ED Vision User's Manual

for EDVision v2.5
6/21/2004

Rick Kjeldsen
IBM T.J. Watson Research Center
PO Box 704
Yorktown Heights, NY 10598

fcmk@us.ibm.com

Contents

Overview

Vision-based User Interfaces
EDVision allows creation of user interfaces based on visual recognition of a user's actions. With EDVision, software events can be triggered by user interactions with any part of the environment. Physical, printed or projected objects can become elements of a user interface.

This is accomplished using "interface widgets" which are located in the camera's field of view such that they overlay objects in the environment. Widgets are arranged relative to each other by the application designer, and their exact location in the image is determined by a calibration procedure that registers them with either the real world or a projected image. When the widgets detect a touch or other user action at their location, they send events back to the application.

EDVision and ED
EDVision was originally designed to provide the vision-based user interface for the Everywhere Display. The pan-tilt camera of the Everywhere Displays projector provides a video stream to EDVision. While the camera can independently view any area within a room, it is generally directed towards a projected graphical display. User activity is interpreted by EDVision on the basis of their interactions with real or projected objects.

EDVision consists of a Windows program running on a machine where the camera is attached. It can be the same machine where the application or other modules of ED are running, or any other machine on the same network. Running the camera control module and EDVision on the same machine makes some operations easier.

Usage overview
EDVision is controlled by both the Vision Interface Markup Language API (VIML) [reference the EDML documentation] and via the local GUI. VIML is the method by which an application or the Everywhere Display software creates and controls a vision-based user interface. The local GUI is used to calibrate the system, monitor operation, test and debug interactions, and tune parameters.

User interactions are defined as sequences of "configurations". Each configuration is a collection of "widgets". Each widget detects a particular type of user interaction at a specific location, and generates local or VIML events in response. The library of widgets and the events they generate is predefined (the current set of widgets is outlined later in this document).

New configurations can be defined either by the VIML interface or by the local GUI. Whenever a configuration is defined, the definition is saved locally and can be reloaded at a later time. This allows processing parameters to be tuned for a particular configuration and recalled whenever the configuration is reused.

Typically a configuration is mapped to a "surface" before it is used, meaning that the configuration, defined in a rectilinear coordinate system, is warped into an arbitrary quadrilateral in the video image. This is equivalent to the interface being "projected" onto a planar surface in the environment. This warping determines where in the image each widget is located. A surface definition consists of the four corners of a (usually) bounding quadrilateral. A surface can also contain information about the size and orientation in which a user's hand will appear. A surface is most often defined and calibrated at the EDVision GUI, but can also be set using VIML.

Hardware Requirements

System Performance

More is better! Usability depends heavily on frame rate, which increases directly with system performance. Our test system is 1.6 MHz P4 with 512 MB memory. This typically gives near video rate performance (20-30 FPS) on 320x240 resolution video, but note that the frame rate can vary greatly with the expected size of the user's hand and the number of widgets active simultaneously.

The practical hardware minimums are:

CPU: Pentium class machine, 500 MHz or better
Slow processors will achieve barely acceptable performance on low (160x120) resolution video. Good performance (15 FPS or better) on 320x240 video generally requires 1 GHz or better.

Memory: 128 MB will probably work, 256 MB should be fine, 512 MB is more comfortable

OS: Any recent Windows OS.
Tested on Win 2000 and XP, it should work on Win 98 (Second Edition) and ME as well.

Video

EDVision requires a Video For Windows (VfW) video source. Typically the source will be a good quality frame grabber capturing video from a Pan/Tilt camera, but it can be any VfW source, such as an inexpensive USB camera.

Our test systems have had good results with the following video hardware:

- Belkin USB VideoBus
- Osprey-100 video card. Note that this card is limited to 15 frames per second with our software.
- Logitech QuickCam Pro series of USB cameras.

Note that video cards from Matrox generally do NOT support VfW.

When it starts for the first time on a new machine, EDVision will automatically detect all VfW sources on your machine. If there are multiple sources, the user will be asked to select between them. The choice will be remembered, so it not be repeated each time the system is started. At any time the user can change what video source is used with the Video Control Dialog (see below).

EDVision can also take in video from AVI files. This is good for debugging and testing when the hardware is not available.

Installation

EDVision consists of ten files:

- EDVision .exe
- EdTrackingWidgets.dll
- VIMain.dll
- ViStdMod.dll
- EDVision.vis
- HighGUI.dll
- CV.dll
- handicon.bmp
- FingerTemplate.bmp
- DotTemplate.bmp

Installation is simply a matter of placing the files in the same directory, and executing EDVision.exe.

It may be useful to understand something about what each file does:

EDVision.exe contains the VIML parser, the local GUI and the ED specific control logic.

VIMain.dll and ViStdmod.dll contain the core of the vision processing system including video capture and display and management of image processing components.

EdTrackingWidgets.dll contains the definitions of the image processing components.

CV.dll and HighGUI.dll allow widgets to use OpenCV routines. OpenCV is not currently used by EDVision, but these are provided for future expansion.

Setup File

EDVision.vis is referred to as the "setup file". The First part contains the widget definitions (i.e. the how the components defined in EdTrackingWidgets.dll are combined to create a widget). The second part contains surface calibration information, the definitions of configurations, and any widget parameters that have been changed from their default values.

The setup file is text, and so human readable, HOWEVER the system is very sensitive to problems in the setup file, so we recommend you do not make changes manually! One exception to this is when you are installing a new version of EDVision and wish to retain existing configuration or surface definitions or tuning. In this case the setup file must be edited to combine the top of the new setup file with the bottom of the old one. Comments in the setup file will guide you through this process.

