ARToolKit Plugin For Virtools 4

Building Block Documentation

David COCHARD

Spring 2007

Table of contents

Sorted by functions (documentation order)

Alphabetical order

ARToolKit Manager

Source file	AR_Manager.h ARTK_Manager.cpp AR_Manager.cpp
Categorized in	Manager

Description

All the managers are loaded by Virtools when this latter is started. These managers run continually in background and provide the user with a higher control level. Building blocks need to be activated through the script execution in order to achieve an action, managers can perform tasks at any time and at a higher level.

This particular manager performs some monitoring and cleaning tasks. For example, when the user hits the *reset* button in the Virtools interface, the manager stops the video capture, closes the video stream and some libraries and frees the memory from all the loaded markers. You can easily check its importance by disabling it and work as usual. You will see that you won't be able to run a script twice without rebooting Virtools, simply because the Virtools' *Stop* button doesn't perform any tasks related to ARToolKit and you will get memory access errors while starting it again.

This manager is also used to share complex data between building blocks. Standard data types can be transferred via output parameters but complex structures can't and you need another way to store this information in memory. The manager can handle it; an example can be given considering the multi-marker configuration structure. This structure is created by the "*Load Multi Markers*" building block and used by the "*Detect Multi Video*" building block. The manager creates a global variable and stores the data when the first building block is processed and keeps it in memory until the second one asks for the value.

A manager can also perform perpetual tasks which have to be performed continually without any building block execution. We can give the frame counter functionality to illustrate this concept. This value has to be increased each time a frame is processed; this is made by the manager whatever happens in the script.

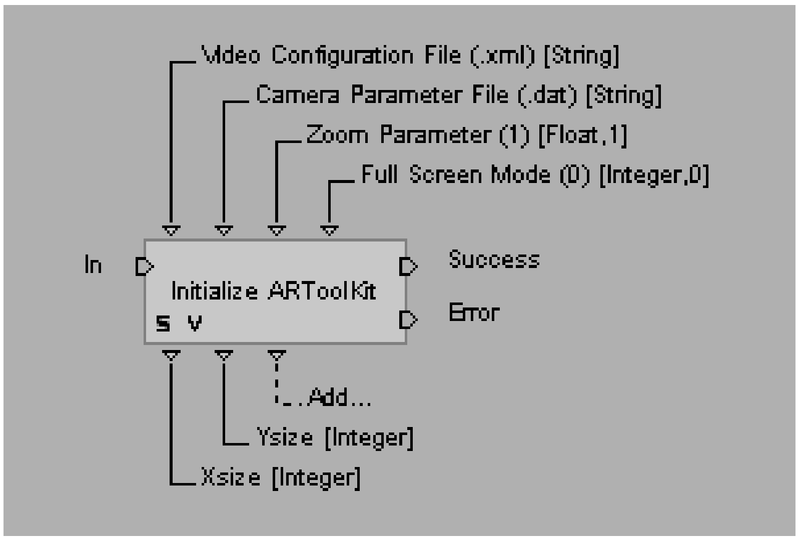
Initialize ARToolKit

Source file	AR_init.cpp
Categorized in	ARToolKit / Start – Close
Example	AR_Start_End.cmo

Description

Initializes ARToolKit using the video and the hardware configuration files provided.

Technical Information


Video Configuration File (.xml) [String]
Camera Parameter File (.dat) [String]
Zoom Parameter (1) [Float,1]
Full Screen Mode (0) [Integer,0]
In
Success
Initialize ARToolKit
S V
Error
..Add...
Ysize [Integer]
Xsize [Integer]

On: activates the process.

Success: is activated if the initialization is successful.
Error: is activated if an error occurs.

Video Configuration File: path of the XML video configuration file.
Camera Parameter File: path of the camera parameter file.
Zoom Parameter: defined a zoom parameter for the final result.
Full Screen Mode: full screen mode (1 enable, 0 disable).

X size: output video width
Y size: output video height

Verbose: information output in the console panel.

Warnings

This building block is a prerequisite for every ARToolKit related action.
Default configuration files can be used but the performances might be reduced.

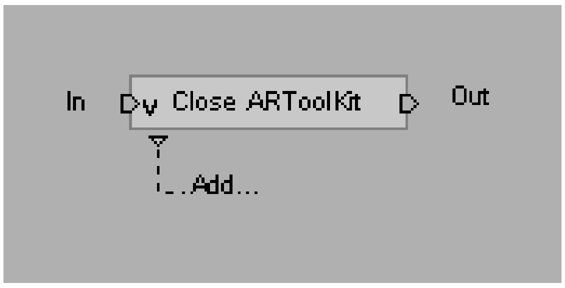
Close ARToolKit

Source file	AR_end.cpp
Categorized in	ARToolKit / Start - Close
Example	AR_Start_End.cmo

Description

Closes ARToolKit properly and frees the memory allocated to the process

Technical Information



In: activates the process.

Out: is activated when the process is completed.

