1.12 FAQ: SampleCamCalib reports "Could not initialize the selected capture backend"

That error message can mean a few things.

1. No compatible camera is attached to the computer.

2. The camera capture system failed to initialize a connection to the camera.

3. The camera does not support accessing image data but provides a video stream.

9.1.13 FAQ: SampleCamCalib reports "Could not find any capture devices"

Check the following things.

1. Do the other samples that require a camera work? For example, does SampleCvTestbed work for you?

2. Does your camera provide access to the raw pixel data? Some cameras only provide an encoded video stream that Alvar can not process.

3. Make sure that the alvarplugins directory is located in the directory that contains the main ALVAR dynamic libraries (DLL). This should be the case if running the samples from within Visual Studio.

9.1.14 FAQ: ALVAR on Windows Vista/7

Although ALVAR is not officially supported on Windows Vista/7, it should work on these platforms.

If there are any problems, they are most likely file access permission problems. The easy solution is to install ALVAR in your user directory. The ALVAR team has tried that on a Windows 7 machine and everything worked as expected, after following the instructions in doc/Compiling.txt

9.1.15 FAQ: Using ALVAR with DirectX

Unfortunately, ALVAR does not support DirectX and nobody on ALVAR team is familiar with the particularities of the DirectX platform. We can only offer some insight as follows.

Source: http://www.toymaker.info/Games/html/matrices.html

There are three matrix used by Direct3D to transform your 3D models into the final 2D image you see on the screen. They are the World Matrix, the View Matrix and the Projection Matrix. ***Note:*** If you are coming from an OpenGL background OpenGL combines the first two together.

Also, the coordinate systems are different.

- **OpenGL**: X to the right, Y to the top and Z to the back.

- **DirectX**: X to the right, Y to the bottom and Z to the front.

The projection matrix comes from `camera.GetOpenglProjectionMatrix(...)` and needs to be converted to the DirectX coordinate system.

The pose matrix comes from `marker.pose.GetMatrixGL(...)`. It needs to be converted to the DirectX coordinate system and then decomposed into view and world matrices. How to do this, we are not sure at the moment.

We can also direct you to the ALVAR OSG demo programs which show how to use OpenSceneGraph to display actual models on top of markers.

9.1.16 FAQ: How to get the pose of a detected marker in ALVAR

Question:

I would like to extract the position of the marker in several values. The values are X, Y, Z, A, B, C, where X = Translation in X direction, Y = Translation in Y direction, Z = Translation in Z direction; and A = Rotation around Z axis, B = Rotation around Y axis, C = Rotation around X axis. Is it possible ?

Answer:

The information you are looking for is available in the `alvar::Pose` class. In the marker detector sample, `alvar::MarkerDetector` is used to find an `alvar::Marker` which contains an `alvar::Pose`.

```
Pose p = (*(marker_detector.markers))[i].pose;
```

The `alvar::Pose` class contains a translation vector and a quaternion. The translation vector will give you X, Y and Z. The rotation will give you the angles. However, there are many conventions for rotation angles, for example Tait-Bryan vs Euler angles. You will need to interpret the quaternion depending on your conventions.

http://en.wikipedia.org/wiki/Euler_angles#Conventions

The `alvar::Pose` class can also give you a 4x4 homogenous transformation matrix using the `GetMatrix()` method.

http://en.wikipedia.org/wiki/Transformation_matrix#Uses

9.1.17 FAQ: How to use text string (ascii) markers

Question:

I would like to know which function extract the const char used to create the marker in order to modify it before to display it. I used Ascii to generate the marker and I want it back in hexadecimal.

Answer:

In order to detect markers created using strings (ascii), you must specify the resolution of the marker since the default resolution in the detector is set to 5. You can do this using `Marker::SetMarkerSize()`. The size auto-detection works fine here.

```
// detect markers of resolution 11
marker_detector.SetMarkerSize(marker_size, 11);
// auto-detect resolution
marker_detector.SetMarkerSize(marker_size, 0);
```

To extract the string, simply use the data member.

```
std::cout << (*(marker_detector.markers))[i].data.str << std::endl;
```

9.1.18 FAQ: Problems with CvMat

Question:

Code like this causes an access violation:

```
CvMat rotation;
p.GetEuler(&rotation);
```

Answer:

When using `CvMat`, you must initialize the matrix before using it. Since you have not initialized it, you are getting invalid pointer errors.

```
  CvMat *euler = cvCreateMat(3, 1, CV_64FC1);
  p.GetEuler(euler);
  cvReleaseMat(&euler);

  CvMat *quaternion = cvCreateMat(4, 1, CV_64FC1);
  p.GetQuaternion(quaternion);
  cvReleaseMat(&quaternion);

  CvMat *rodriques = cvCreateMat(3, 1, CV_64FC1);
  p.GetRodriques(rodriques);
  cvReleaseMat(&rodriques);
```

9.1.19 FAQ: ALVAR seems to produce inaccurate results with a widescreen camera

First, make sure you calibrate your camera. This means that you should run the SampleCamCalib application and use the calibration when you are detecting markers. This will ensure that you have a correct model-view matrix.