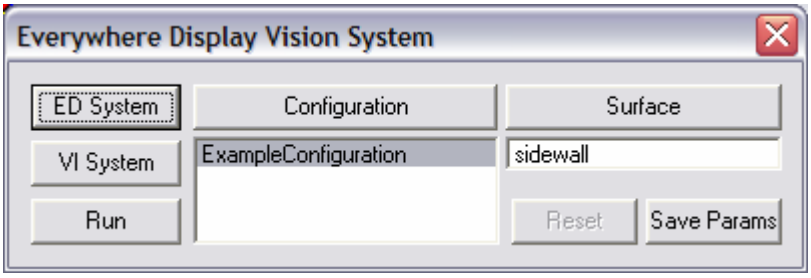
It is recommended that you periodically backup a known good setup file. This will protect your local definitions in the event the setup file is corrupted or lost.

GUI

The EDVision GUI consists of a set of dialog boxes, a video window, and a parameter control window.

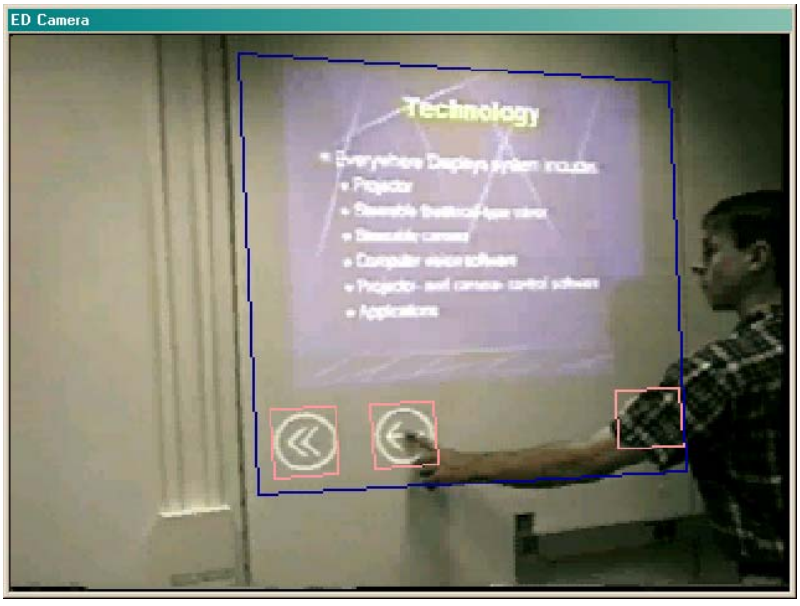
When you start EDVision, the main dialog and a video window are displayed. Other modeless dialog boxes can be brought up to control ED system connectivity, the video source, configurations, surfaces and to control widget parameters. Each of these will be described below.

The Main dialog shows basic information about what configurations and surfaces are currently being used. It has buttons to display the other dialog boxes, and save system parameters.



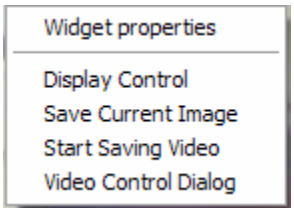
When the system is being controlled remotely (via VIML), this window is often sufficient to monitor the current state of the system. The window on the left shows all the configurations that are currently being used. When a configuration is selected, the window on the right shows the surface on which it is running.

The Video window defaults to show the video stream with the surface, widgets and search regions drawn over it.



Buttons are shown in light red, tracking regions in green, the surface quadrilateral in blue, and the widget search regions in black. When the system is tracking a user's hand, the recent path of the fingertip is shown in green. Some component parameters can modify this display to show additional information that may be useful for debugging.

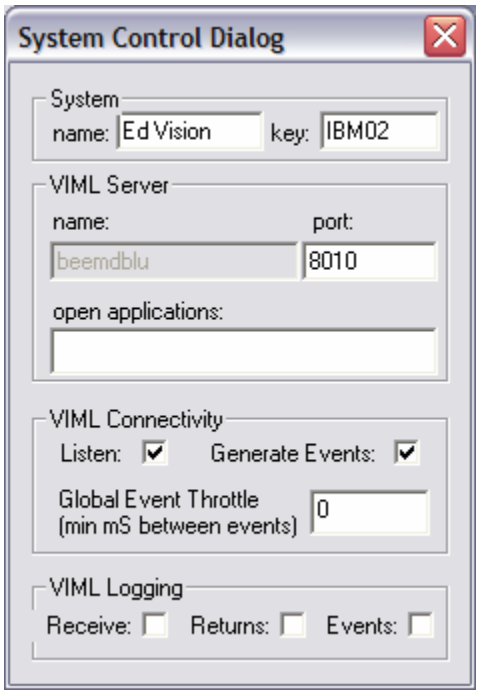
Right-clicking on the video window brings up this menu of operations:



From here you can do several operations, including bringing up the Widget Properties dialog for adjusting component parameters (see below), bringing up the Video Control dialog for controlling the input video stream (see below), plus tools to save the current image or video stream and adjust the video display (show the frame rate, and adjust (flip / mirror) the video).

The following modeless dialog boxes can be displayed using buttons on the Main Dialog to allow local control of the system.

The ED System Control dialog shows information about, and allows control over the connection between EDVision and ED applications.