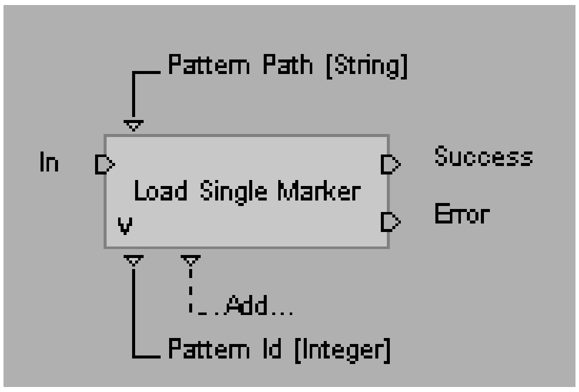
Load Single Marker

Source file	AR_load_patt.cpp
Categorized in	ARToolkit/Single Marker
Example	AR_load_unload_patt.cmo Simple Interaction.cmo

Description

This building block aims to load a pattern which is going to be recognized by the camera. This pattern must be described by a .patt file whose path is given as input parameter (e.g. "C:/Program_Name/Data/4x4_52.patt"). The building block then provides the user with the pattern id which can be used in a mutli pattern configuration.

Technical Information



In: activates the process.

Success: is activated if the marker has been loaded successfully.
Error: is activated if an error occurs.

Pattern Path: file containing the marker description (patt file).

Pattern id: unique pattern identification number.

Warnings

The original ARToolKit can load 50 markers at the most. More patterns generate errors and won't get any identification numbers.

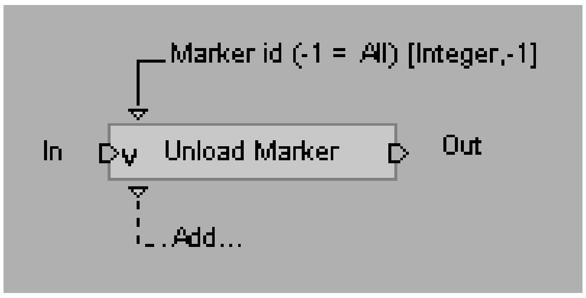
Unload Marker

Source file	AR_unload_patt.cpp
Categorized in	ARToolkit/Tools
Example	AR_load_unload_patt.cmo

Description

This building block unloads one particular or all the markers from memory. This process frees the identification numbers associated to each unloaded marker.

Technical Information



In: activates the process.

Out: is activated when the process is completed.

Marker id: identification number of the marker you want to unload (-1 for all markers).

Warnings

The default input parameter value is set to -1 which means “unload all markers from memory”.

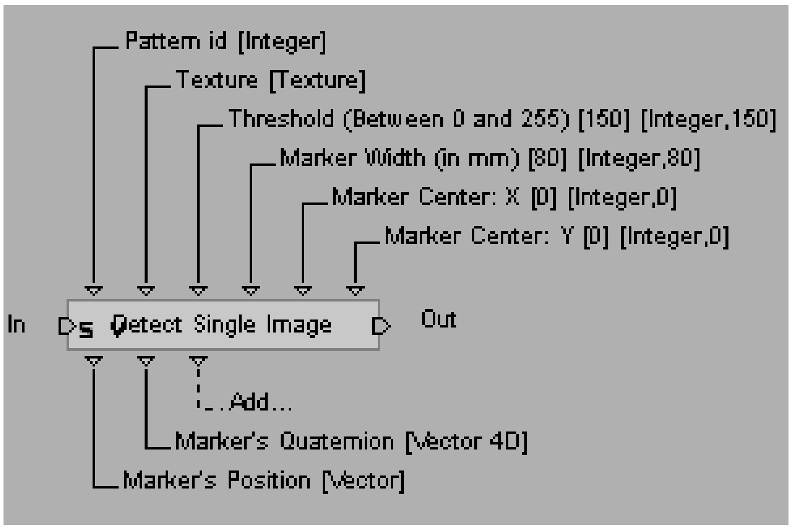
Detect Single Image

Source file	AR_detect_pattern_in_image.cpp
Categorized in	ARToolkit/Single Marker
Example	AR_detect_pattern_in_image.cmo

Description

This object runs the ARToolKit detection algorithm for a single marker given via its identification number on a Virtools texture. It returns the marker's position vector and orientation quaternion into the camera's coordinate system.

Technical Information


Pattern id [Integer]
Texture [Texture]
Threshold (Between 0 and 255) [150] [Integer,150]
Marker Width (in mm) [80] [Integer,80]
Marker Center: X [0] [Integer,0]
Marker Center: Y [0] [Integer,0]
In
Detect Single Image
Out
...Add...
Marker's Quaternion [Vector 4D]
Marker's Position [Vector]

In: activates the process.

Out: is activated when the process is completed.

Pattern id: identification number of the marker you want to get information from.
Texture: image in which you want to run the detection algorithm.
Threshold: luminosity threshold applied to the detection algorithm.
Marker Width: size of the marker in millimeters.
Marker Center X: distance between the marker center and the coordinate system X origin.
Marker Center Y: distance between the marker center and the coordinate system Y origin.

