Gator Auto Special Sensor

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory
October 30th, 2003

Written Report
By Robert Peterson

Written Report
By Robert Peterson

Description

The Gator Auto's special sensor is its web cam for image processing. A Logitech 3000 pro web cam was used, however in this implementation any windows compatible camera will work. In fact, that is the strongest advantage of this design—scalability. For example, if multiple cameras are needed or an upgrade is desired, this design is perfect because no coding changes are needed.

The main behavior of this sensor is to follow a road. Thus, this document will explain the algorithm used to do line following with a camera.

Components Needed

(1) Windows Compatible Camera
(2) Windows 98 or higher
(3) Direct X 9.0 Software Development Kit
(4) C# Compiler such as Visual Studio .NET edition
(5) DShowNet.dll

All of the above are free except for the camera.

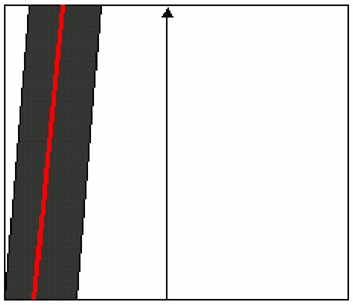
Camera Access Overview

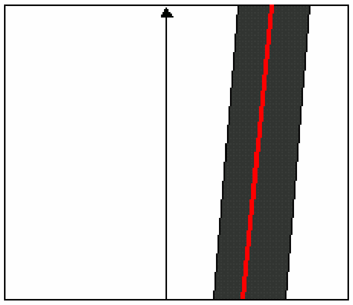
Essentially, the Direct Show libraries for DirectX handle all the hard work. An interface call is used to auto-detect the camera. Once the camera has been initialized, a picture can be taken asynchronously. When the camera is done, a method is triggered by an event called OnCaptureDone(). The returned image is stored in a byte array. The byte array represents each pixel of the image in 3 consecutive bytes—(blue, green, red). Each color can be from 0-255.

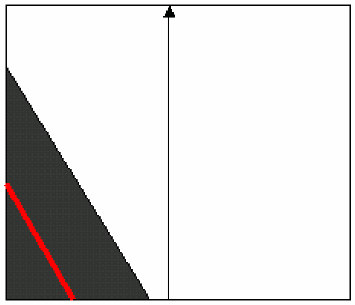
The appendix of this document has a sample program which captures an image from the first auto-detected camera device and outputs it to a file. The file name is specified as a command line argument.

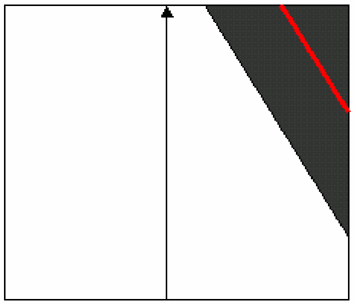
Gator Auto's Line Following Algorithm

The basic concept for following a line is actually very simple. A vertical line drawn at the center of the image may be used as a reference. A best fitting line for the road is determined. If the two lines converge, the robot continues to move forward. If not, it either turns left or right to correct. For example:


Forward


Turn Right


Turn Left


Forward

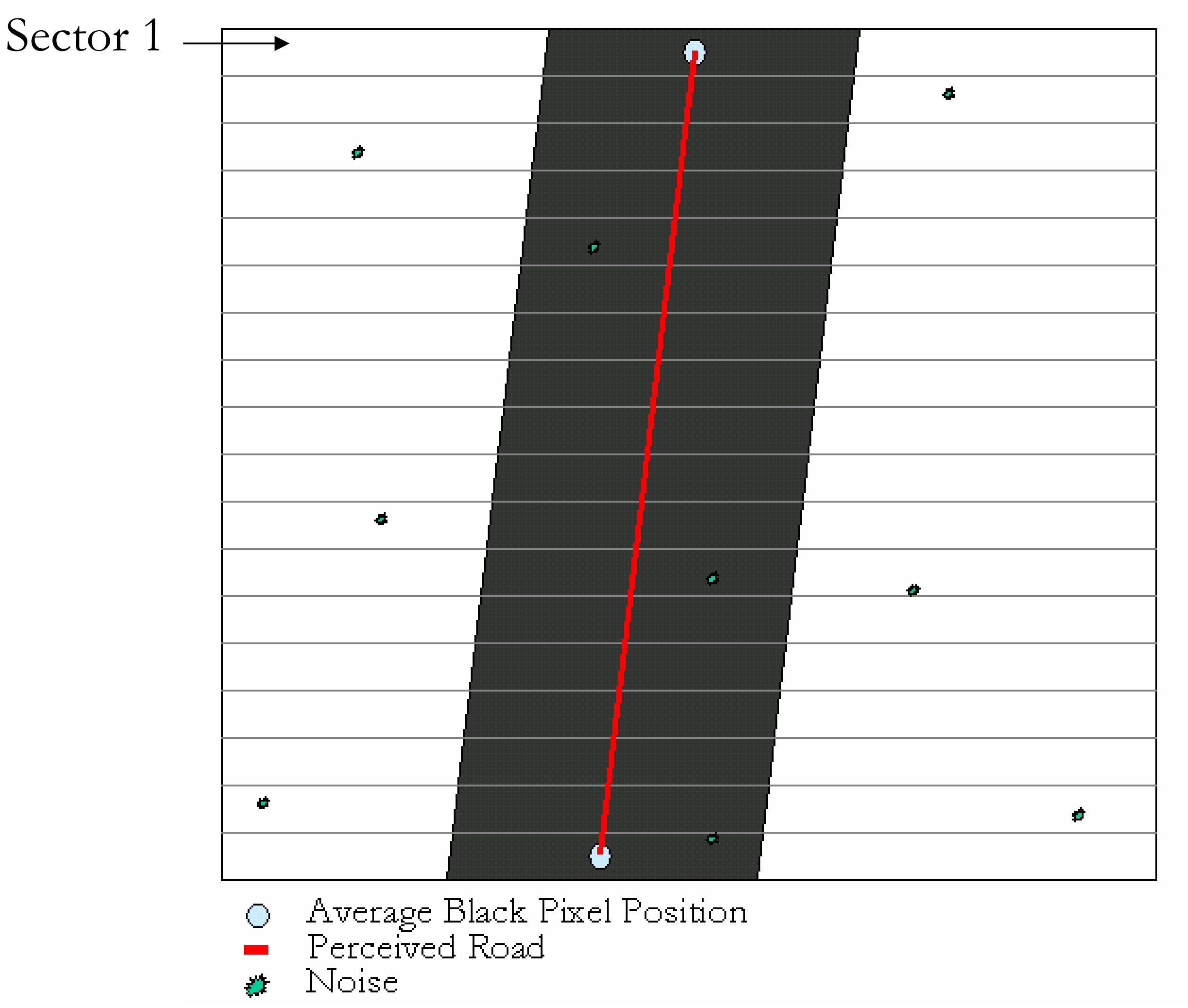
How to Find the Best Fitting Line

Problems arise in trying to find the red lines drawn above. How will noise be handled? What if the Gator Auto needs to go through an intersection?

The way this is overcome is in the algorithm used to determine the line. The image is split into 24 horizontal partitions (each 10 pixels high). These partitions will be referred to as sectors.