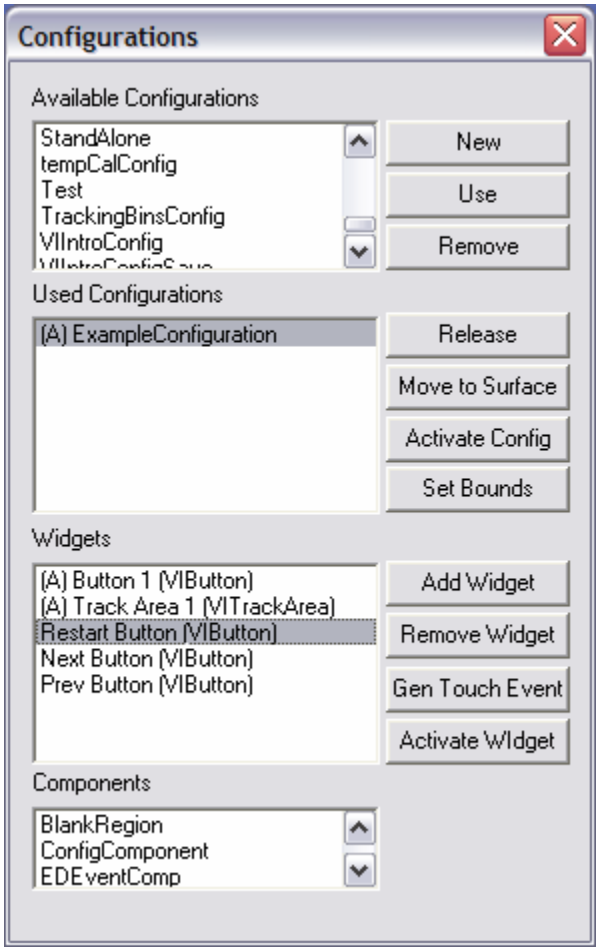
The system Name, Key and Port can all be changed to prevent interference between multiple ED systems on the same network. These changes must be made in conjunction with corresponding changes to the application using EDVision.

The VIML Connectivity section allows you to isolate EDVision from the application. Unchecking the Listen button will make EDVision ignore incoming VIML commands. Unchecking the Generate Events button will stop widgets from generating outgoing VIML events.

The Global Event Throttle allows you to slow down the rate at which events are generated back to the application. At full speed, EDVision can generate nearly 30 events per second, especially from tracking areas, which can swamp some applications, and is often more than are needed.

The VIML Logging section allows EDVision to save a record of VIML activity to a file.

The Configuration dialog allows local control over configurations and widgets.



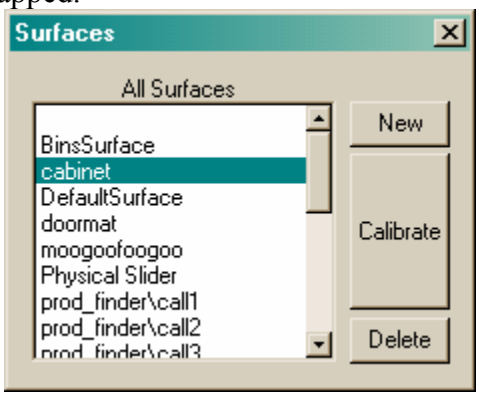
All the configurations in the current setup file are shown in one of the top two windows. The configurations currently being used are in the second window, the

ones not currently in use are in the top window. When a configuration is selected in the Used Configurations window, its widgets are shown in the Widgets window. When a widget is selected, its components are shown in the Components window. When a configuration or widget is activated, "(A)" is shown before its name.

Buttons in this window allow local control over interface objects. It is important to note that the Use and Release buttons make local changes NOT visible to any application communicating with EDVision via VIML. In other words, if a configuration is Used at the local GUI, applications will not become aware of it, and will not receive events from its widgets. Typically, local Use and Release commands are only used during testing and debugging. For example, it is often desirable to tune a widget's parameters without running the application that will use it. With the Configuration dialog, a configuration can be enabled in order to test the widget behavior. The system will behave normally, but no events will be sent to any application. In this mode the parameters can be tuned and saved to the setup file so that the next time an application does Use the configuration, the tuned parameter values will be in effect. (The Activate and Move to Surface buttons can be used to affect any Used configuration, whether it was Used by an application or locally. These changes will affect an application Using those configurations.)

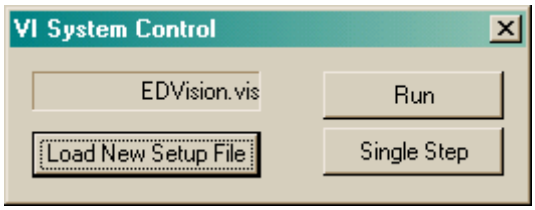
The Configuration dialog allows widgets and configurations to be added and removed. Any changes will be saved to the local setup file, and will be effective on any subsequent Use, by the application or locally. Widgets can also be forced to generate an event to the application that Used them.

The Surfaces dialog allows creation and calibration of surfaces onto which configurations can be mapped.



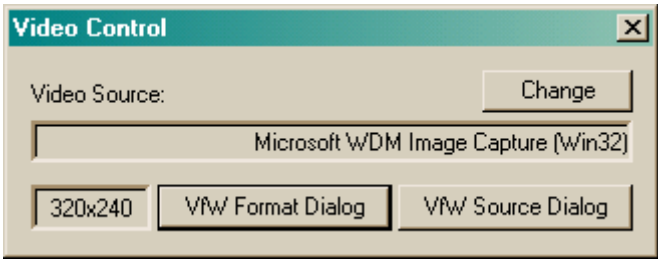
Selecting a surface and pressing Calibrate takes the user through a series of dialogs that set the user's hand size and orientation, as well as the corners of the surface within the image (described below). These values are automatically saved, then used when a configuration is mapped to the surface.