Marker's Position: marker position vector in the camera's coordinate system.
Marker's Quaternion: quaternion giving the marker orientation.

Verbose: information output in the console panel.

Remarks

If you want to get the camera location into the marker’s coordinate system, you have to use the building block "*Get Camera’s Location*” applied to the outputs you get.

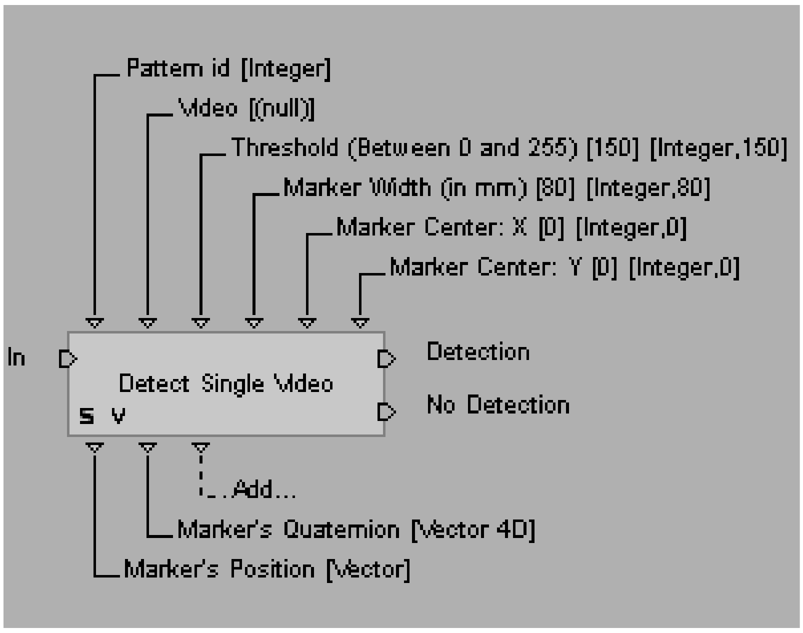
Detect Single Video

Source file	AR_detect_pattern_in_video.cpp
Categorized in	ARToolkit/Single Marker
Example	AR_detect_pattern_in_video.cmo
	Simple Interaction.cmo

Description

This object runs the ARToolKit detection algorithm for a single marker given via its identification number on a video stream. It returns the marker's position vector and orientation quaternion into the camera's coordinate system.

Technical Information


Pattern id [Integer]
Video [(null)]
Threshold (Between 0 and 255) [150] [Integer,150]
Marker Width (in mm) [80] [Integer,80]
Marker Center: X [0] [Integer,0]
Marker Center: Y [0] [Integer,0]
In
Detect Single Video
S V
Detection
No Detection
Add...
Marker's Quaternion [Vector 4D]
Marker's Position [Vector]

In: activates the process.

Detection: is activated if a marker is recognized.
No Detection: is activated nothing has been detected.

Pattern id: identification number of the marker you want to get information from.
Video: video stream (e.g. from a webcam) in which you want to run the detection algorithm.
Threshold: luminosity threshold applied to the detection algorithm.
Marker Width: size of the marker in millimeters.
Marker Center X: distance between the marker center and the coordinate system X origin.
Marker Center Y: distance between the marker center and the coordinate system Y origin.

Marker’s Position: marker position vector in the camera’s coordinate system.
Marker’s Quaternion: quaternion giving the marker orientation.

Verbose: information output in the console panel.

Remarks

If you want to get the camera location into the marker’s coordinate system, you have to use the building block "*Get Camera’s Location*” applied to the outputs you get.

Load Multi Markers

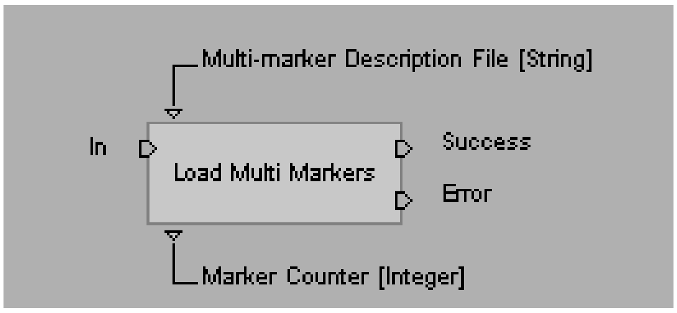
Source file	AR_load_multi_patt.cpp
Categorized in	ARToolkit/Multi Relative Markers
Example	AR_detect_multi_pattern_in_video.cmo

Description

This building block aims to load markers which are going to be recognized by the camera. Each marker should be stored in a .patt file and the multi-marker configuration in a .dat file given as parameter.

The MultiMarker Configuration File contains the list of all markers and their exact positions within the fixed coordonate system (this can be relative to the corner of one marker, the middle of the plane of the set of markers, or an arbitrary position).

Technical Information



In: activates the process.

Success: is activated if markers have been loaded successfully.
Error: is activated if an error occurs.

Multi-marker Description File: path of the *dat* file containing the configuration.

Marker Counter: number of markers loaded from the configuration file.