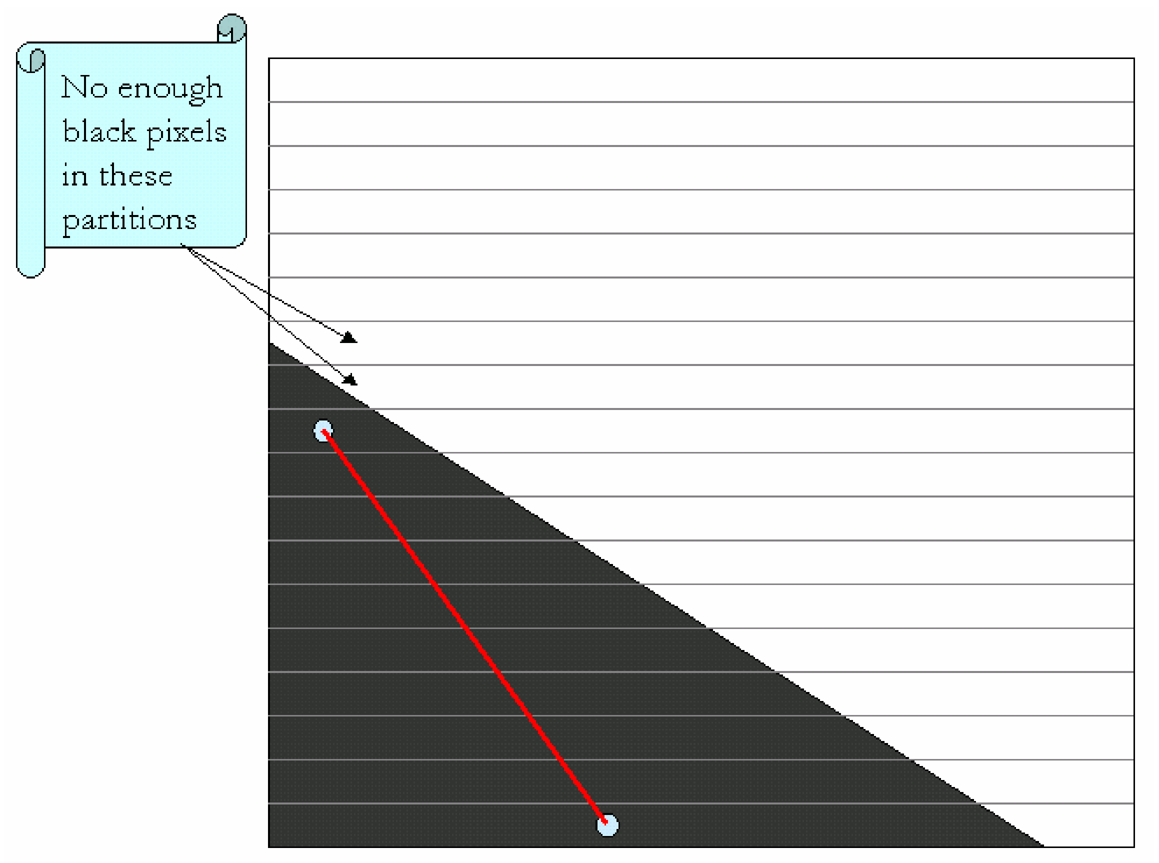
As a loop traverses the byte array, each pixel is tested to see if it is black. If a black pixel is found, the algorithm adds its x coordinate to a running sum. This is then divided by the total amount of black pixels found. When it is done scanning the sector, the resulting value is the average x position of black pixels within a sector.

