VisionLab 7.5

Quick Start

www.openwire.org
www.mitov.com

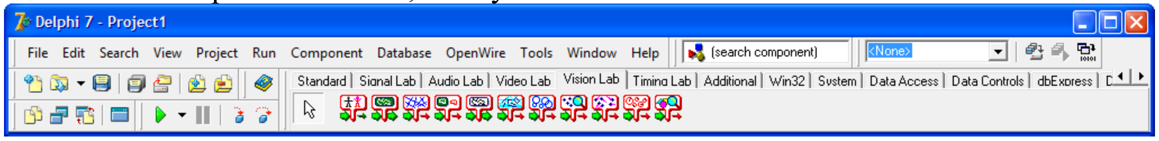
Index

Installation

VisionLab comes with an installation program. Just start the installation by double-clicking on the Setup.exe file and follow the installation instructions.

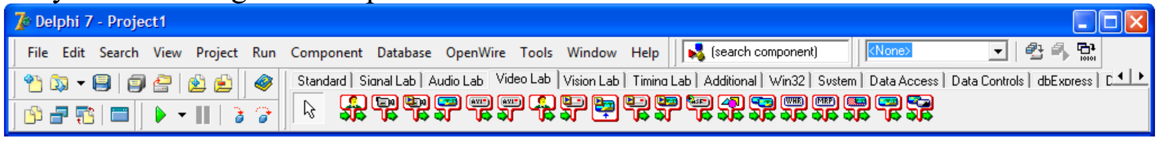
Where is VisionLab?

After the installation, start your Delphi or C++ Builder.
Scroll the "Component Palette", until you see the last four tabs:

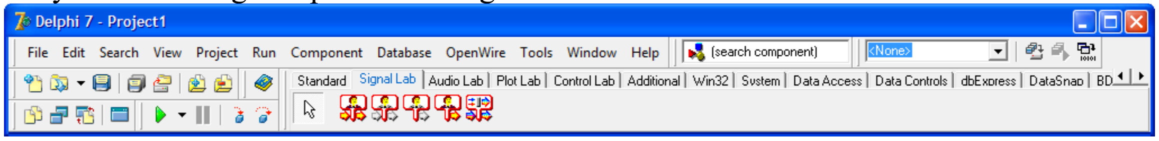

Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Vision Lab | Timing Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | D

If the installation was successful, they should be named "Video Lab", "Signal Lab", "Audio Lab", and "Vision Lab".
On the SignalLab palette you will have only a subset of the SignalLab components necessary for processing histogram data. SignalLab is a separated product, and will not be shipped as full with VisionLab.
On the AudioLab palette you will have only a subset of the AudioLab components necessary for basic processing audio data. AudioLab is a separated product, and will not be shipped as full with VisionLab.
On the VideoLab palette you will have only a subset of the AudioLab components necessary for video capturing, and visualization.
VideoLab is a separated product, and will not be shipped as full with VisionLab.

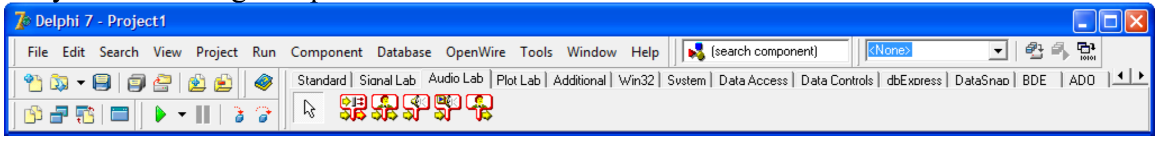
Only the following two components of VideoLab will be available:


Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Vision Lab | Timing Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | D

Only the following components of SignalLab will be available:


Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Plot Lab | Control Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap | BD

Only the following components of AudioLab will be available:


Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Plot Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap | BDE | ADO

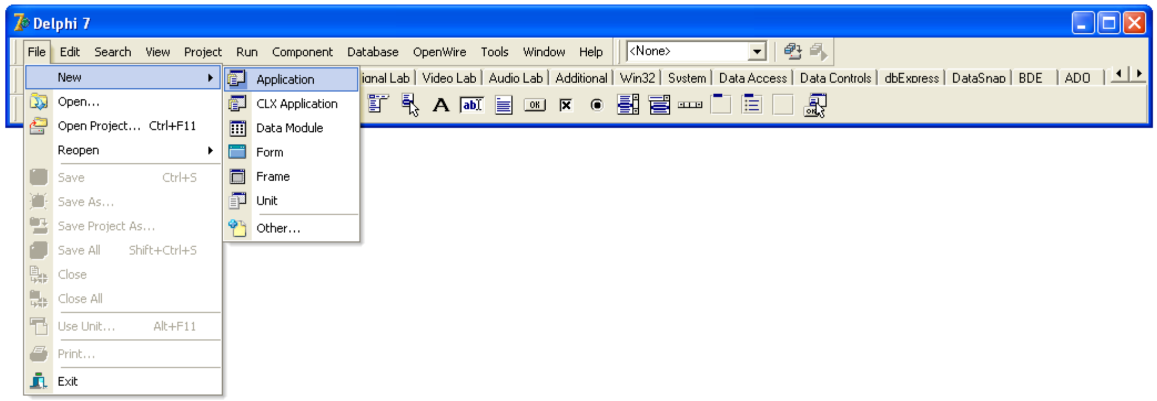
Why some of the examples don't work?

Video lab is a unique library that supports both the Win32 API's AVIFile (VFW) functions (ACM) and DirectShow. You as a developer have the ultimate choice to use either the Win32 API or DirectShow components or both at the same time.

The advantage of the Win32 API components is that hey will work on any Windows 95 and up system out of the box, however they are much less capable than the DirectShow components, and should be avoided if not necessary.
The advantage of the DirectShow components is that they will use the latest and greatest capability of DirectShow, the latest video camera devices, and TV Tuners, but they require DirectShow 9.0 or higher to be installed in order to work.
If you don't have DirectX 9.0 or higher installed on your system, you will not be able to use see the DirectShow examples working.

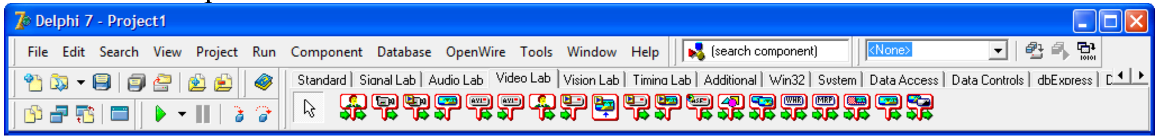
Creating a simple video capture application using DirectShow

WARNING: In order to run the application in this example you must have DirectX 9.0 or higher installed!
From the Delphi/C++Builder menu select | File | New | Application |.



An empty form will appear on the screen.
From the Object Inspector change the form Caption to "DirectShow Video Capture Demo".