Finally, the VI System dialog is used to Start/Stop and Single Step the system, as well as to manage the setup file.



The default setup file (EDVision.vis in the same directory as EDVision.exe) is loaded automatically at startup. The setup file can be changed (for the current session only) using the Load New Setup File button.

The Video Control dialog is accessed by a right click on the video window:



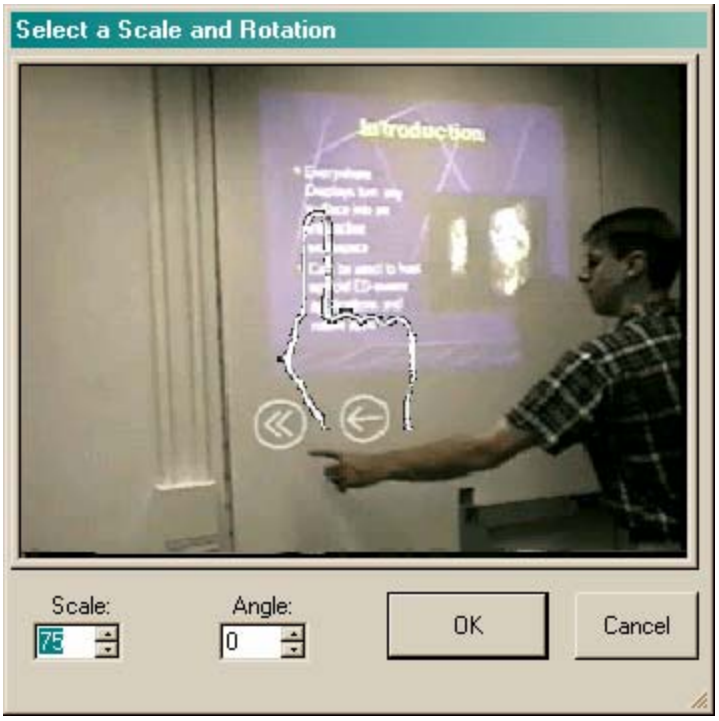
The dialog shows the video source (camera or file) and resolution. The Change button allows the user to select a different video source. The VfW Source and VfW Format buttons allow access to the control dialogs of Video For Windows sources. The details of these dialogs depend on the camera, but they typically allow control over many aspects of the video stream including resolution. The resolution of video *files* cannot currently be changed.

System Calibration

When a surface is created locally or via VIML, it defaults to cover the entire image. Most surfaces must then be defined to match some planar surface in the field of view by using the Calibrate button on the Surface dialog.

Calibration consists of two steps, first defining the expected size and orientation of the user's hand when they are interacting with this surface, second defining the corners of the bounding region where configurations will be placed in the image. The Calibration button on the Surface dialog brings up a series of dialog boxes that guide you through this procedure.

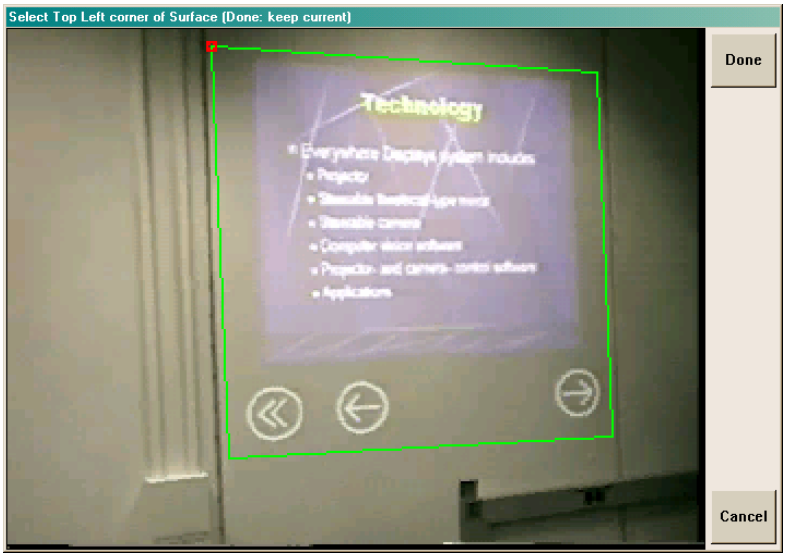
The Hand Scale and Rotation dialog allows calibration of the appearance of the user's hand.



The dialog shows a live stream from the video source. To calibrate, ask a user to place their hand in the image as if they were interacting with widgets in the scene, then use the spin controls to adjust the hand icon to the size and orientation of their hand. For best results have the user stand in the location where they will during operation, and place their hand in the center of the region where the widgets will appear. Only a single orientation / scale will be used for all widgets on a surface, so if the hand appearance changes for interactions with different widgets, use an average value, or set the values for good performance on the most important widgets.

Note that not all widgets use these hand parameters. Only the VIButton, MouseClickButton and VITrackArea widgets use this information. If none of these widget types are being used, these parameters can be left at their default values.

The Surface Corners dialog allows calibration of where corners of the surface bounding box.



This is actually a series of four dialogs, each one asking you to click on a different corner of the box (top left, top right, bottom left, bottom right). As the corners are defined, the edges of the bounding box are shown in green. The current location of the corner you are being asked to define is shown in red.

Note the order in which the corners must be defined! The title of each dialog indicates which corner is being asked for.

Note that the bounding box will generally appear as a quadrilateral, rather than a rectangle, unless the camera is directly over the surface.