Second, you need to make sure that you are properly setting up your projection matrix. This should take into account the width and height of the camera as well as the camera calibration from the previous step.

9.1.20 FAQ: Optimal set of ALVAR markers (minimize chance of improper detection)

ALVAR markers have built-in error correction and in most cases, there should be no problems with mismatches. However, if you are paranoid about robustness and reliability, please read the following explanation.

This question seems to be asked many times. Is there a set of ALVAR markers that minimize the chance of improper detection? This is usually asked when we are choosing markers for a particular large scale application (ex: not a demo). In these cases, we also want "pretty" markers. The following explains how to select markers that meet these requirements.

When talking about markers, beauty is often equated with symmetry. The following is a list of all symmetric 5x5 ALVAR markers.

```
000 - 00000000
004 - 00000100
011 - 00001011
015 - 00001111
064 - 01000000
068 - 01000100
075 - 01001011
079 - 01001111
176 - 10110000
180 - 10110100
187 - 10111011
191 - 10111111
240 - 11110000
244 - 11110100
251 - 11111011
255 - 11111111
```

From this set, we can extract a subset of marker ids that minimize the chance of improper detection. The simplest approach is to select ids that differ by 2 or more bits. The optimal subset is as follows.

```
000 - 00000000
011 - 00001011
068 - 01000100
079 - 01001111
176 - 10110000
187 - 10111011
255 - 11111111
```

9.1.21 FAQ: How to interpret a pose

Question:

When the ALVAR library calculates the position of the markers, it also calculates the pose. As a result, a matrix is returned with a whole lot of information. Problem is that we do not know how to interpret the data in this matrix or the way to use it. Is there any documentation on this matrix and how to use it? Or can you supply me more detailed information?

Answer:

You say that when ALVAR computes the position of a marker it also computes a pose. Those two are the same thing. A pose contains a translation (position and direction) as well as an orientation (rotation). These are the two members of the Pose class. Although you have access to these two member via GetTranslation() and GetQuaternion(), it is typical to return a "3d transformation matrix" [1] via GetMatrix(). This matrix essentially contains a translation and a rotation in homogeneous coordinates [2]. A 3d transformation matrix can be decomposed into the following, where P is a 4x4 transformation matrix, R is a 3x3 rotation matrix, T is a 3x1 translation matrix and the 1 is for the homogeneous coordinate.

$$P = \begin{matrix} R & T \\ 0 & 1 \end{matrix}$$

This allows you to easily project points in the marker coordinate frame to the image plane. If X is a 3D point in marker coordinates and x is a 2D point in image plane coordinates.

$$X = \begin{matrix} t_x \\ t_y \\ t_z \\ 1 \end{matrix} \qquad x = \begin{matrix} i_x \\ i_y \\ i_z \\ 1 \end{matrix} \qquad x = PX \qquad x' = \begin{matrix} i_x & i_z \\ i_y & i_z \end{matrix}$$

[1] http://en.wikipedia.org/wiki/Transformation_matrix

[2] http://en.wikipedia.org/wiki/Homogeneous_coordinates

9.1.22 FAQ: Marker position in image coordinates

Question:

How I can get the real position of the marker in my image?

Answer:

If you want the position of the marker in image coordinates, you have to project the origin of the marker into the camera reference frame using the given pose. Here is the code.

```
CvPoint2D64f imagePoint;
camera.ProjectPoint(cvPoint3D64f(0, 0, 0), &pose, imagePoint);
```

The values in imagePoint will be in pixels and correspond to the center of the marker in the image coordinates.

You can also access the pixel locations of the four corners of the marker directly from the marker object. They can be found in the `Marker::marker_corners_img` vector.

9.1.23 FAQ: ALVAR and multithreading

The MarkerDetector class in ALVAR is re-entrant and not thread-safe. This means that you can safely call methods of the class from multiple threads as long as each thread uses a different instance of the class. To put it in other words, you should be fine if you have two threads that each have their own instance of a MarkerDetector class.

