52 Card Psycho: Implementation details

Andrei M. Rothenstein and Mikhail Sizintsev

July 9, 2008

1 Overview

The project "52 Card Psycho" relies on overlaying a video clip imagery onto the image taken from the camera in the real time by employing the recent advances in the Augmented Reality Tracking.

Each particular clip is tied to the so-called marker – a black-and-white pattern of squares. Each pattern has its own identity. The patterns and the technology we used is the ARTag Fiducial Marker System for augmented reality applications, developed by Mark Fiala for the National Research Council of Canada. [Fia05].

The "52 card psycho" application has been realized as a patch for MAX/MSP (currently we use the latest version 5.0) that employs a Max/MSP/Jitter External Object. The development of the External Object concerned with two major steps:

1. Interfacing with MAX to feed the video and control data and output the processed video (overlayed clips). This is implemented as a Max/MSP/Jitter External Object compiled for Mac OSX/Intel. The object is called `fcl.jitagmolto` and can be invoked from a patcher window from within Max/MSP 4.6 and higher.

2. Interfacing with ARTag to do the actual video processing, specifically marker detection, and rendering images.

2 Interface with MAX/MSP and Jitter

This section describes how the Max/MSP/Jitter External Object is implemented.

The purpose of the external object is to:

- accept a video input (video frames as Jitter matrices via a Jitter matrix inlet) of a scene containing zero or more ARTag markers,

- accept zero or more additional video inputs (also via Jitter matrix inlets) called *overlays*.

- define a number of Jitter object attributes that allow the Max user to specify information about how the overlays should be rendered, such as which ARTag ID should correspond to a given overlay input and how they should be positioned relative to the tag in the input image, and overall behaviour of the rendering.

- output a video output (via a Jitter matrix outlet) consisting of the video input, but with overlay frames positioned relative to the position of corresponding ARTag markers, in accordance with the Jitter object attributes described above.

- work consistently for ARGB, UYVY or grey-level videos of any frame size.

The object is implemented as an extension of a Jitter Matrix Operation (MOP) object.

The Jitter class definition is implemented in the body of a `main()` function in a Matrix Wrapper module, and defines a Matrix Operation with `overlays`+1 matrix inlets, one matrix outlet, one list outlet and one dump outlet. The attributes defined in Table 1 are then registered.

attribute	**description**	**notes**
`autoclear`	clear output image on empty bang	boolean
`livevideo`	output video input with overlays instead of chroma key color	boolean
`chroma_r`	red channel of chroma key color	[0...255], default is 0
`chroma_g`	green channel of chroma key color	[0...255], default is 0
`chroma_b`	blue channel of chroma key color	[0...255], default is 0
`chroma_a`	alpha channel of chroma key channel	[0...255], default is 0
`overlays`	number of overlay video inputs	Read-only.
`tagid`i	ARTag id corresponding to overlay i	$1 \leq i \leq$ `overlays`
`scaling`i	image scale factor for overlay i	$1 \leq i \leq$ `overlays`
`xoffset`i	x translation of image for overlay i	$1 \leq i \leq$ `overlays`
`yoffset`i	y translation of image for overlay i	$1 \leq i \leq$ `overlays`
`height`i	apparent height of overlay i from surface in 3d	$1 \leq i \leq$ `overlays`

Table 1: `fcl.jitagmolto` Max/MSP/Jitter external object attributes.

In the present implementation, only a single `fcl.jitagmolto` object can be used in one instance of Max, because a bug in the ARTag initialization routine of the ARTag library allows it to only be called once. We expect this limitation to be lifted with the next release of the ARTag library.

3 Interface with ARTag

This section describes how the external object calls marker detection routines and renders output video frames. The typical processing stage consists from ARTag object initialization, frame processing and ARTag object destruction.

Initialization has to specify the assumed input image dimensions, colour bit depth and focal length. Note that camera focal length is not used in marker detection and is only needed for 3D object rendering; thus, is not crucial for the current project which overlays 2D images only. Functions involved are `init_artag` and `artag_set_camera_params`.

Marker detection is conveniently performed by calling `artag_find_objects` function which detects all markers and pre-set the internal ARTag datastructures for the subsequent processing. To check whether a particular marker has been found, the boolean `artag_is_object_found` function must be called. The target image/video overlaying on the detected marker is done as explained in details in section 3.1 below.

Object destruction is performed wia `close_artag` method plus explicit deletion of all structures used in the patch. The current implementation of ARTag has a bug in `close_artag` function, which might introduce the unexpected behaviour when patch is stopped and exited, but MAX/MSP process not killed explicitly.

3.1 Target image overlaying on the detected

Once all markers has been detected via calling `artag_find_objects` function, the `artag_project_point` function can be used to project the point from the marker coordinate frame to the camera image

coordinate frame and the point-by-point rendering can be done in principle. However, this is a very computationally demanding task for two reasons: (i) The function `artag_project_point` has to be called for every point in the overlaying image (ii) The processing time depends on the size of overlaying image and not on the size of the detected marker, which is much smaller in practice.

It is very important for the image overlaying process to be proportional to the number of pixels occupied by a marker in the input image – in this case the processing would be essentially independent on the number of markers and bound by the number of pixels in the input image, which is fixed. Since we want to map the points from the marker plane to the image plane, the corresponding function can be easily described via a homography H (linear multidimensional relationship) described by the equation (1) below. The overlaying process from marker onto the input image plane would trivially involve the application of inverse, i.e. H to each point in the image.

$$p_{marker} \simeq \mathsf{H} p_{image}, \tag{1}$$

where H is the invertible 3×3 matrix, p_{marker} is the homogeneous coordinate of a point in the marker plane and p_{image} in the homogeneous coordinate of a point in the image plane, $\simeq$ is the equality up to scale. The homography H can be computed from equation (1) once at least 4 correspondence points are known (they can be found explicitly using `artag_project_point` function for corners of the marker image). For detailed description of projective geometry and homography computation please refer to [HZ04].

Prior to overlaying an image onto the marker, we estimate the bounding box in the input image where the marker has been found to localize the processing.

References

[Fia05] Mark Fiala. ARTag, a fiducial marker system using digital techniques. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 590–596, 2005.

[HZ04] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision. Second Edition.* Cambridge University Press, Cambridge, UK, 2004.