Note that VIML can define widgets to lie outside the surface bounding box. This can only be done via VIML, not from the EDVision GUI.

Note that the corners of the bounding box may be outside the video image. This is supported during calibration by clicking the estimated location of the outside the dialog box. Because of a bug, the edges (green lines) extending to corners outside the dialog box are not shown, but bounding boxes with corners outside the video image will work properly.

Widgets

EDVision supports several different widgets, generally classified as Buttons or TrackAreas. Buttons detect a touch-like motion at some point and return a VIeventTouch in response (refer to the EDML language specification for more info on VIML events). TrackAreas generate a VIeventPositionChange whenever they detect an object moving within their boundaries. The different types of buttons and track areas respond to different types of activity in the image.

From the EDVision GUI any type of button or track area can be created. When VIML is being used to create a configuration, however, the value of the "other" field of the Use method controls the type of widget created, as described in the section of this document entitled **VIML "other" field documentation**.

The default widget types are suitable for most applications, but some interactions require different capabilities. Some widgets were built for very specific applications, and are included here for completeness. Some widgets were built to support various calibration operations. This section describes the different types of widgets currently supported, followed by a brief description of how widgets are implemented, and some notes on tuning the parameters of the most important widgets for best performance.

Widget Types

The following are the widget types supported as of EDVision v2.4.5. A subset of these types are supported in earlier versions.

VIOmniDirButton: Responds to any long thin object (finger, pen, etc) which moves toward the button, enters it, does not pass through it, then retracts the way it came. This is generally the most reliable type of button.

VILocalOmniDirButton: Like a VIOmniDirButton, but generates a local mouse click event at the screen coordinates of the center of the button, as well as generating an XML event.

VIButton: Locates fingertips using the hand size/orientation data in the VIsurface. Tracks the fingertip extended furthest from the user and responds when it moves in a touch-like motion that extends into the button. Can be grouped into "keyboards" where only one button can fire at a time. The user must be in a known location, but this can be useful when you need to ignore possible touch motions from other directions. This button requires care in surface calibration to get the hand orientation right, and can be less reliable than OmniDirButtons.

MouseClickButton: Tracks fingertips as a VIButton, and responds to any touch motion within its borders, returning the location (in config coordinates) of the touch. This is useful when an application must know where in an area a touch occurred. This button can be configured to generate Windows events on the local host (the system where EDVision is running), as well as VIML events. So for example, the camera could be aimed at the computer screen and a MouseClickButton placed over the entire image of the screen so that Windows mouse click events are generated

where touches occur, creating a vision-based touch screen (for this to work the config space coordinates of the widget must match the screen coordinates).

VITrackArea: Locates fingertips using the hand size/orientation data in the VIsurface. Tracks the fingertip within its borders extended furthest from the user and returns VIeventPositionChange indicating its position each video frame. The user must be in a known location. Sensitivity can be adjusted.

VIPolarTrackArea: Tracks motion in the vicinity of the button and returns VIeventPositionChange indicating the angle and distance from the center of the widget to that motion (x=angle in degrees, y =distance in config space units). This is very useful when an application must know from what direction a user is approaching a widget, and respond accordingly.

VIColorTransSliderV: Returns the y location of the first sharp color transition down from the top of the track area. Good for sliders where the user sticks their finger into a slider and moves it up and down to control a value. Works best when the track area is narrow horizontally. Contains logic to detect when the user drops their finger down and out of the widget, and return the top-most value reached.

VIPhysicalSliderV: Tracks the motion of a distinctively colored bar moving vertically within the widget, and returns the y location of the bar within the range of the track area. The color of the bar must be marked a short distance above the widget for calibration.

VIIgnoreMotion: Forces all motion in its boundary to be ignored by any other widget. Useful for blocking out known incidental motion in the image. (most widgets respond to movement within the image. If some location in the image is known to change regularly, and causes problems of recognition, this widget can be used to block it out).

BlinkDotFinder; Returns the location of the largest blinking dot in the widget. Useful for calibration operations.

VIQuadLoc: Shows the corners of the widget as a quadrilateral in EDVision's video window. Each time the user moves a corner with the mouse, it returns four VIeventPositionChange's to the application indicating the new corners. Useful for calibration operations.

MouseClickLoc: Shows the user a dialog box containing the video stream and returns the location where the user clicks their mouse. One call can generate any number of dialogs (see **VIML "other" field documentation**). Useful for calibration operations.

Widget Implementation

Widgets are implemented as trees of image processing "components", where each component performs some elementary operation and passes the result to its child components. The initial image processing operations are often in components which are shared between widgets, while later operations are generally in components specific to each widget. When a configuration is Used, the components making up its widgets are instantiated and formed into a tree structure. When the system is running, images from the video stream are passed down this tree, processed by each component in turn, till events are generated by the leaf components. For more details of this architecture, see reference [2].

Each component has a number of parameters that control various aspects of its operation, for example the sensitivity of a MotionFinder component to changes in the image. The parameters default to reasonable values, but they can be optimized for various situations. All changes to these parameters are done via the local GUI (see below). When a new configuration is defined, its widgets get the default parameters. When a pre-existing configuration is reused, any changes that have been made to its parameters are recalled.

Many widgets make use of motion in the video stream. In this domain, motion segmentation is often more reliable than background subtraction or similar techniques because projected imagery significantly alters the appearance of both background and foreground objects. Movement is detected by looking at differences between adjacent images in the video stream. The raw motion energy is filtered to remove the effects of vibration and noise, then passed to subsequent components.