From the "Component Palette" select the "Video Lab" tab:

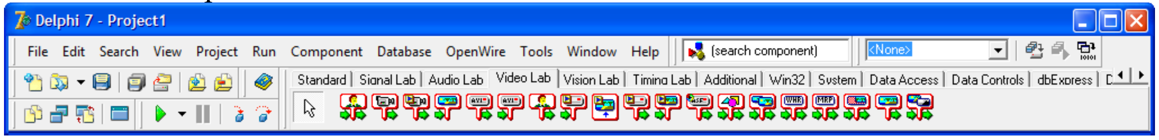


From the tab select and drop on the form the following components:

One - TVLDSCapture

Two - TVLImageDisplay

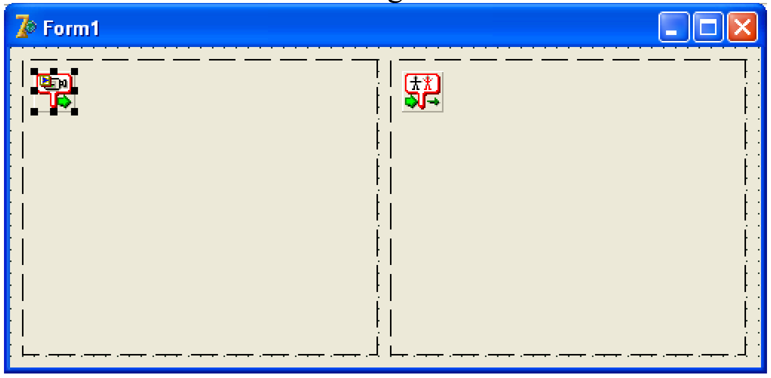
From the "Component Palette" select the "Vision Lab" tab:



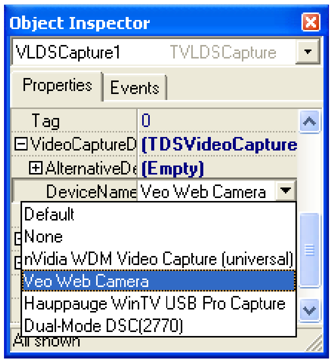
From the tab select and drop on the form the following component:

One - TVLMotionDetect

On the form select the VLDSCapture1 component.
Your form will look something like this:



In the Object Inspector expand the VideoCaptureDevice property and set the DeviceName to a video source(web camera, etc.) available on your system:


Object Inspector

VLDSCapture1 TVLDSCapture

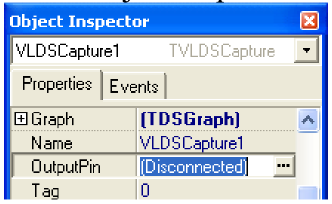
Properties | Events

Tag	0
⊟VideoCaptureD	(TDSVideoCapture
⊞AlternativeDe	(Empty)
DeviceName	Veo Web Camera

Default
None
nVidia WDM Video Capture (universal)
Veo Web Camera
Hauppauge WinTV USB Pro Capture
Dual-Mode DSC(2770)

All shown

In the Object inspector select the OutputPin property and click the button.

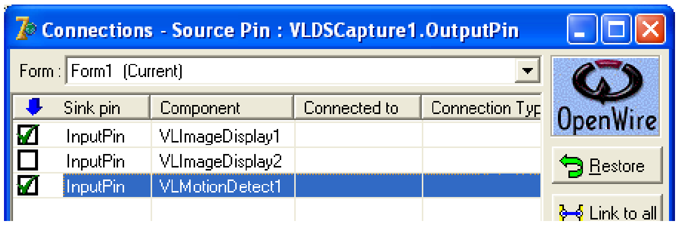

Object Inspector

VLDSCapture1 TVLDSCapture

Properties | Events

⊞Graph	(TDSGraph)
Name	VLDSCapture1
OutputPin	(Disconnected)
Tag	0

You should see the Pin Editor:

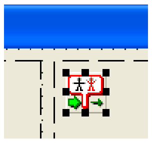

Connections - Source Pin : VLDSCapture1.OutputPin

Form : Form1 (Current)

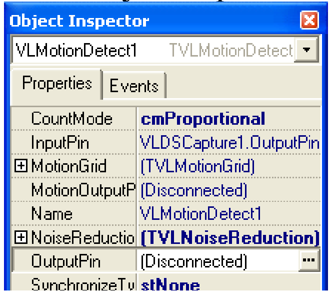
	Sink pin	Component	Connected to	Connection Typ
☑	InputPin	VLImageDisplay1		
☐	InputPin	VLImageDisplay2		
☑	InputPin	VLMotionDetect1		

Click on the check boxes to make it the connections as shown on the picture, and then click OK.

On the form select the VLMotionDetect1 component:



In the Object inspector select the OutputPin property and click the ⋯ button.

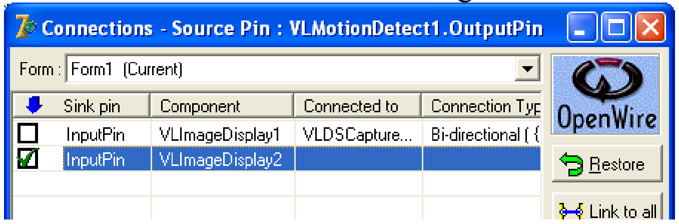


VLMotionDetect1 TVLMotionDetect

Properties | Events

CountMode	cmProportional
InputPin	VLDSCapture1.OutputPin
⊞ MotionGrid	(TVLMotionGrid)
MotionOutputP	(Disconnected)
Name	VLMotionDetect1
⊞ NoiseReductio	(TVLNoiseReduction)
OutputPin	(Disconnected)
SynchronizeTy	stNone

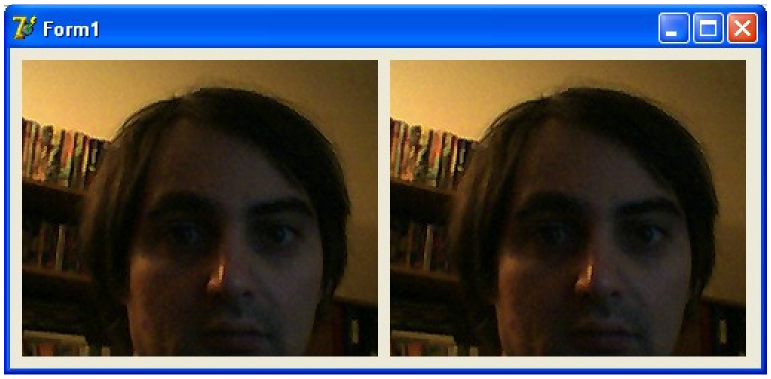
In the Pin Editor make the following selection and click OK:



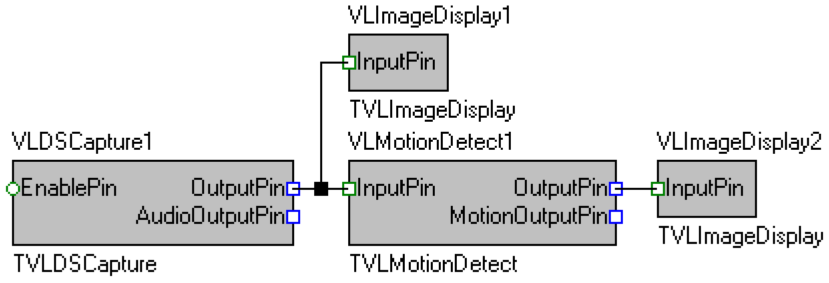
Form : Form1 (Current)

	Sink pin	Component	Connected to	Connection Typ
☐	InputPin	VLImageDisplay1	VLDSCapture...	Bi-directional ({
☑	InputPin	VLImageDisplay2		

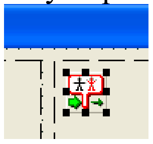
Compile and run the application. You should see the two displays like shown on the picture. The first display will show the input of the camera, and the second will show the last frame where there was a movement detected:



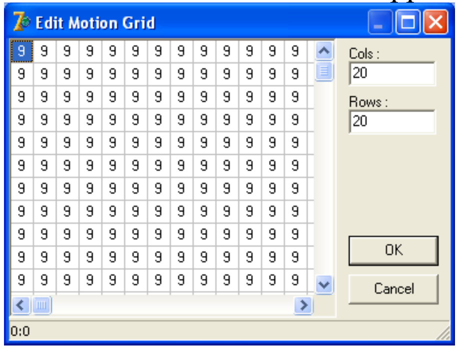
Congratulations! You have just created your first Motion Detection application.
Here are the OpenWire connections in this application:



This example is probably too sensitive and will detect even the slightest movement. You may want to reduce the sensitivity in some areas of the image, and detect movements only in particular zone. To do so, double-click on the VLMotionDetect1 component:



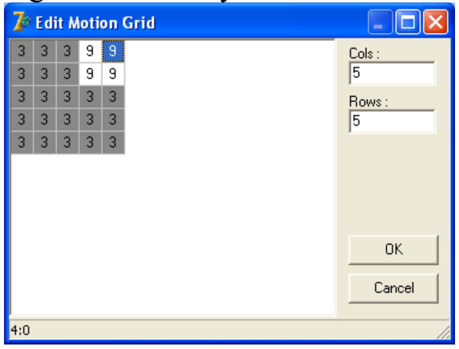
The Motion Grid Editor will appear:



9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	9	9	9	9	9	9

Cols : 20
Rows : 20
0:0

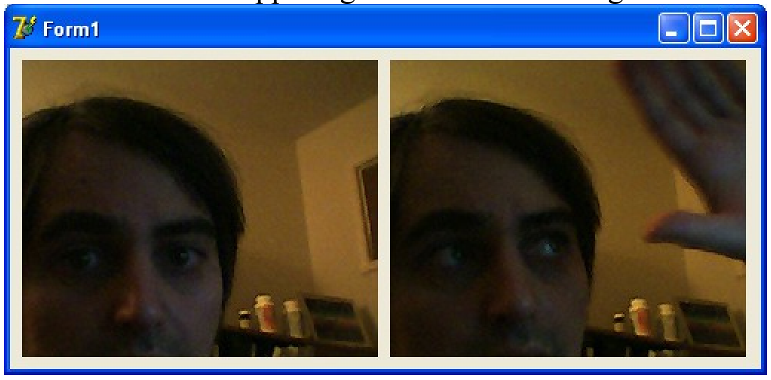
You can change the size of the grid, and you can set the sensitivity in each cell by pressing the keys 0-9 when the cell is selected. 0 means no sensitivity, and 9 means the highest sensitivity:



3	3	3	9	9
3	3	3	9	9
3	3	3	3	3
3	3	3	3	3
3	3	3	3	3

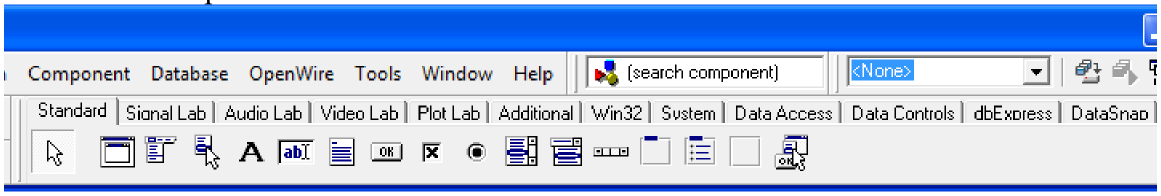
Cols : 5
Rows : 5
4:0

Click OK, compile and run the application, you will see that now it will be sensitive to movements in the upper right corner of the image:



The next step is to get the information about where in the image the motion event occurred, as well as to do something like send alert, or take other type of action.

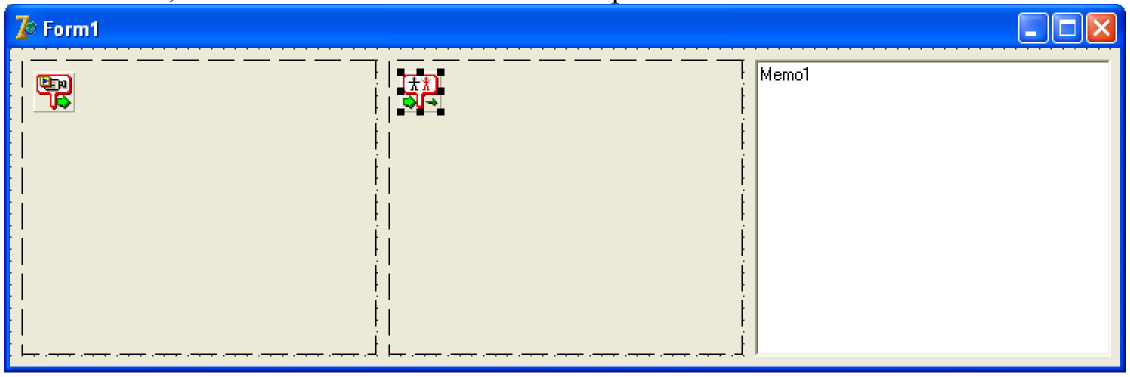
From the “Component Palette” select the “Standard” tab:


Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Plot Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap

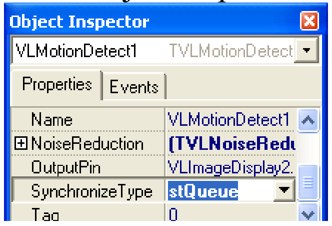
select and drop on the form a TMemo.

- TMemo

On the form, select the VLMotionDetect1 component:

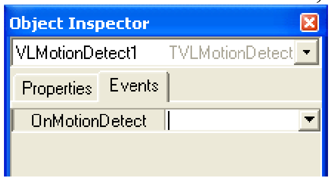


In the Object Inspector set the SynchronizeType tp stQueue:



Name	VLMotionDetect1
⊞NoiseReduction	(TVLNoiseRed
OutputPin	VLImageDisplay2.
SynchronizeType	stQueue
Tag	0

Switch to the Events tab, and double-click on the OnMotionDetect event:



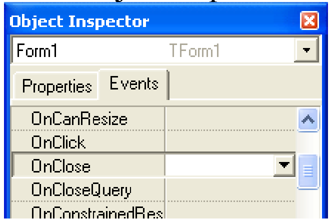
If you are using Delphi, In the event handler add the following code:

```
procedure TForm1.VLMotionDetect1MotionDetect(Sender: TObject;
  Motions: TVLMotions);
begin
  Memo1.Lines.Add( 'Movement at ' + IntToStr( Motions.MaxCell.X ) +
',' + IntToStr( Motions.MaxCell.Y ) );
end;
```

If you are using C++ Builder, In the event handler add the following code:

```
void __fastcall TForm1::VLMotionDetect1MotionDetect(TObject *Sender,
      TVLMotions *Motions)
{
  Memo1->Lines->Add( AnsiString( "Movement at " ) + Motions->MaxCel-
l.x + "," + Motions->MaxCell.y );
}
```

Select the form.
In the object inspector double-click on the OnClose event:



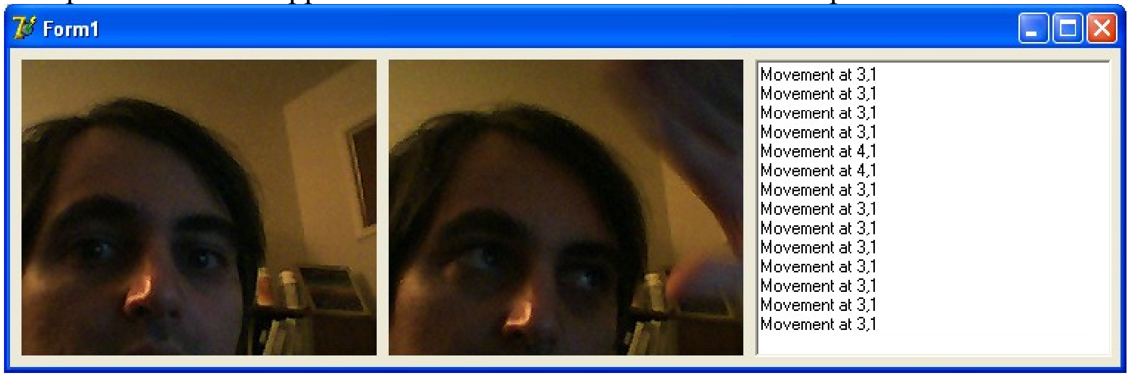
If you are using Delphi, In the event handler add the following code:

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  VLDSCapture1.Stop();
end;
```

If you are using C++ Builder, In the event handler add the following code:

```
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Ac-
tion)
{
  VLDSCapture1->Stop();
}
```

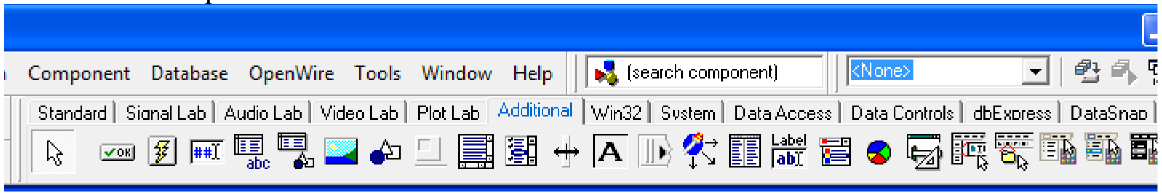
Compile and run the application. You will see the movements reported in the Memo:



```
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 4,1
Movement at 4,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
```

The last step is to obtain the full motion information for each cell.

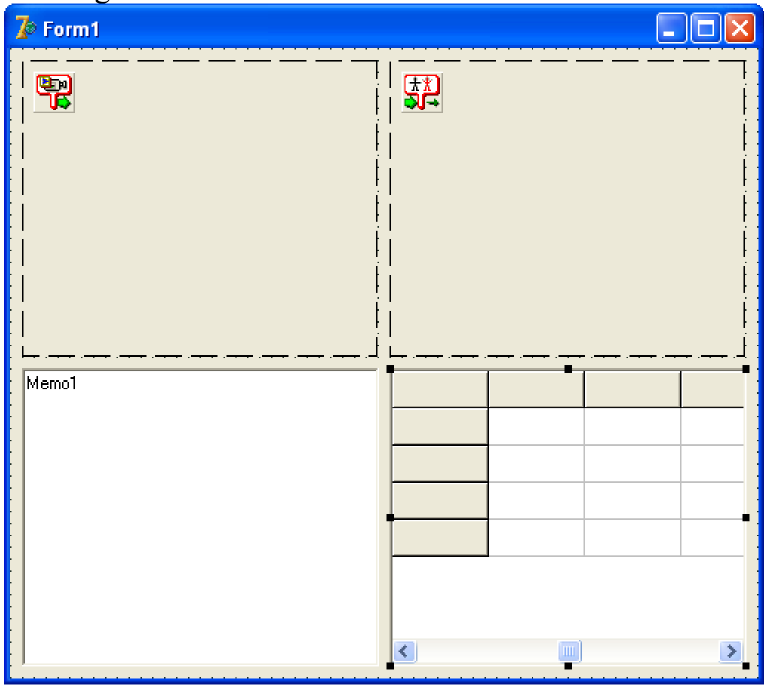
From the "Component Palette" select the "Additional" tab:



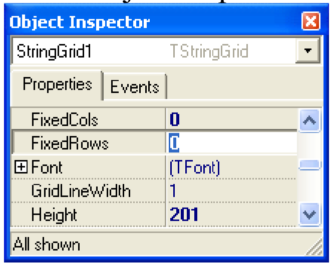
select and drop on the form a TStringGrid component:

- TStringGrid.

Rearange the form like this one:



In the Object Inspector set the FixedCols and FixedRows of the StringGrid1 to 0:



Object Inspector

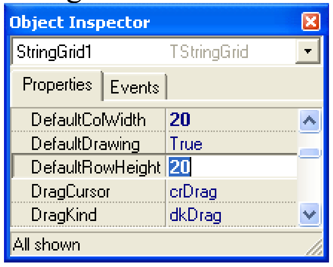
StringGrid1 TStringGrid

Properties | Events

FixedCols	0
FixedRows	0
⊞Font	(TFont)
GridLineWidth	1
Height	201

All shown

In the Object Inspector set the DefaultColWidth and DefaultRowHeight of the StringGrid1 to 20:



Object Inspector

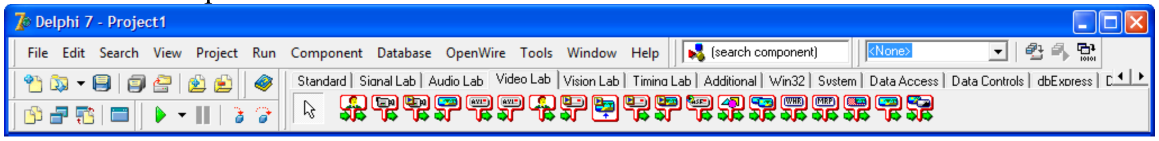
StringGrid1 TStringGrid

Properties | Events

DefaultColWidth	20
DefaultDrawing	True
DefaultRowHeight	20
DragCursor	crDrag
DragKind	dkDrag

All shown

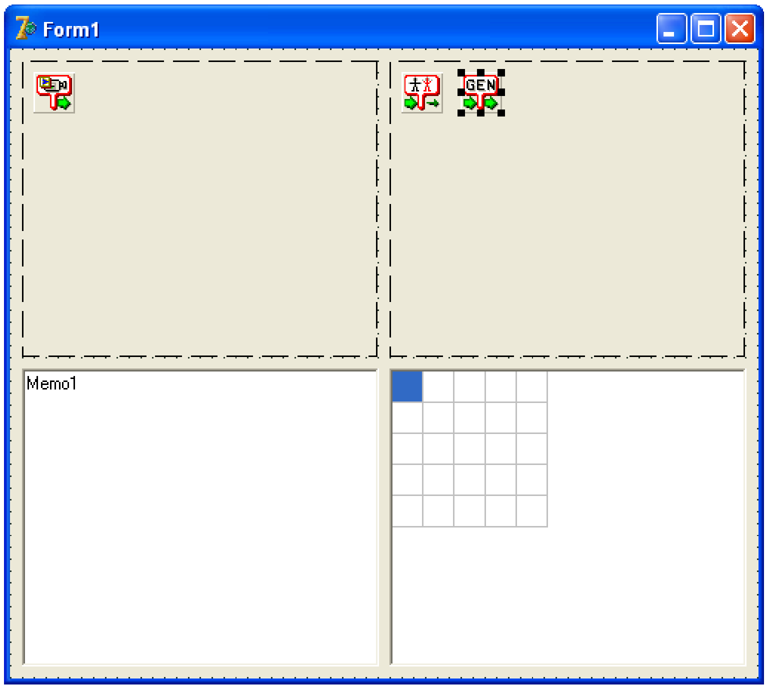
From the “Component Palette” select the “Video Lab” tab:


Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Vision Lab | Timing Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | D

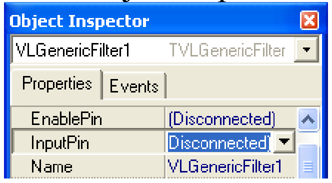
From the tab select and drop on the form the following component:

GEN - TVLGenericFilter

The form now will look like this:



In the Object Inspector select the OutputPin property and click the ··· button.



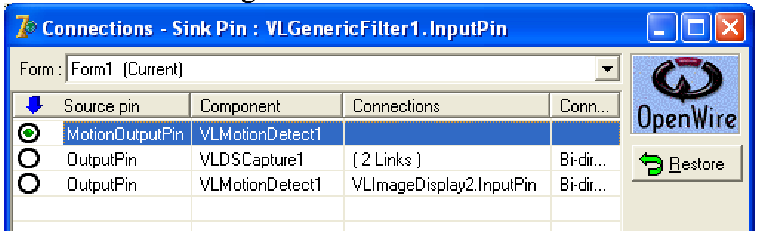
Object Inspector

VLGenericFilter1 TVLGenericFilter

Properties | Events

EnablePin	(Disconnected)
InputPin	Disconnected
Name	VLGenericFilter1

Make the following selection in the Pin Editor and click OK:

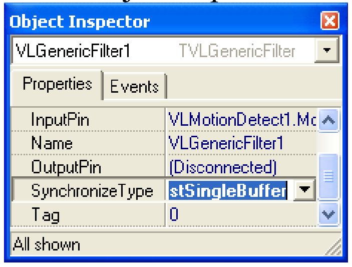


Connections - Sink Pin : VLGenericFilter1.InputPin

Form : Form1 (Current)

	Source pin	Component	Connections	Conn...
●	MotionOutputPin	VLMotionDetect1		
○	OutputPin	VLDSCapture1	(2 Links)	Bi-dir...
○	OutputPin	VLMotionDetect1	VLImageDisplay2.InputPin	Bi-dir...

In the ObjectInspector set the SynchronizeType to stSingleBuffer:



Object Inspector

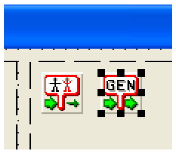
VLGenericFilter1 TVLGenericFilter

Properties | Events

InputPin	VLMotionDetect1.Mc
Name	VLGenericFilter1
OutputPin	(Disconnected)
SynchronizeType	stSingleBuffer
Tag	0

All shown

Double-click on the LVGenericFilter1 on the form:



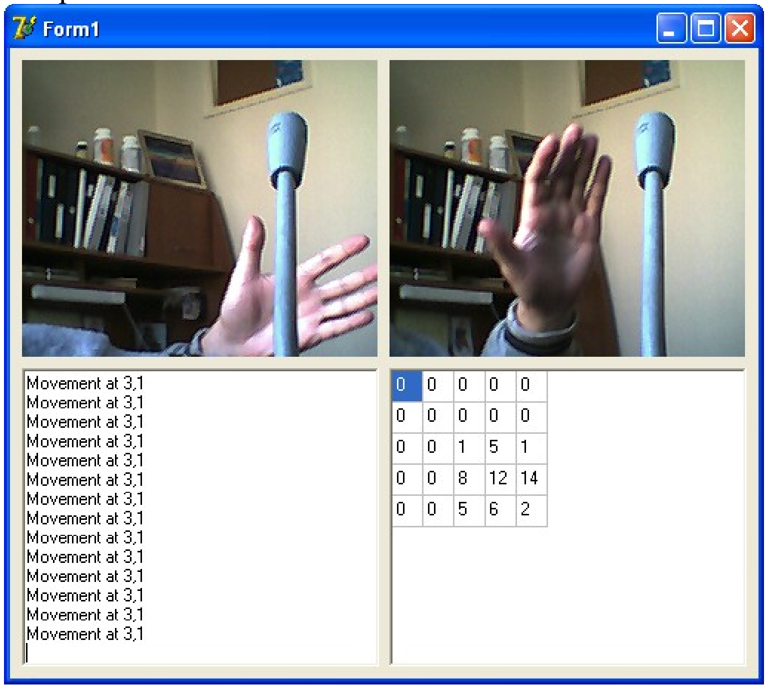
If you are using Delphi, In the event handler add the following code:

```
procedure TForm1.VLGenericFilter1ProcessData(Sender: TObject;
  InBuffer: IVLImageBuffer; var OutBuffer: IVLImageBuffer;
  var SendOutputData: Boolean);
var
  X, Y : Integer;

begin
  StringGrid1.ColCount := VLMotionDetect1.MotionGrid.Cols;
  StringGrid1.RowCount := VLMotionDetect1.MotionGrid.Rows;

  for X := 0 to StringGrid1.ColCount - 1 do
    for Y := 0 to StringGrid1.RowCount - 1 do
      StringGrid1.Cells[ X, Y ] :=
IntToStr( VLMotionDetect1.Items[ X, Y ] );

end;
```

If you are using C++ Builder, In the event handler add the following code:

```
void __fastcall TForm1::VLGenericFilter1ProcessData(TObject *Sender,
      TVLCVideoBuffer InBuffer, TVLCVideoBuffer &OutBuffer,
      bool &SendOutputData)
{
  StringGrid1->ColCount = VLMotionDetect1->MotionGrid->Cols;
  StringGrid1->RowCount = VLMotionDetect1->MotionGrid->Rows;

  for( int x = 0; x < StringGrid1->ColCount; x ++ )
    for( int y = 0; y < StringGrid1->RowCount; y ++ )
      StringGrid1->Cells[ x ][ y ] = VLMotionDetect1->Items[ x ]
[ y ];

}
```

Compile and run the application. You will see the movements reported in the StringGrid component:



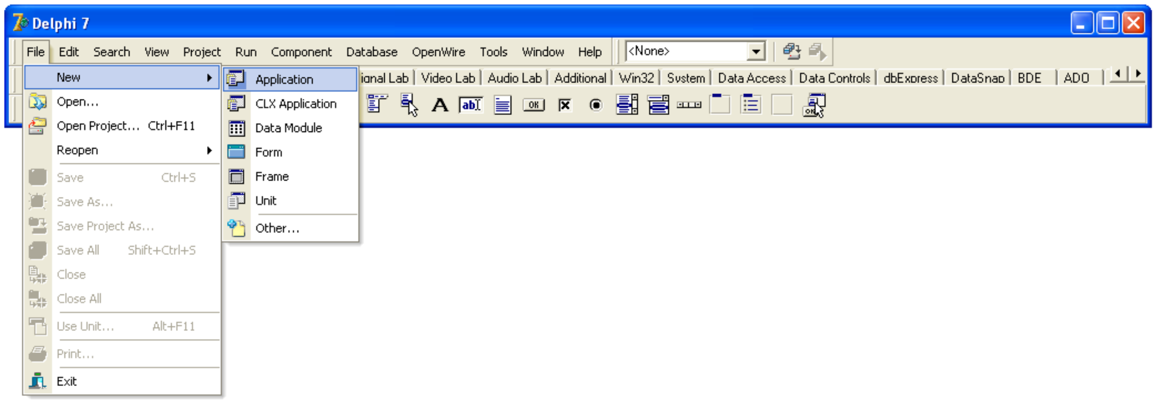
```
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
Movement at 3,1
```