Remarks

See http://www.hitl.washington.edu/artoolkit/documentation/tutorialmulti.htm for details about the configuration file structure.

Detect Multi Video

Source file	AR_detect_multi_pattern_in_video.cpp
Categorized in	ARToolkit/Multi Relative Markers
Example	AR_detect_multi_pattern_in_video.cmo

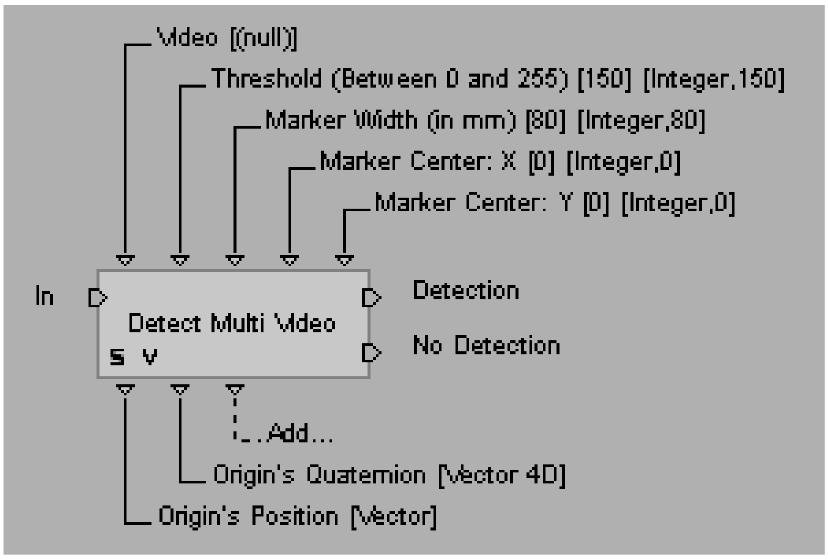
Description

This building block aims to detect a multi-marker structure in a video stream. It must be used with “Load Multi Markers” in order to load the multi-marker configuration.

We use the multiMarker tracking principle: a set of markers are defined based on their relative positions. When at least one marker is visible we can compute the position of the marker set in the camera’s coordinate system.

The MultiMarker Configuration File contains the list of all markers and their exact positions within the fixed coordinate system (this can be with respect to the corner of one marker, the middle of the plane of the set of markers, or an arbitrary position).

Technical Information


Video [(null)]
Threshold (Between 0 and 255) [150] [Integer,150]
Marker Width (in mm) [80] [Integer,80]
Marker Center: X [0] [Integer,0]
Marker Center: Y [0] [Integer,0]
In
Detect Multi Video
S V
Detection
No Detection
Add...
Origin's Quaternion [Vector 4D]
Origin's Position [Vector]

In: activates the process.

Detection: is activated if a marker is recognized.
No Detection: is activated nothing has been detected.
.

Video: video stream (e.g. from a webcam) in which you want to run the detection algorithm.
Threshold: luminosity threshold applied to the detection algorithm.

Marker Width: size of the marker in millimeters.
Marker Center X: distance between the marker center and the coordinate system X origin.
Marker Center Y: distance between the marker center and the coordinate system Y origin.

Origin's Position: coordinate system origin's position vector.
Origin's Quaternion: coordinate system origin's orientation quaternion.

Verbose: information output in the console panel.

Load Independent Markers

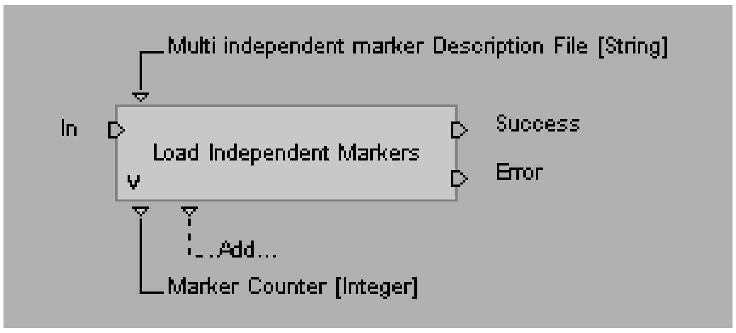
Source file	AR_load_independent_patt.cpp
Categorized in	ARToolkit/Multi Independent Markers
Examples	AR_detect_independent_patterns_in_video01.cmo
	AR_detect_independent_patterns_in_video02.cmo

Description

This building block aims to load the multi pattern configuration in which the markers are totally independent. This is useful when the user wants to use more than one pattern, each of them associated to a different augmentation.

This is different from the multi marker where all the markers are linked together (relative positions). Here every single pattern has its own configuration and is considered autonomous.

Technical Information



In: activates the process.

Success: is activated if markers have been loaded successfully.
Error: is activated if an error occurs.

Multi independent Description File: path of the file containing the configuration.

Marker Counter: number of markers loaded from the configuration file.