Widgets do various types of analysis of the motion data in their vicinity. The "Polar…" or "OmniDir…" widgets look for focused beams of motion energy aimed toward them from any direction. Other widgets track fingertips based on their shape, and evaluate their paths using knowledge of the user's location to determine if the user has interacted with the widget. A VIButton looks for the out-pause-back movement indicative of a touch at a specific location. Sliders and track areas report the relative location of fingertip in one and two dimensions.

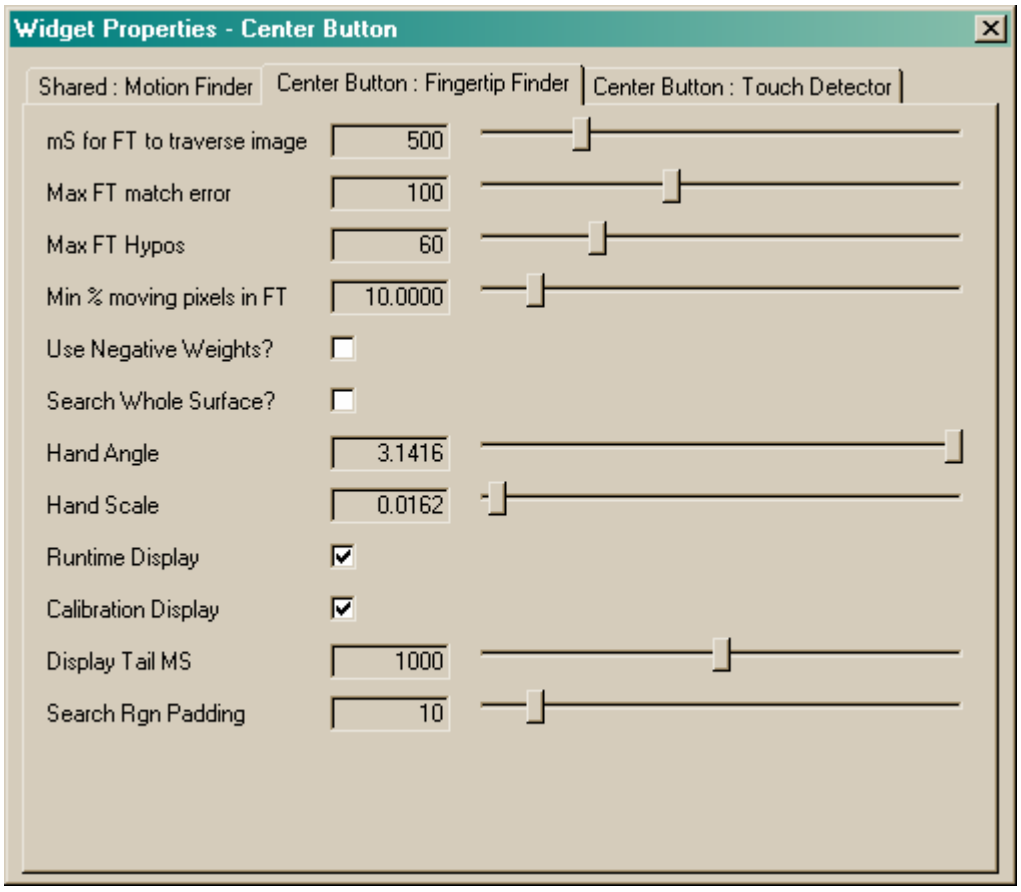
In addition to widgets that track motion, we have also implemented widgets that track colored or high contrast objects in a specific area. The "ColorTransition…" widgets track strong color transitions within their borders. The "PhysicalSlider…" widgets track colored blocks moving within a known region. This allows a different type of interaction where the movement or location of a physical object is salient, rather than the user's interaction with it.

Widgets are generally used to track hands or objects to create a user interface, however they are not limited to this application. In the abstract, a widget is an object EDVision creates on request which processes some or all of the video stream and can return events to the calling application as a result. Widgets have been created to mask off regions of the video stream from known distracters, and to support calibration operations. There is a well defined mechanism for creating new widgets, which is beyond the scope of this document. Contact the author for more information.

Tuning Widget Parameters

Widget Parameter Dialog

Right clicking on the video window, then selecting Widget Properties brings up a dialog box that allows control of a widget's parameters.



On this dialog there will be a tab for each component of a widget that has parameters that can be adjusted. Clicking on a widget in the video window changes the parameter control dialog to show just the components for that widget. Clicking on a part of the image with no widgets shows the components for all widgets in all Used configurations.

The dialog is modeless and changes are made to the system in real time, so parameters can be adjusted while the system is running, and the effect can be observed. **NOTE that parameter changes will not be saved unless the Save Params button on the Main Dialog is pressed.**

Note that this dialog will only be displayed when a configuration of at least one widget has been Used. Only widgets in a Used configuration will be shown.

At the top of the Widget Properties dialog box are tabs for each image processing "component". In each tab, the first part of the title is either the name of the widget using that component or the word "Shared". The second part of the title is the name

of the component. Referring to the figure above, to tune the Fingertip Finding component of the Center Button widget, you would select the tab labeled "Center Button : Fingertip Finder". Any changes made would affect only that widget. To tune the Motion Finding component of the Center Button, you would select the tab labeled "Shared : Motion Finder". Changes made here would affect all widgets that share that component.

The remainder of this section will describe how to tune the parameters for some of the more important image processing components.

Tuning Motion Finder

Most widgets start by finding motion in the video stream. The default values for motion finding are good for a wide range of lighting and camera conditions, but occasionally, especially in very bright or dark conditions, they need to be adjusted to get good performance.