```
0  0  0  0   0
0  0  0  0   0
0  0  1  5   1
0  0  8  12  14
0  0  5  6   2
```

Congratulations! You have just learned how to create complex motion detection applications with just few lines of code.

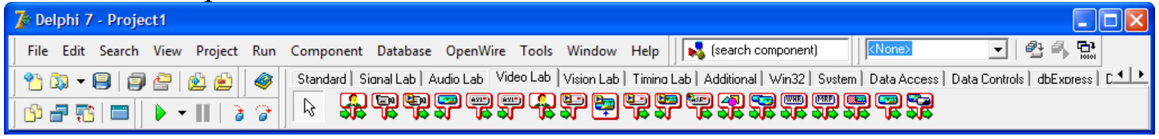
Creating a simple contour detection application

From the Delphi/C++Builder menu select | File | New | Application |.



An empty form will appear on the screen.

From the “Component Palette” select the “Video Lab” tab:

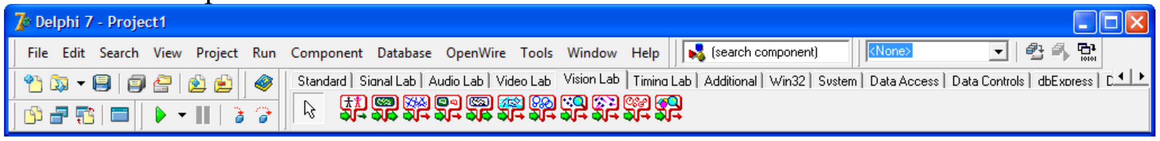

Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Vision Lab | Timing Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress

From the tab select and drop on the form the following components:

One - TVLAVIPlayer

Two - TVLImageDisplay

From the "Component Palette" select the "Vision Lab" tab:

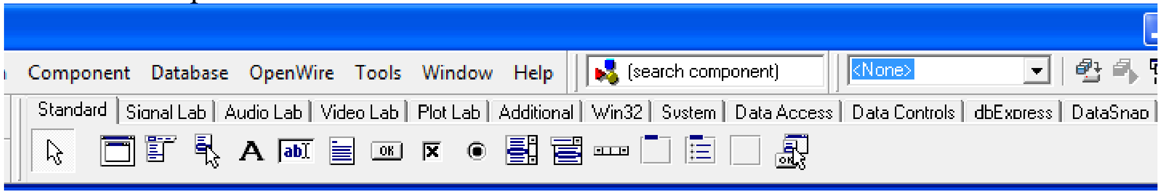

Delphi 7 - Project1
File Edit Search View Project Run Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Vision Lab | Timing Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress

From the tab select and drop on the form the following component:

One - TVLCanny

One - TVLFindContour

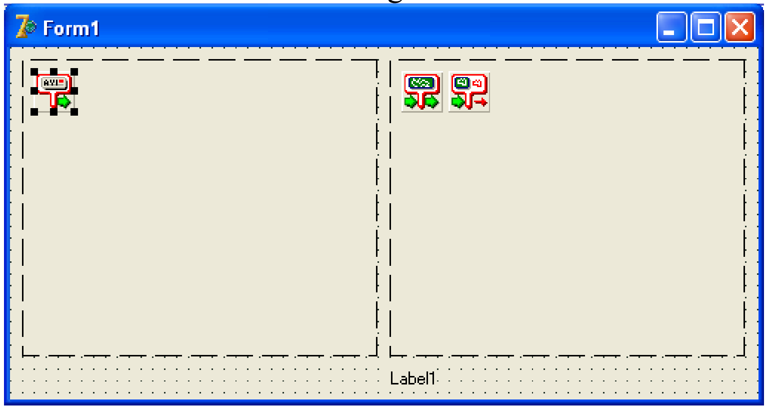
From the Component Palette select the "Standard" tab:


Component Database OpenWire Tools Window Help
(search component)
<None>
Standard | Signal Lab | Audio Lab | Video Lab | Plot Lab | Additional | Win32 | System | Data Access | Data Controls | dbExpress | DataSnap

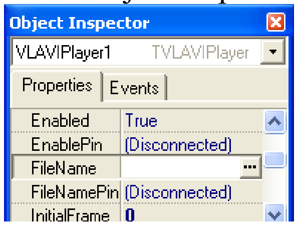
From the tab select and drop on the form the following component:

One - TLabel

On the form select the VLAVIPlayer1 component.
Your form will look something like this:



In the Object Inspector select the FileName property and click the ... button.

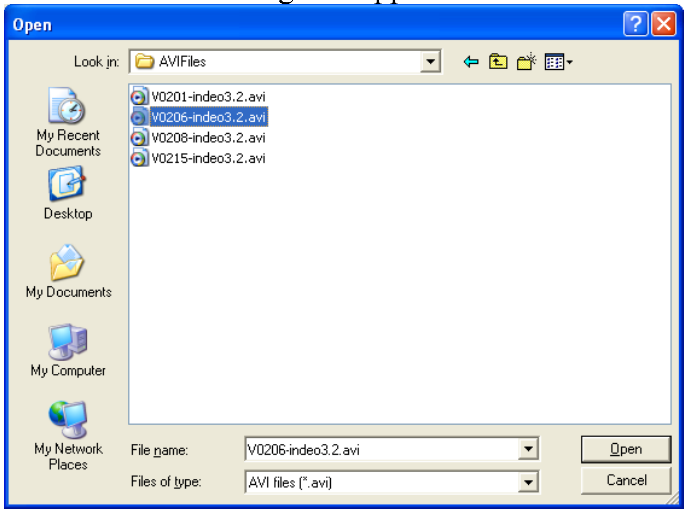

Object Inspector

VLAVIPlayer1 TVLAVIPlayer

Properties | Events

Enabled	True
EnablePin	(Disconnected)
FileName	...
FileNamePin	(Disconnected)
InitialFrame	0

A File selection dialog will appear:


Open

Look in: AVIFiles

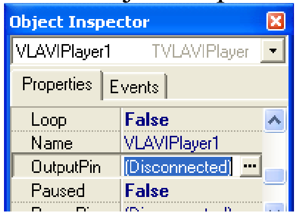
V0201-indeo3.2.avi
V0206-indeo3.2.avi
V0208-indeo3.2.avi
V0215-indeo3.2.avi

File name: V0206-indeo3.2.avi

Files of type: AVI files (*.avi)

Select a file to play and click “Open”.

In the Object inspector select the OutputPin property and click the ... button.

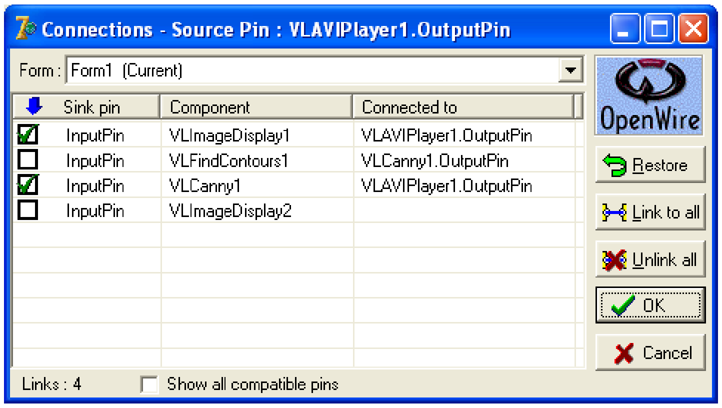

Object Inspector

VLAVIPlayer1 TVLAVIPlayer

Properties | Events

Loop	False
Name	VLAVIPlayer1
OutputPin	(Disconnected) ...
Paused	False

You should see the Pin Editor:


Connections - Source Pin : VLAVIPlayer1.OutputPin

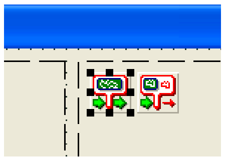
Form : Form1 (Current)

	Sink pin	Component	Connected to
☑	InputPin	VLImageDisplay1	VLAVIPlayer1.OutputPin
☐	InputPin	VLFindContours1	VLCanny1.OutputPin
☑	InputPin	VLCanny1	VLAVIPlayer1.OutputPin
☐	InputPin	VLImageDisplay2	

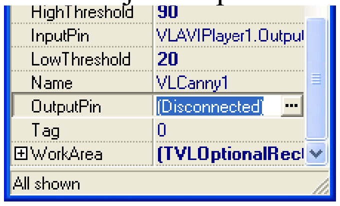
Links : 4 ☐ Show all compatible pins

Click on the check boxes to make it look as in the picture, and then click OK.

Select the VLCanny1 component on the form:



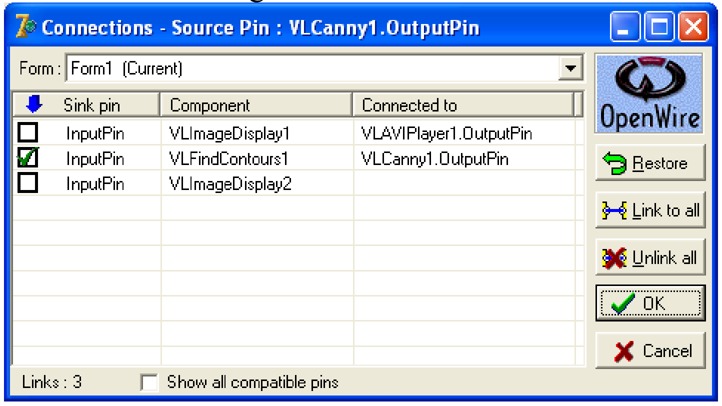
In the Object inspector select the OutputPin property and click the ... button.



HighThreshold	90
InputPin	VLAVIPlayer1.Output
LowThreshold	20
Name	VLCanny1
OutputPin	(Disconnected)
Tag	0
⊞WorkArea	(TVLOptionalRect

All shown

Check the following connection in the Pin Editor and click OK:



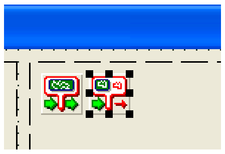
Connections - Source Pin : VLCanny1.OutputPin

Form : Form1 (Current)

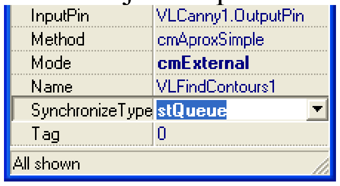
	Sink pin	Component	Connected to
☐	InputPin	VLImageDisplay1	VLAVIPlayer1.OutputPin
☑	InputPin	VLFindContours1	VLCanny1.OutputPin
☐	InputPin	VLImageDisplay2	

Links : 3 ☐ Show all compatible pins

Select the VLFindContours1 component on the form:



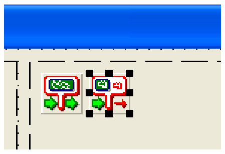
In the Object inspector set the SynchronizeType property to stQueue:



InputPin	VLCanny1.OutputPin
Method	cmAproxSimple
Mode	cmExternal
Name	VLFindContours1
SynchronizeType	stQueue
Tag	0

All shown

Double click on the VLFindContours1 component:



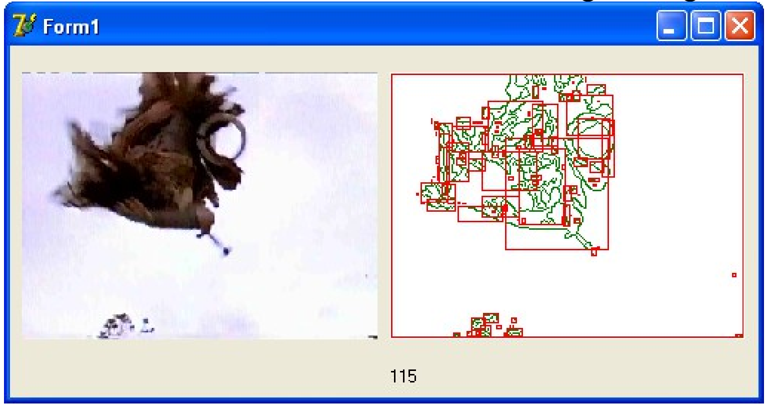
If you are using Delphi, in the event handler add the following code:

```
procedure TForm1.VLFindContours1Contours(Sender: TObject;
  Contours: TVLContours);
var
  I : Integer;
  J : Integer;
  ABitmap : TBitmap;

begin
  ABitmap := TBitmap.Create();
  ABitmap.Width := 240;
  ABitmap.Height := 180;
  Label1.Caption := IntToStr( Contours.Count );
  for I := 0 to Contours.Count - 1 do
    begin
    if( Contours[ I ].ContourType = ctOuter ) then
      ABitmap.Canvas.Pen.Color := clGreen

    else
      ABitmap.Canvas.Pen.Color := clBlue;

    for J := 0 to Contours[ I ].Count - 1 do
      begin
      if( J = 0 ) then
        ABitmap.Canvas.MoveTo( Contours[ I ][ J ].X, Contours[ I ][ J
].Y )

      else
        ABitmap.Canvas.LineTo( Contours[ I ][ J ].X, Contours[ I ][ J
].Y );

      end;
    end;

  ABitmap.Canvas.Pen.Color := clRed;
  ABitmap.Canvas.Brush.Style := bsClear;
  for I := 0 to Contours.Count - 1 do
    ABitmap.Canvas.Rectangle( Contours[ I ].BoundRect );

  VLImageDisplay2.DisplayBitmap( ABitmap );
  ABitmap.Free();
end;
```

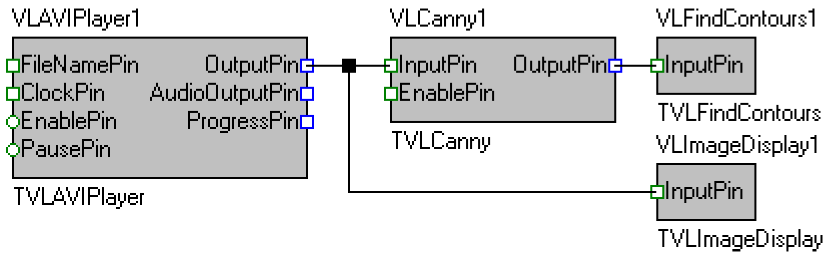
If you are using C++ Builder, in the event handler add the following code:

```
void __fastcall TForm1::VLFindContours1Contours(TObject *Sender,
      TVLContours *Contours)
{
  Graphics::TBitmap *ABitmap = new Graphics::TBitmap();
  ABitmap->Width = 240;
  ABitmap->Height = 180;
  Label1->Caption = Contours->Count;
  for( int i = 0; i < Contours->Count; i ++ )
    {
    if( Contours->Items[ i ]->ContourType == ctOuter )
      ABitmap->Canvas->Pen->Color = clGreen;

    else
      ABitmap->Canvas->Pen->Color = clBlue;
    for( int j = 0; j < Contours->Items[ i ]->Count; j ++ )
      {
      if( j == 0 )
        ABitmap->Canvas->MoveTo( Contours->Items[ i ]->Items[ j ].x,
Contours->Items[ i ]->Items[ j ].y );

      else
        ABitmap->Canvas->LineTo( Contours->Items[ i ]->Items[ j ].x,
Contours->Items[ i ]->Items[ j ].y );

      }
    }

  ABitmap->Canvas->Pen->Color = clRed;
  ABitmap->Canvas->Brush->Style = bsClear;
  for( int i = 0; i < Contours->Count; i ++ )
    ABitmap->Canvas->Rectangle( Contours->Items[ i ]->BoundRect );

  VLImageDisplay2->DisplayBitmap( ABitmap );
  delete ABitmap;
}
```

Compile and run the application.
You should see the contours and the bounding rectangles drawn:



Here are the OpenWire connections in this application:



Using the TSLCRealBuffer in C++ Builder and Visual C++

The C++ Builder version of the library comes with a powerful data buffer class, called TSLCRealBuffer.

The TSLCRealBuffer is capable of performing basic math operations over the data as well as some basic signal processing functions. The data buffer also uses copy on write algorithm improving dramatically the application performance.
The TSLCRealBuffer is an essential part of the SignalLab generators and filters, but it can be used independently in your code.
You have seen already some examples of using TSLCRealBuffer in the previous chapters. Here we will go into a little bit more details about how TSLCRealBuffer can be used.
In order to use TSLCRealBuffer you must include SLCRealBuffer.h directly or indirectly (trough another include file):

```
#include <SLCRealBuffer.h>
```

Once the file is included you can declare a buffer:
Here is how you can declare a 1024 samples buffer:

```
TSLCRealBuffer Buffer( 1024 );
```

Version 4.0 and up does not require the usage of data access objects. The data objects are now obsolete and have been removed from the library.

You can obtain the current size of a buffer by calling the GetSize method:

```
Int ASize = Buffer.GetSize(); // Obtains the size of the buffers
```

You can resize (change the size of) a buffer:

```
Buffer.Resize( 2048 ); // Changes the size to 2048
```

You can set all of the elements (samples) of the buffer to a value:

```
Buffer.Set( 30 ); // Sets all of the elements to 30.
```

You can access individual elements (samples) in the buffer:

```
Buffer [ 5 ] = 3.7; // Sets the fifth elment to 3.7

Double AValue = Buffer [ 5 ]; // Assigns the fifth element to a vari-
able
```

You can obtain read, write or modify pointer to the buffer data:

```
const double *data = Buffer.Read() // Starts reading only
double *data = Buffer.Write()// Starts writing only
double *data = Buffer.Modify()// Starts reading and writing
```

Sometimes you need a very fast way of accessing the buffer items. In this case, you can obtain a direct pointer to the internal data buffer. The buffer is based on copy on write technology for high performance. The mechanism is encapsulated inside the buffer, so when working with individual items you don’t have to worry about it. If you want to access the internal buffer for speed however, you will have to specify up front if you are planning to modify the data or just to read it. The TSLCRealBuffer has 3 methods for accessing the data Read(), Write(), and Modify (). Read() will return a constant pointer to the data. You should use this method when you don't intend to modify the data and just need to read it. If you want to create new data from scratch and don't intend to preserve the existing buffer data, use Write(). If you need to modify the data you should use Modify (). Modify () returns a non constant pointer to the data, but often works slower than Read() or Write(). Here are some examples:

```
const double *pcData = Buffer.Read(); // read only data pointer

double Value = *pcData; // OK!
*pcData = 3.5; // Wrong!


double *pData = Buffer.Write(); // generic data pointer

double Value = *pData; // OK!
*pData = 3.5; // OK!
```

You can assign one buffer to another:

```
Buffer1 = Buffer2;
```

You can do basic buffer arithmetic:

```
TSLCRealBuffer Buffer1( 1024 );
TSLCRealBuffer Buffer2( 1024 );
TSLCRealBuffer Buffer3( 1024 );

Buffer1.Set( 20.5 );
Buffer2.Set( 5 );

Buffer3 = Buffer1 + Buffer2;
Buffer3 = Buffer1 - Buffer2;
Buffer3 = Buffer1 * Buffer2;
Buffer3 = Buffer1 / Buffer2;
```

In this example the elements of the Buffer3 will be result of the operation (+,-,* or /) between the corresponding elements of Buffer1 and Buffer2.
You can add, subtract, multiply or divide by constant:

```
// Adds 4.5 to each element of the buffer
Buffer1 = Buffer2 + 4.5;

// Subtracts 4.5 to each element of the buffer
Buffer1 = Buffer2 - 4.5;

// Multiplies the elements by 4.5
Buffer1 = Buffer2 * 4.5;

// Divides the elements by 4.5
Buffer1 = Buffer2 / 4.5;
```

You can do "in place" operations as well:

```
Buffer1 += Buffer2;
Buffer1 += 4.5;
```

```
Buffer1 -= Buffer2;
Buffer1 -= 4.5;

Buffer1 *= Buffer2;
Buffer1 *= 4.5;

Buffer1 /= Buffer2;
Buffer1 /= 4.5;
```

Those are just some of the basic buffer operations provided by SignalLab.
If you are planning to use some of the more advanced features of TSLCRealBuffer please refer to the online help.
SignalLab also provides TSLCComplexBuffer and TSLCIntegerBuffer. They work similar to the TSLCRealBuffer but are intended to be used with Complex and Integer data. For more information on TSLCComplexBuffer and TSLCIntegerBuffer please refer to the online help.

Distributing your application

Once you have finished the development of your application you most likely will need to distribute it to other systems. In order for some VisionLab built application to work, you will have to include a set of DLL files together with the distribution. The necessary files can be found under the [install path]\DLL directory([install path] is the location where the VisionLab was installed). You can distribute them to the [Windows]\System32 ([Windows]\SysWOW64 on 64 bit Windows) directory, or to the distribution directory of your application([Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS).
Not all of the components in the library require additional DLLs. Please check if the DLLs are needed by the application before including them in the install.

Deploying your 32 bit application with the IPP DLLs

The compiled applications can be deployed to the target system by simply copying the executable. The application will work, however the performance can be improved by also copying the Intel IPP DLLs provided with the library.
The DLLs are under the [install path]\LabPacks\IppDLL\Win32 directory([install path] is the location where the library was installed).
In 32 bit Windows to deploy IPP, copy the files to the [Windows]\System32 directory on the target system.
In 64 bit Windows to deploy IPP, copy the files to the [Windows]\SysWOW64 directory on the target system.
[Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS
This will improve the performance of your application on the target system.

Deploying your 64 bit application

The current version of the library requires when deploying 64 bit applications, the Intel IPP DLLs to be deployed as well.

The DLLs are under the [install path]\LabPacks\IppDLL\Win64 directory([install path] is the location where the library was installed).

To deploy IPP, copy the files to the [Windows]\System32 directory on the target system.
[Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS
This will improve the performance of your application on the target system.