Remarks

This kind of configuration can also be set up using several “Load Single Marker” and “Detect Single Video” building blocks but this is not efficient and can significantly decrease the performances, see the following CMO file for illustration:

Multi independant markers [BAD METHOD - without loadMultiple].cmo

Detect Independent Video 01

Source file	AR_detect_independant_patterns_in_video_01.cpp
Categorized in	ARToolkit/Multi Independent Markers
Examples	AR_detect_independent_patterns_in_video01.cmo

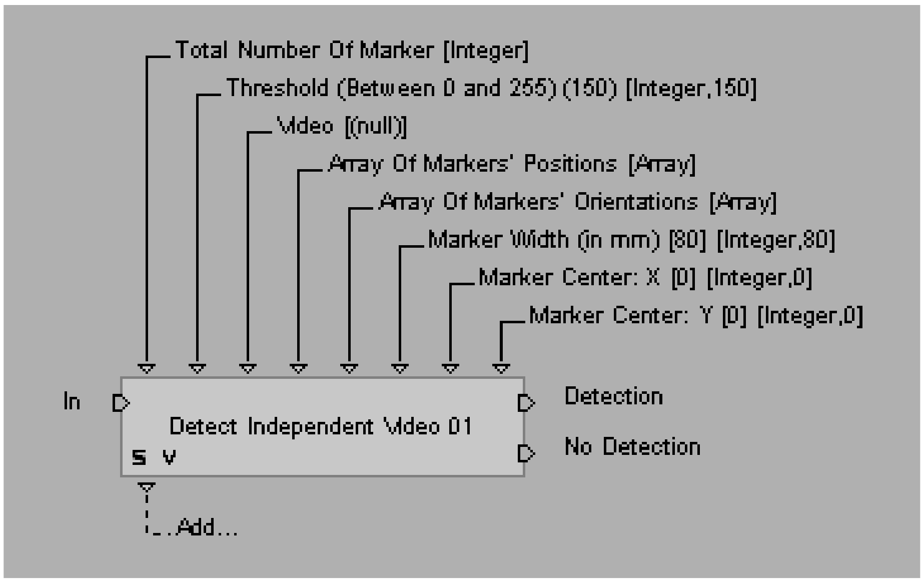
Description

This building block concerns a configuration in which markers are totally independent. This is useful when the user wants to use more than one pattern, each of them associated to a different augmentation.

This is different from the multi marker where all the markers are linked together (relative positions). Here every single pattern has its own configuration and is considered autonomous.

The 1st version of the detection building block for this method doesn't have any output parameter. The user has to provide 2 arrays (created beforehand in Virtools) and the markers' positions and orientations will be stored in these arrays.

Technical Information



Total Number Of Marker [Integer]
Threshold (Between 0 and 255) (150) [Integer,150]
Video [(null)]
Array Of Markers' Positions [Array]
Array Of Markers' Orientations [Array]
Marker Width (in mm) [80] [Integer,80]
Marker Center: X [0] [Integer,0]
Marker Center: Y [0] [Integer,0]
In
Detect Independent Video 01
S V
Detection
No Detection
Add...

In: activates the process.

Detection: is activated if a marker is recognized.
No Detection: is activated nothing has been detected.
.

Total Number: The number of markers to consider.
Threshold: luminosity threshold applied to the detection algorithm.
Video: video stream (e.g. from a webcam) in which you want to run the detection algorithm.
Marker's Positions: Array containing the detection results concerning position vectors.
Marker's Orientations: Array containing the detection results concerning quaternions.
Marker Width: size of the marker in millimeters.
Marker Center X: distance between the marker center and the coordinate system X origin.
Marker Center Y: distance between the marker center and the coordinate system Y origin.

Verbose: information output in the console panel.

Detect Independent Video 02

Source file	AR_detect_independant_patterns_in_video_02.cpp
Categorized in	ARToolkit/Multi Independent Markers
Examples	AR_detect_independent_patterns_in_video02.cmo