The first step in adjusting the motion finder is to observe its operation. Start the system running, and in the Motion Finder tab of the Widget Parameter dialog turn on "Show Mask ?". The video window will then display an image in which white pixels indicate where the system has detected change. In this Motion Mask, moving objects must show up clear and strong, and where there is no movement the image must be black. The following describes how to adjust the parameters to get this result.

If there are pixels turning on where there is no motion, they must be eliminated. This noise may be caused either by camera or environmental vibration, or by overly sensitive motion finding (note that cameras tend to create more random noise in dark environments).

The first step to try is turning on the "Desensitize to Persistent" check box. This will desensitize the motion finder to pixels that continuously change, as would be caused by vibration. If the noise level is not reduced after a few seconds, turn this back off, as it can sometimes have undesirable effects.

If there is still image noise, the next step is to increase the sensitivity threshold. Move the slider "Difference Thresh" to the right slightly (increasing the number in the box). Move the slider only enough to eliminate the noise.

The next step is to ensure motion finding is sensitive enough to detect true motion. In the camera's field of view make a gesture you want the system to recognize (e.g. touching a button), and observe the result. During the movement, the system should show strong outline of the moving hand and arm, as shown in the figure.

The outline of a moving hand must extend all the way down to the fingertips. If the outline is sparse, broken up or does not include thin sections such as the fingers, Motion Finding needs to be made more sensitive. Do this by decreasing the Difference Threshold till it is.

Under extreme circumstances it may not be possible to reduce the threshold sufficiently to get good detection without causing random image noise as well. If this happens, you can leave the threshold at a setting that produces some noise, and try to reduce that noise by increasing the "Number of Erosion Steps" parameter.

The result should look like the figure above. In most lighting conditions it should be easy to obtain a good motion mask. With very poor lighting or with a bad camera, it may be impossible.

Default values:

Difference Threshold: .065
Number of Erosion steps: 1
Desensitize to Persistent Motion: off

Tuning PolarTouchButton

One of the most important gestures is touch detection, and the best touch detection widget is the VIOmniDirButton. The problems with touch detection will be either False Positives (touches detected when they should not be) or False Negatives (touches not being detected when they should). False Positives will most often occur during "near miss" gestures, such as waving your finger sideways through a widget, or when standing and moving between the camera and the surface where the widget is projected.

If you are getting to many **False Negatives**, first **increase** the value of the "Ltng Precursor Angle" parameter on the "Polar Touch Button 2" tab. If the value is increased above about 100, and you still get to many False Negative touches, **decrease** the value of the "Dwell Time (MS)" parameter on the "Polar Touch Button" tab. Only change dwell if you are not getting sufficient effect by increasing the Precursor Angle. In extreme cases, Dwell can be reduced to 0, which makes the buttons very sensitive, even to non-touch movements.

If you are getting to many **False Positives**, first **decrease** the value of "Ltng Precursor Angle" on the "Polar Touch Button 2" tab. Do not reduce this value too much, or the button may become completely unresponsive. If this value must be reduced below about 50, and there are still too many False Positives, then **increase** the "Dwell Time (MS)" parameter on the "Polar Touch Button" tab. If Dwell is increased too much, again the button will become unresponsive.

Default values:

Dwell: 300
Ltng Precursor Angle: 60

VIML “other” field documentation

A field of the form “other= …” appears in several VIML methods. For details on syntax, see the EDML language specification. The other field provides for extensions to VIML between architecture updates. This section documents the current usage of the other field (as of EDVision v2.4).

The most important use of the other field is to control what type of widget is actually created by a Use method. There are several types of buttons and track areas available for use in user interfaces, as documented in the Widgets section of this document. Currently VIML is only aware of two types of widgets, VIbuttons and VItrackAreas. By using the “other” field, an application can determine exactly what type of widget is created. For example, the following VIML command creates a configuration named Example with one VILocalOmniDirButton.

```
<viml application="Documentation">
        <use >
                <VIconfiguration name="Example">
                        <VIbutton name="anODB" other=”LocalOmniDirButton” />
                </VIconfiguration>
        </use>
</viml>
```

Note that if the other field is omitted, a VIOmniDirButton or a VITrackArea is created (depending on VIML widget type).

As well as the button type, the other field can contain the string “group=xxx”. If this is present in the other field it indicates that the button is part of a group of buttons, which may behave differently than if the buttons are not grouped. For example the following VIML command creates a configuration with two track areas that share fingertip finding, meaning that only one of them (the one containing the best fingertip hypothesis) can generate an event any video frame:

```
<viml application="Documentation">
        <use >
                <VIconfiguration name="Example2">
                        <VIbutton name="B1" other=”TrackArea   group=foo” />
                        <VIbutton name="B2" other=”group=bar TrackArea” />
                </VIconfiguration>
        </use>
</viml>
```

Note that the widget type and “group=” can appear in any order.
Note that the group name is ignored, any button with “group=” as part of the other field will be part of the same group
Note that only some widget types support “group=”

When the VIML Use method is used to define a configuration, the VIbutton and VItrackArea objects can contain an "other=" field as in the examples above. For this application, the following other field values are supported:

VIbutton

OmniDirButton	Crates a VIOmniDirButton (default)
LocalOmniDirButton	Creates a VILocalOmniDirButton
TouchMotionButton	Creates a VIButton
MouseClickButton	Creates a MouseClickButton
group=xyz	All VIButtons with this designator share the FingertipFinder component Ignored for other button types xyz can be anything, and is ignored