9.1.24 FAQ: FernImageDetector does not track very well

Question:

I am using ALVAR to track a planar object, but the system doesn't track very well when the camera is far from the object.

I tried to tune some of the parameters as well, but the result does not improve very much. Basically the tracking is very robust when the camera is close, but it deeply decreases when the frame planar object takes up ~1/4 of the image area.

Answer:

I'm assuming that you are using the FernImageDetector to track a natural feature image and not a fiducial marker. If you are seeing these problems with MarkerDetector then we have a bug or you are not using the library correctly since marker detection in ALVAR is very robust even at farther distances (within reason).

The markerless tracking support is based on a Ferns classifier. The way this works is the classifier must be trained beforehand. During the training phase, the parameters essentially determine a working volume (distance and angle) around the image. During the tracking phase, if the camera moves outside this training volume, then the tracking basically fails to work reliably.

The classifier data is already quite large, so increasing the working volume is probably not the best approach. A nicer approach would be to use the Ferns classifier for initialization and recovery but then rely on feature-based tracking or structure from motion to track the camera pose after the initial detection.

9.1.25 FAQ: Using another version of OpenCV

It is possible to use the version provided by the OpenCV project. In some cases, you have to tell CMake where to find it. You can do this by setting the root path of the OpenCV installation directory to the "OpenCV_ROOT_DIR" CMake variable. This can be done using the CMake GUI (cmakegui) or on the command line.

```
mkdir build
cd build
cmake -DOpenCV_ROOT_DIR=c:\path\to\opencv ..\
```

9.1.26 FAQ: Marker detection speed benchmark

ALVAR 2.0 has been benchmarked on a 4 year old workstation (Intel Xeon X5550, 4 cores, 2.67GHz, 3Gb). SampleMarkerDetector ran at 30 frames per second with 30-40% utilization rate of 1 core. This was for images of 320x240 and 640x480 resolutions. Higher resolutions are most likely not suitable for real-time processing. The framerate was capped at 30 frames per second since the capture classes are blocking and the camera only produces 30 frames every second.

Of course, the more visually cluttered the scene, the more time it takes to analyze all of the detected blobs. Depending on your setup, you can increase the blob size threshold and only process the larger blobs.

9.1.27 FAQ: Marker detection distance benchmark

ALVAR can detect small markers or markers that are far away from the camera. However, the resolution of the image has a lot to do with the limits of marker detection. In practice, the markers in the image should have a diagonal size of over 25px for accurate and robust detection.

Image Resolution (px)	Marker Size (cm)	Distance (m)	Diagonal Marker Resolution (px)
320x240	2.5	0.4	21-22
320x240	5.0	0.8	21-22
320x240	10.0	1.5	23-24
640x480	2.5	0.8	21-22
640x480	5.0	1.5	21-22
640x480	10.0	2.8	23-24

9.1.28 FAQ: Explanation of simple calibration

Question:

In the `SetSimpleCalib(int _x_res, int _y_res, double f_fac=1.)` method, is the specified focal length in millimeters or pixels?

Answer:

The f_fac parameter represents a focal length factor. This factor is defined as "half of the normalized focal length" where the normalization is done with respect to the image width.

x_{res} : x resolution or width
f_{fac} : focal length factor

f_p : focal length in pixels
f_n : noramlized focal length (projective geometry normalizes with respect to xres / 2)

$$f_p = x_{res} \cdot f_{fac}$$

$$f_n = \frac{f_p}{x_{res}\ 2} = \frac{2 \cdot f_p}{x_{res}}$$

f_p in f_n

$$f_n = \frac{2 \cdot x_{res} \cdot f_{fac}}{x_{res}} = 2 \cdot f_{fac}$$

$$f_{fac} = \frac{f_n}{2}$$

This focal length factor is probably not the best API for this method, but the above describes what is currently implemented in ALVAR.

9.1.29 FAQ: Z-axis flip

Question:

When the z-axis of the detected marker is close to the perpendicular of the screen plane, it keeps changing signal very fast (flips from point forward to point backwards and vice-versa).