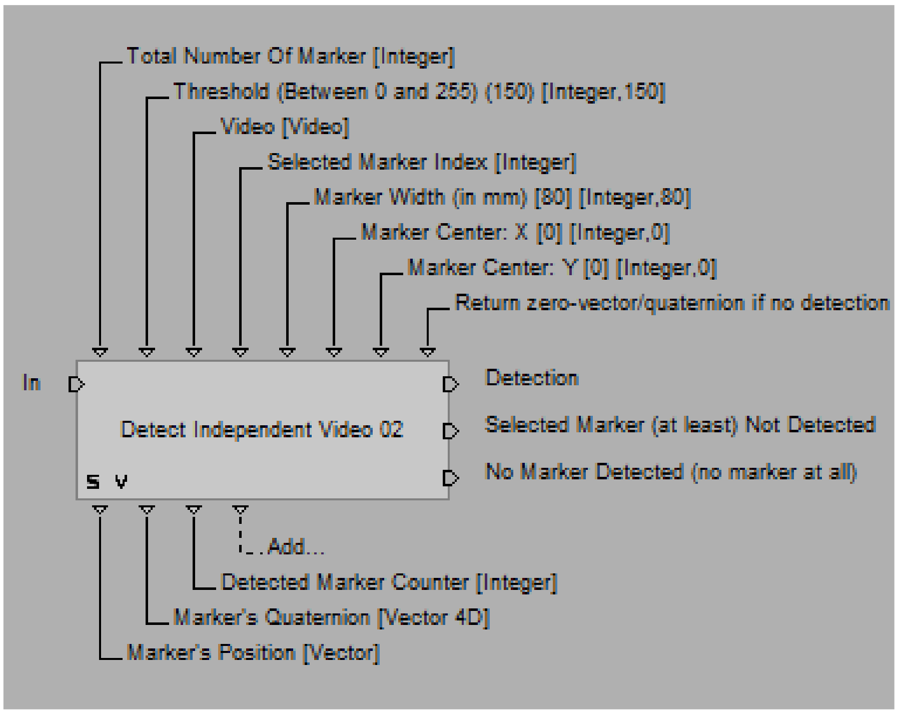
Description

This concerns the building of a configuration in which the markers are totally independent. This is useful when the user wants to use more than one pattern, each of them associated to a different augmentation.

This is different from the multi marker where all the markers are linked together (relative positions). Here every single pattern has its own configuration and is considered as autonomous.

In this second version, the user has to provide a marker index as input parameter and the building block will return the position and the orientation of this marker (and only this one). It can make loop executions easier (the user knows how many markers he is dealing with thanks to the building block "*AR_load_independant_patt*").

Technical Information


Total Number Of Marker [Integer]
Threshold (Between 0 and 255) (150) [Integer,150]
Video [Video]
Selected Marker Index [Integer]
Marker Width (in mm) [80] [Integer,80]
Marker Center: X [0] [Integer,0]
Marker Center: Y [0] [Integer,0]
Return zero-vector/quaternion if no detection
In
Detect Independent Video 02
S V
Detection
Selected Marker (at least) Not Detected
No Marker Detected (no marker at all)
Add...
Detected Marker Counter [Integer]
Marker's Quaternion [Vector 4D]
Marker's Position [Vector]

In: activates the process.

Detection: is activated if a marker is recognized.
Selected Marker Not Detected: is activated if the marker which identifier is given as input parameter is not detected (at least)
No Detection: is activated nothing has been detected (no marker at all).

Total Number: The number of markers to consider.
Threshold: luminosity threshold applied to the detection algorithm.
Video: video stream (e.g. from a webcam) in which you want to run the detection algorithm.
Selected Marker Index: Marker from which you want to get information.
Marker Width: size of the marker in millimeters.
Marker Center X: distance between the marker center and the coordinate system X origin.
Marker Center Y: distance between the marker center and the coordinate system Y origin.
Return Zero-vector… : will any value be returned if there is no detection.

Marker's Position: marker position vector in the camera's coordinate system.
Marker's Quaternion: quaternion giving the marker orientation.
Detected Marker Counter: how many markers have been detected.

Verbose: information output in the console panel.

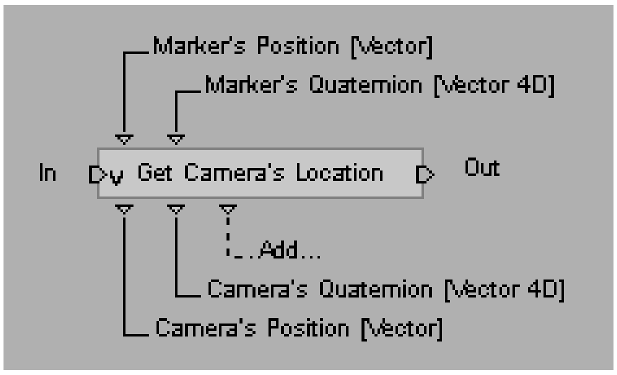
Get Camera’s Location

Source file	AR_get_camera_location.cpp
Categorized in	ARToolkit/Tools
Examples	AR_get_camera_location.cmo

Description

The detection methods always give you the marker's location according to the camera. It's also intuitive to consider the marker static and the real camera moving. That is to say, we want to obtain the position of the camera in the marker's coordinate system. For example in a complex augmentation containing a lot of objects, it could be easier and more efficient to move the camera than the whole 3D models.

Technical Information



Marker's Position [Vector]
Marker's Quaternion [Vector 4D]
In
Get Camera's Location
Out
Add...
Camera's Quaternion [Vector 4D]
Camera's Position [Vector]

In: activates the process.

Out: is activated when the process is completed.

Marker’s Position: given position vector (in camera’s coordinate system).
Marker’s Quaternion: given orientation quaternion (in camera’s coordinate system).

Camera’s Position: camera’s position vector into the marker’s coordinate system.
Camera’s Quaternion: camera’s orientation quaternion into the marker’s coordinate system.