The top most and bottom most positions are used as points to create a line. This line is what the algorithm considers the road.



Example of the basic algorithm.

There are additional considerations. What if the road does not span the entire Y axis? For example, what if the top most sector has only 1 black pixel because of noise? This should not be considered. The algorithm has a minimum amount of black for a sector to be considered as valid.



Example of case where sectors are not used.

The actual code for this algorithm may be found in appendix of the comprehensive Gator Auto written report.

Appendix

```
/******************************************************
                          DirectShow .NET
                          robbyp@ufl.edu

                          Robby
******************************************************/

using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Collections;
using System.Runtime.InteropServices;
using System.Threading;
using System.Diagnostics;

using DShowNET;
using DShowNET.Device;


namespace Capture
{

        /// <summary> Summary description for MainForm. </summary>
        public class Capture : ISampleGrabberCB
        {

                // so we can wait for the async job to finish
                public ManualResetEvent reset = new ManualResetEvent(true);

                // the image - needs to be global because it is fetched with an async method call
                public Image image;

                /// <summary> base filter of the actually used video devices. </summary>
                private IBaseFilter                     capFilter;

                /// <summary> graph builder interface. </summary>
                private IGraphBuilder                   graphBuilder;

                /// <summary> capture graph builder interface. </summary>
                private ICaptureGraphBuilder2           capGraph;
                private ISampleGrabber                  sampGrabber;

                /// <summary> control interface. </summary>
                private IMediaControl                   mediaCtrl;

                /// <summary> grabber filter interface. </summary>
                public IBaseFilter                      baseGrabFlt;

                /// <summary> structure describing the bitmap to grab. </summary>
```

```
public    VideoInfoHeader                    videoInfoHeader;

/// <summary> buffer for bitmap data. </summary>
public    byte[]                                   savedArray;

/// <summary> list of installed video devices. </summary>
public ArrayList                             capDevices;

private DsDevice dev;

public Capture()
{
        if( ! DsUtils.IsCorrectDirectXVersion() )
        {
                throw new Exception("DirectX 8.1 NOT installed!");
        }

        if( ! DsDev.GetDevicesOfCat( FilterCategory.VideoInputDevice, out capDevices ) )
        {
                throw new Exception("No video capture devices found!");
        }

        dev = capDevices[0] as DsDevice;

        StartupVideo( dev.Mon );

}

public static Image GetImage(Capture cam)
{
        try
        {

                Thread.Sleep(200);

                cam.reset.Reset();

                cam.CaptureImage();

                return cam.image;
        }
        catch (Exception ex)
        {
                cam.CloseInterfaces();
                Debug.WriteLine(ex);
                return null;
        }
}

public static DsDevice[] GetInterfaceList()
{
        ArrayList devices = null;
```

```
			if( ! DsUtils.IsCorrectDirectXVersion() )
			{
				throw new Exception("DirectX 8.1 NOT installed!");
			}

			if( ! DsDev.GetDevicesOfCat( FilterCategory.VideoInputDevice, out devices ) )
			{
				//throw new Exception("No video capture devices found!");
				devices = new ArrayList();
			}

			return (DsDevice[])devices.ToArray(typeof(DsDevice));
		}

		private void CaptureImage()
		{

			int hr;
			if( sampGrabber == null )
				return;

			if( savedArray == null )
			{
				int size = videoInfoHeader.BmiHeader.ImageSize;
				// debugging sanity check
				//if( (size < 1000) || (size > 16000000) )
				//		return;
				//savedArray = new byte[ size + 64000 ];
			}

			hr = sampGrabber.SetCallback( this, 1 );

			reset.WaitOne();
		}

		public void Init()
		{
			StartupVideo( dev.Mon );
		}

		private void Init(DsDevice device)
		{
			// store it for clean up.
			capDevices = new ArrayList();
			capDevices.Add(device);

			StartupVideo( device.Mon );
		}

		/// <summary> capture event, triggered by buffer callback. </summary>
		private void OnCaptureDone()
		{
			if( sampGrabber == null )
				return;

			int w = videoInfoHeader.BmiHeader.Width;
			int h = videoInfoHeader.BmiHeader.Height;
			if( ((w & 0x03) != 0) || (w < 32) || (w > 4096) || (h < 32) || (h > 4096) )
				return;

			int stride = w * 3;

			GCHandle handle = GCHandle.Alloc( savedArray, GCHandleType.Pinned );
			int scan0 = (int) handle.AddrOfPinnedObject();
			scan0 += (h - 1) * stride;
			image = new Bitmap( w, h, -stride, PixelFormat.Format24bppRgb, (IntPtr) scan0 );

//*** YOU CAN ACCESS THE IMAGE DIRECTLY HERE
//*** savedArray contains the image

			handle.Free();

			savedArray = null;
			reset.Set();
		}



		/// <summary> start all the interfaces, graphs and preview window. </summary>
		private bool StartupVideo( UCOMIMoniker mon )
		{
			int hr;
			if( ! CreateCaptureDevice( mon ) )
				return false;

			if( ! GetInterfaces() )
				return false;

			if( ! SetupGraph() )
				return false;

			hr = mediaCtrl.Run();
			if( hr < 0 )
				Marshal.ThrowExceptionForHR( hr );

			return true;
		}

		/// <summary> build the capture graph for grabber. </summary>
		private bool SetupGraph()
		{
			int hr;
			hr = capGraph.SetFiltergraph( graphBuilder );
```

```
            if( hr < 0 )
                    Marshal.ThrowExceptionForHR( hr );

            hr = graphBuilder.AddFilter( capFilter, "Ds.NET Video Capture Device" );
            if( hr < 0 )
                    Marshal.ThrowExceptionForHR( hr );

            AMMediaType media = new AMMediaType();
            media.majorType  = MediaType.Video;
            media.subType     = MediaSubType.RGB24;
            media.formatType = FormatType.VideoInfo;                // ???
            hr = sampGrabber.SetMediaType( media );
            if( hr < 0 )
                    Marshal.ThrowExceptionForHR( hr );

            hr = graphBuilder.AddFilter( baseGrabFlt, "Ds.NET Grabber" );
            if( hr < 0 )
                    Marshal.ThrowExceptionForHR( hr );

            Guid cat;
            Guid med;

            cat = PinCategory.Capture;
            med = MediaType.Video;
            hr = capGraph.RenderStream( ref cat, ref med, capFilter, null, baseGrabFlt );