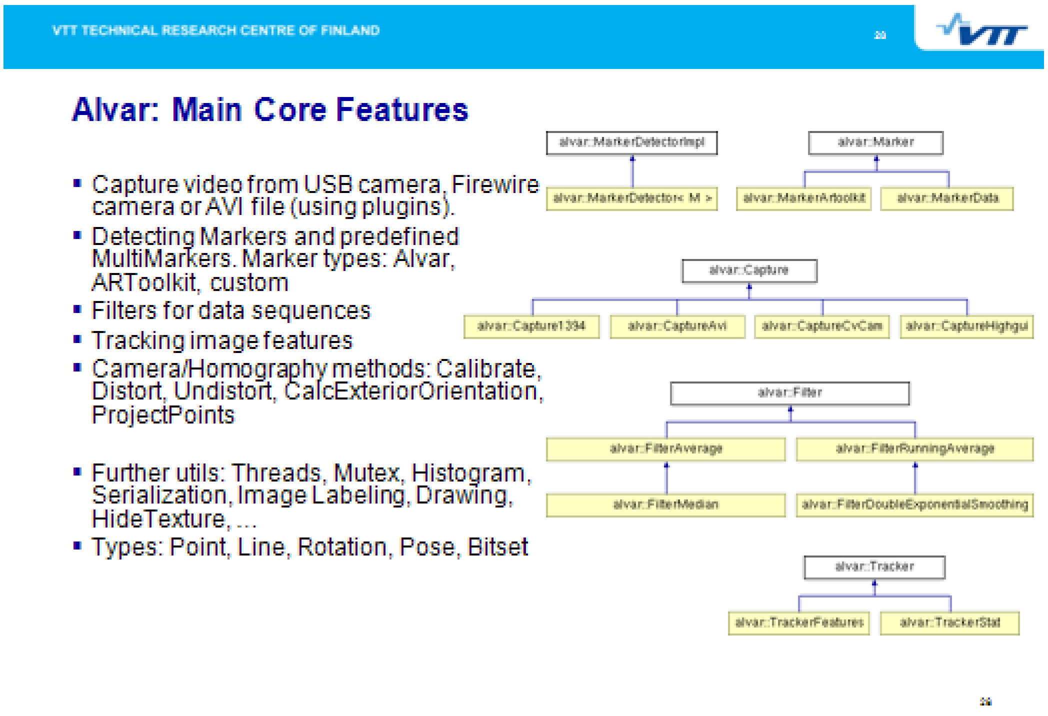
Answer:

It is not that the Z axis is flipping from pointing forwards to backwards, but rather that it flips between angled to the left of the normal and angled to the right of the normal. This is a known problem with the marker detection algorithm and has to do with finding a local minimum when running the optimization routine for the pose estimation.

The above is a bit hard to explain without drawing some pictures. The following paper discusses the issue and provides a solution.