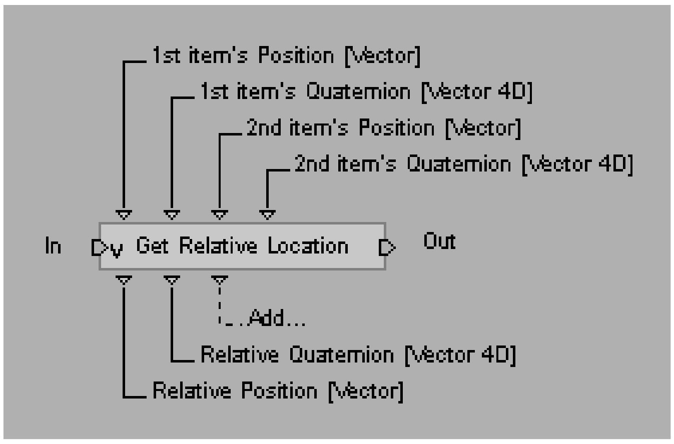
Get Relative Location

Source file	AR_get_relative_location.cpp
Categorized in	ARToolkit/Tools
Examples	AR_get_camera_location.cmo

Description

Considering two markers 1 and 2 and their locations (positions and orientations) in the SAME coordinate system, this building block gives you marker 2's location in marker 1's coordinate system (relative location).

Technical Information


1st item's Position [Vector]
1st item's Quaternion [Vector 4D]
2nd item's Position [Vector]
2nd item's Quaternion [Vector 4D]
In
Get Relative Location
Out
..Add...
Relative Quaternion [Vector 4D]
Relative Position [Vector]

In: activates the process.

Out: is activated when the process is completed.

1st item's Position: position vector of the first item.
1st item's Quaternion: orientation quaternion of the first item.
2nd item's Position: position vector of the second item.
2nd item's Quaternion: orientation quaternion of the second item.

Relative Position: 1st item's position vector into the 2nd item's coordinate system.
Relative Quaternion: 1st item's orientation quaternion into the 2nd item's coordinate system.

Warnings

The two given locations have to be expressed in the same coordinate system in order to get a coherent result. If it's not the case, you first have to translate one of the two item into the other one's coordinate system.

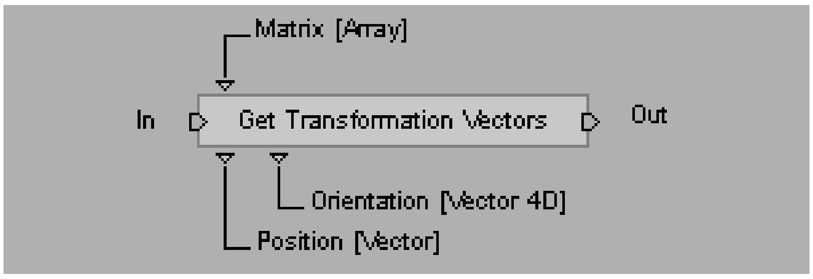
Get Transformation Vectors

Source file	AR_matrix_to_vectors.cpp
Categorized in	ARToolkit/Tools
Examples	AR_get_camera_location.cmo

Description

This tool returns the position vector and the orientation quaternion of a given item from its transformation matrix.

Technical Information



In: activates the process.

Out: is activated when the process is completed.

Matrix: transformation matrix

Position: position vector.
Orientation: orientation quaternion.

Warnings

The user must provide the building block with an array (3*4) created in Virtools beforehand in order to store the processed transformation matrix. This is due to the fact that Virtools cannot give an array through an output parameter.

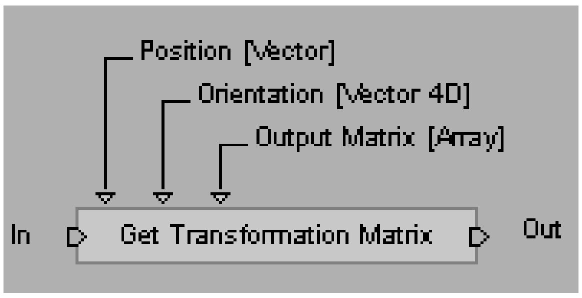
Get Transformation Matrix

Source file	AR_vectors_to_matrix.cpp
Categorized in	ARToolkit/Tools
Examples	AR_get_camera_location.cmo

Description

This tool returns the transformation matrix of a given item from its position vector and its orientation quaternion.

Technical Information



In: activates the process.

Out: is activated when the process is completed.

Position: position vector.
Orientation: orientation quaternion.
Output Matrix: transformation matrix.

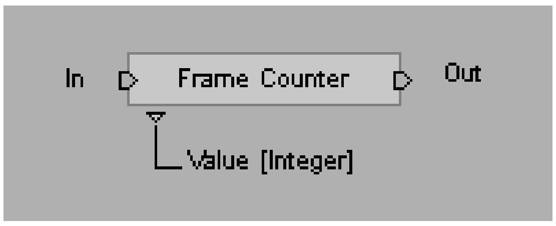
Frame Counter

Source file	AR_frame_counter.cpp
Categorized in	ARToolkit/Tools
Example	AR_frame_counter.cmo

Description

This building block returns the number of frame displayed since the "*play*" button has been hit. The number comes from the manager and is reinitialized when the user presses "*reset*"

Technical Information



In: activates the process.

Out: is activated when the process is completed.