VItrackArea

TrackArea	Creates a VITrackArea (default)
PolarTrackArea	Creates VIPolarTrackArea
ColorTransitionSliderV	Creates VIColorTransSliderV
PhysicalSliderV	Creates VIPhysicalSliderV
IgnoreMotion	Creates VIIgnoreMotion
BlinkDot	Creates BlinkDotFinder
QuadSelect	Creates VIQuadLoc
MouseClickLocN	Creates MouseClickLoc where N is the number of points for the vision system to ask the user for
group=xyz	All VITrackAreas with this designator share the FingertipFinder component Ignored for other widget types xyz can be anything, and is ignored

When the VIML Set command is used to control various parameters of widgets, the widget object can have the following values in the other field:

Button

event	Generate a touch event
width=*w*;height=*h*	Set a rectangular size for the button (*w* x *h*) Not supported by type VIOmniDirButton Must be the first value in the other field

TrackArea

event	Generates a track event at x=100, y=20
MouseClickLoc*N*	If widget is a MouseClickLoc, Sets the number of points to ask the user for to *N* Ignored for other widget types

Application Design and Best Practice Tips

Under Construction

To get the best performance out of EDVision, the following guidelines should be followed:

- It is generally good practice to zoom the camera in so the user's hands are large in the image. This helps avoid false positives from spurious noise, and false negatives when small fingers are ignored as noise.
- The camera / projector should be mounted solidly, so that there is no vibration in the camera image. EDVision works by detecting and analyzing movement. As such it is robust to variations in lighting conditions, skin color, and the like, but it is sensitive to vibration in the image. If the whole image vibrates, or the projection vibrates within an otherwise stable image, it will cause performance problems.
- It sometimes helps to defocus the camera image slightly when you calibrate the camera surface, so sharp edges appear slightly blurry. Sharp edges are aliased in digital images, causing what is referred to as "pixel crawl". This is interpreted by the system as motion, and can cause performance problems.

Miscellaneous notes and gotchas:

EDVision stores config information locally. That is what enables you to set parameters (tune the system) from the EDVision GUI, and have them remembered each time the system is used (low level image processing parameters are un-architected, invisible to the application).

When an application specifies a config with the same name as one stored previously, that config is reused as much as possible (i.e. if new widgets are specified, they are added. Widgets are not deleted). If a new config (or widget) is specified, it will be dynamically created with default parameters that should give reasonable performance.

It is generally good practice for an application to specify all the configs it will use each time it starts (at least), so that there are no problems with undefined or incorrectly defined configs.

There are some issues with this architecture. One is that an application should not Activate an entire config (though it is architected), as that will activate all the widgets within it. If two applications use the same config name, or a config changes during development, it may have more widgets in it than an application is aware of. If the config is activated as a whole, widgets unknown to the application may start running, stealing system resources. There it is good practice for an application to activate each widget it needs individually.

When upgrading to a new version of EDVision, the release generally includes a new setup file (EDVision.vis). The new setup file will often have changes required for the new code to operate properly. Unfortunately if you just replace the existing setup file with the new one, you will loose all calibration information (surfaces, etc). It is possible to edit the setup file so that you get the new function, but retain your existing calibration. To do that you must replace the bottom half of the supplied setup file with the bottom half of your existing setup file. Starting with version 2.0, there is a comment in the setup file where this cut should take (the setup file is plain text, and can be edited with Notepad).

Basic Troubleshooting

Under Construction

- **Testing EDVision**:
  To test EDVision, you can create and run a simple configuration from the local user interface as follows: With the Configuration dialog, either Use an existing configuration (several should be defined in the setup file in the release package) or create a New configuration and Add Widgets to it (an OmniDirButton is probably the best widget to use). Then Activate the configuration and use the Move to Surface button to put it on a calibrated surface. Finally from VI System dialog select Run to start the system operating. At this point you should see the system track the hand when it is near or within buttons, and hear a beep from the vision system computer whenever you touch a button (make sure sound is turned up and not muted!)

  If this works, the vision system itself is working properly, and a correctly written application should have no trouble using it.
- For VfW cameras the video format must be set to RGB24. This can be set using the Video Control Dialog / Format Dialog. Some cameras and capture cards default to other formats when they are installed. This can cause a variety of odd behavior from the system, from error messages to strange recognition results.
- Remember that the orientation of the hand during an interaction (e.g. a button touch) is important for VIButtons and VITrackAreas. The further the user stands from the expected position, the more performance of the widget degrades. If the user stands on the opposite side of a widget than how the surface has been calibrated, they will usually not work at all.
- The out/back motion that triggers a VIButton is assumed to occur along the hand orientation angle set during surface calibration. While this assumption is generally valid, it fails for some camera geometries, particularly when the camera is at low angles to the surface (far from perpendicular) or when the user naturally swings their hand in an arc to touch a button, rather than reaching out to touch it. In these circumstances

the surface hand calibration may need to be adjusted away from the actual orientation of the user's hand in order to get good performance.

Known Bugs

As of 6/21/2004:

- Open Applications are not shown in the ED System dialog.
- Widgets of the same name and type in different configurations can conflict.
- Deleting a widget from a configuration occasionally causes a crash.
- Leaving the Widget Properties dialog open while configurations are Used or Released can cause a crash.

References

For addition detail on the system theory and design of EDVision see:

[1] Kjeldsen, R., et al. Interacting with Steerable Projected Displays. in Proceedings of the 5th International Conference on Automatic Face and Gesture Recognition. 2002. Washington, DC (ICAFGR '02)

[2] Kjeldsen, R., Levas, Pinhanez, C., Dynamically Reconfigurable Vision-based User Interfaces, Intl Conf on Vision Systems (ICVS '03)