              // baseGrabFlt above

            media = new AMMediaType();
            hr = sampGrabber.GetConnectedMediaType( media );
            if( hr < 0 )
                    Marshal.ThrowExceptionForHR( hr );
                      if( (media.formatType != FormatType.VideoInfo) || (media.formatPtr ==
                      IntPtr.Zero) )
                    throw new NotSupportedException( "Unknown Grabber Media Format" );

            videoInfoHeader = (VideoInfoHeader) Marshal.PtrToStructure( media.formatPtr,
                  typeof(VideoInfoHeader) );
            Marshal.FreeCoTaskMem( media.formatPtr ); media.formatPtr = IntPtr.Zero;

            hr = sampGrabber.SetBufferSamples( false );
            if( hr == 0 )
                    hr = sampGrabber.SetOneShot( false );
            if( hr == 0 )
                    hr = sampGrabber.SetCallback( null, 0 );
            if( hr < 0 )
                    Marshal.ThrowExceptionForHR( hr );

            return true;
    }


    /// <summary> create the used COM components and get the interfaces. </summary>
```

```
private bool GetInterfaces()
{
	Type comType = null;
	object comObj = null;
	try
	{
		comType = Type.GetTypeFromCLSID( Clsid.FilterGraph );
		if( comType == null )
			throw new NotImplementedException( @"DirectShow FilterGraph
			  not installed/registered!" );
		comObj = Activator.CreateInstance( comType );
		graphBuilder = (IGraphBuilder) comObj; comObj = null;

		Guid clsid = Clsid.CaptureGraphBuilder2;
		Guid riid = typeof(ICaptureGraphBuilder2).GUID;
		comObj = DsBugWO.CreateDsInstance( ref clsid, ref riid );
		capGraph = (ICaptureGraphBuilder2) comObj; comObj = null;

		comType = Type.GetTypeFromCLSID( Clsid.SampleGrabber );
		if( comType == null )
			throw new NotImplementedException( @"DirectShow
			  SampleGrabber not installed/registered!" );
		comObj = Activator.CreateInstance( comType );
		sampGrabber = (ISampleGrabber) comObj; comObj = null;

		mediaCtrl	= (IMediaControl) graphBuilder;
		baseGrabFlt	= (IBaseFilter)		sampGrabber;
		return true;
	}
	catch( Exception ee )
	{
		if( comObj != null )
			Marshal.ReleaseComObject( comObj ); comObj = null;

		throw ee;
	}
}

/// <summary> create the user selected capture device. </summary>
private bool CreateCaptureDevice( UCOMIMoniker mon )
{
	object capObj = null;
	try
	{
		Guid gbf = typeof( IBaseFilter ).GUID;
		mon.BindToObject( null, null, ref gbf, out capObj );
		capFilter = (IBaseFilter) capObj; //capObj = null;
		return true;
	}
	catch( Exception ee )
	{
		if( capObj != null )
```

```
            Marshal.ReleaseComObject( capObj ); //capObj = null;
        throw ee;
    }
}



/// <summary>
/// MUST do this.
/// Notice alot of crap in here - I've had some problems over extending the bandwidth of
/// the USB bas.
/// </summary>
private void CloseInterfaces()
{
        int hr;
        try
        {
            if( mediaCtrl != null )
            {
                hr = mediaCtrl.Stop();
                mediaCtrl = null;
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex);
        }

                baseGrabFlt = null;

                try
                {
                    if( sampGrabber != null )
                    Marshal.ReleaseComObject( sampGrabber );
                    sampGrabber = null;
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(ex);
                }

            try
                {
                    if( capGraph != null )
                    Marshal.ReleaseComObject( capGraph );
                    capGraph = null;
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(ex);
                }
```

```
                        try
                        {
                            if( graphBuilder != null )
                            Marshal.ReleaseComObject( graphBuilder );
                            graphBuilder = null;
                        }
                        catch (Exception ex)
                        {
                            Debug.WriteLine(ex);
                        }

                        try
                        {
                            if( capFilter != null )
                            Marshal.ReleaseComObject( capFilter );
                            capFilter = null;
                        }
                        catch (Exception ex)
                        {
                            Debug.WriteLine(ex);
                        }

                        try
                        {
                            if( capDevices != null )
                            {
                                foreach( DsDevice d in capDevices )
                                    d.Dispose();

                                capDevices = null;
                            }
                        }
                        catch (Exception ex)
                        {
                            Debug.WriteLine(ex);
                        }

        }

        /// <summary> sample callback, NOT USED. </summary>
        int ISampleGrabberCB.SampleCB( double SampleTime, IMediaSample pSample )
        {
            return 0;
        }

        /// <summary> buffer callback, COULD BE FROM FOREIGN THREAD. </summary>
        int ISampleGrabberCB.BufferCB( double SampleTime, IntPtr pBuffer, int BufferLen )
        {
            if( savedArray == null )
            {
                return 0;
            }
```

```
                        if( (pBuffer != IntPtr.Zero) && (BufferLen > 1000) && (BufferLen <=
savedArray.Length) )
                                Marshal.Copy( pBuffer, savedArray, 0, BufferLen );

                        this.OnCaptureDone();
                        return 0;
                }

                public static void Main(string[] args)
                {
                        if(args.Length != 1)
                        {
                                Console.WriteLine("Path for image output needed as argument\nexample:
                                    c:\\pic.jpg");
                        }
                        else
                        {

                                Capture cam = new Capture();

                                while(true)
                                {

                                        Image image = Capture.GetImage(cam);

                                        string path = args[0];

                                        image.Save( path, ImageFormat.Jpeg );

                                        image.Dispose();

                                }

                                cam.CloseInterfaces();

                                cam = null;
                        }
                }
        }
}
```