Value: number of frames displayed since the script started.

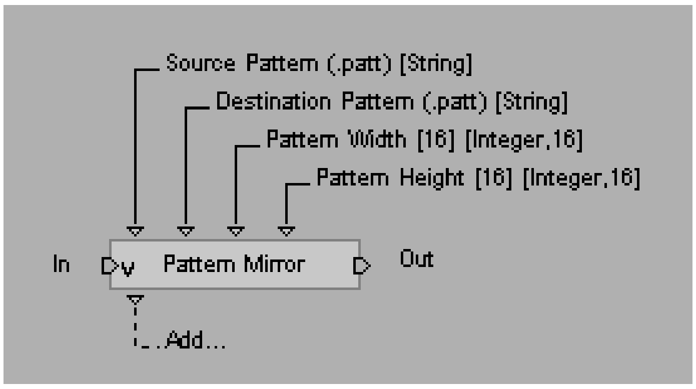
Pattern Mirror

Source file	ARP_mirror_patt.cpp
Categorized in	ARToolkit/ Tools
Example	ARP_mirror_pattern.cmo

Description

This tool reads a pattern description file and saves it mirrored in its y-direction.

Technical Information


Source Pattern (.patt) [String]
Destination Pattern (.patt) [String]
Pattern Width [16] [Integer,16]
Pattern Height [16] [Integer,16]
In
Pattern Mirror
Out
Add...

In: activates the process.

Out: is activated when the process is completed.

Source Pattern: absolute path of the source file.
Destination Pattern: absolute path of the output file.
Pattern Width: width of the given pattern.
Pattern Height: height of the given pattern.

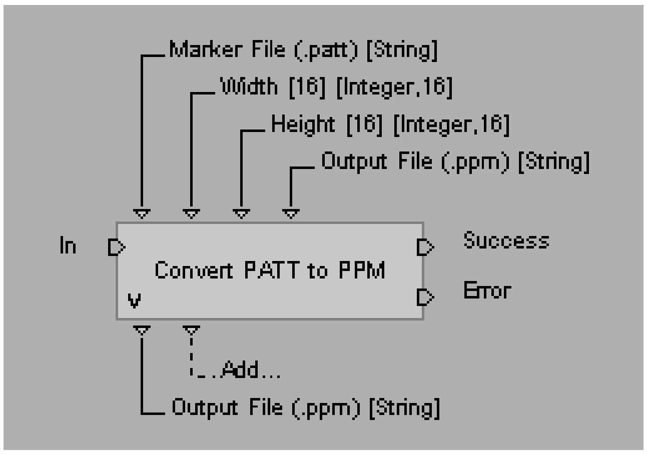
Convert PATT to PPM

Source file	ARP_patt_to_ppm.cpp
Categorized in	ARToolkit/ Tools
Example	ARP_patt_to_ppm.cmo

Description

This tool reads a pattern description file and saves the pattern as a Portable PixMap (.ppm) picture file.

Technical Information



In: activates the process.

Success: is activated when the process is completed.
Error: is activated if an error occurs.

Marker File: absolute path of the pattern description file.
Width: width of the given pattern.
Height: height of the given pattern.
Output File: absolute path of the picture output file (optional)

Output File: absolute path of the picture output file

Remarks

The user can specify the path for the output PPM file (fourth input parameter) but this is optional. If this remains NULL, the output file will have the same name (with a PPM extension) and location as the source file. The output path can still be known thanks to the output parameter.

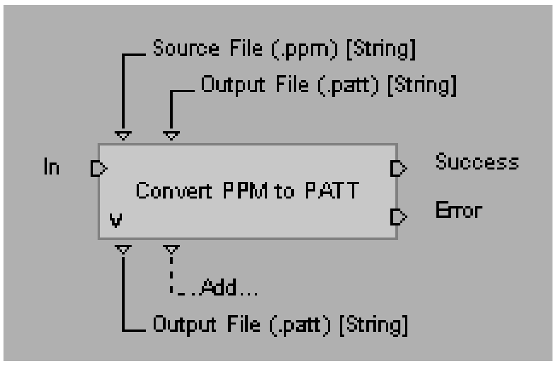
Convert PPM to PATT

Source file	ARP_ppm_to_patt.cpp
Categorized in	ARToolkit/ Tools
Example	ARP_patt_to_ppm.cmo

Description

This tool reads a Portable PixMap file (binary only) and saves it as an ARToolKit pattern file (.PATT).

Technical Information


Source File (.ppm) [String]
Output File (.patt) [String]
In
Convert PPM to PATT
Success
Error
..Add...
Output File (.patt) [String]

In: activates the process.

Success: is activated when the process is completed.
Error: is activated if an error occurs.

Source File: absolute path of the original picture file.
Output File: absolute path of the pattern description output file (optional)

Output File: absolute path of the pattern description output file.

Remarks

The user can specify the path for the output PATT file (second input parameter) but this is optional. If this remains NULL, the output file will have the same name (with a PATT extension) and location as the source file. The output path can still be known thanks to the output parameter.