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1717461&tag=1

9.1.30 ALVAR Main Core Features

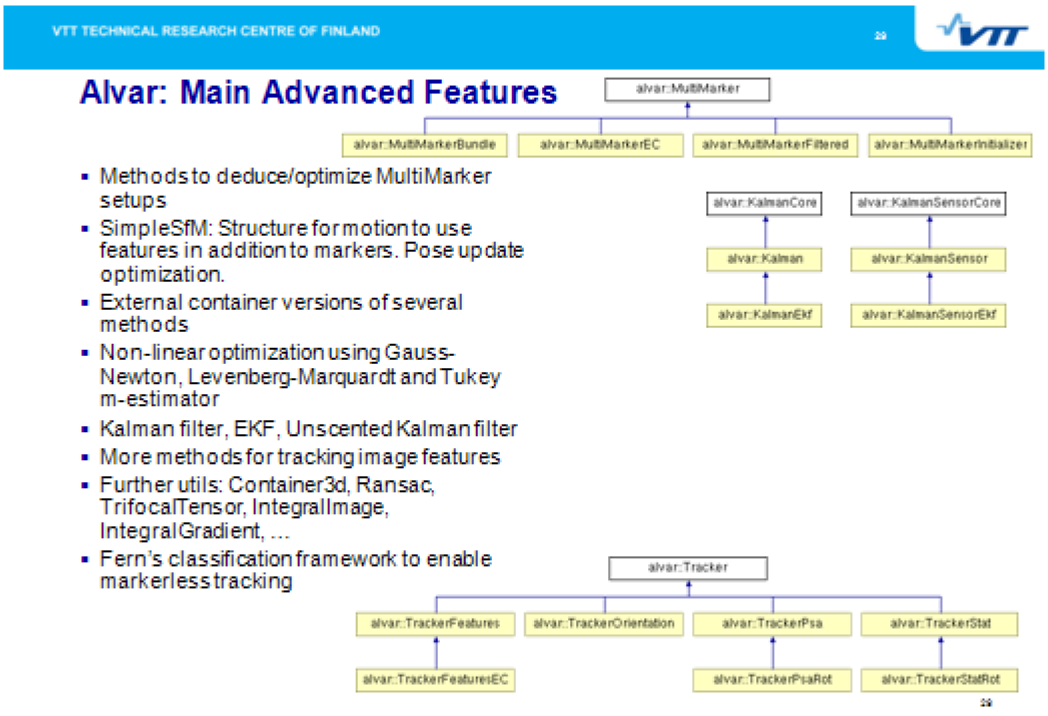


VTT TECHNICAL RESEARCH CENTRE OF FINLAND

Alvar: Main Core Features

- Capture video from USB camera, Firewire camera or AVI file (using plugins).
- Detecting Markers and predefined MultiMarkers. Marker types: Alvar, ARToolkit, custom
- Filters for data sequences
- Tracking image features
- Camera/Homography methods: Calibrate, Distort, Undistort, CalcExteriorOrientation, ProjectPoints
- Further utils: Threads, Mutex, Histogram, Serialization, Image Labeling, Drawing, HideTexture, …
- Types: Point, Line, Rotation, Pose, Bitset

9.1.31 ALVAR Main Advanced Features



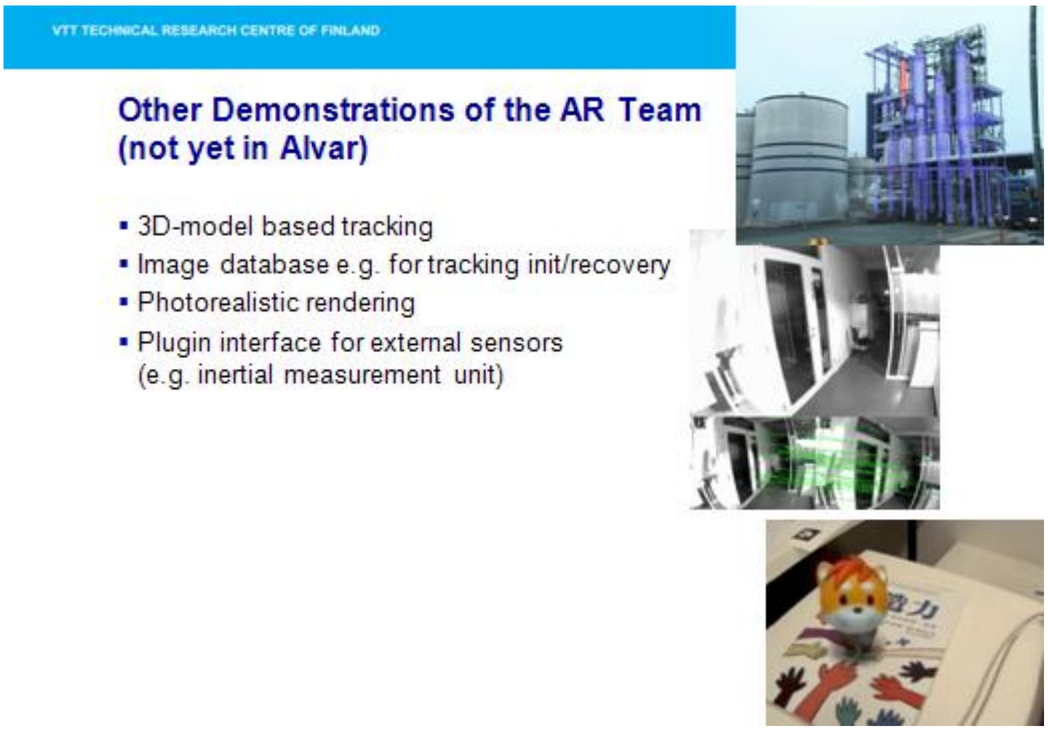
VTT TECHNICAL RESEARCH CENTRE OF FINLAND

Alvar: Main Advanced Features

- Methods to deduce/optimize MultiMarker setups
- SimpleSfM: Structure for motion to use features in addition to markers. Pose update optimization.
- External container versions of several methods
- Non-linear optimization using Gauss-Newton, Levenberg-Marquardt and Tukey m-estimator
- Kalman filter, EKF, Unscented Kalman filter
- More methods for tracking image features
- Further utils: Container3d, Ransac, TrifocalTensor, IntegralImage, IntegralGradient, …
- Fern's classification framework to enable markerless tracking

9.1.32 Appetizer: Features Not Yet in ALVAR

These techniques have been demonstrated by VTT AR Team, but they have not yet been incorporated into ALVAR...



VTT TECHNICAL RESEARCH CENTRE OF FINLAND

Other Demonstrations of the AR Team (not yet in Alvar)

- 3D-model based tracking
- Image database e.g. for tracking init/recovery
- Photorealistic rendering
- Plugin interface for external sensors (e.g. inertial measurement unit)