The Gator Auto

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

Written Report
By Robert Peterson

Table of Contents

Abstract

The purpose of this report is to give a detailed description of how the Gator Auto was successfully designed and constructive. It may be invaluable for (1) construction of a machine which has weight or power constraints (2) interfacing traditional digital design or micro-controller with a laptop *or* (3) a design which carries a laptop

Executive Summary

This report should give sufficient information about the construction of the Gator Auto. The robot was designed, assembled, and tested within one semester in the world renown University of Florida Intelligent Machine Design Laboratory. The Gator Auto is a robot which emulates the behaviors of an automobile at a miniature scale.

Introduction

This report is a detailed descriptions of the objectives, specifications, requirements, and design characteristics of the Gator Auto Robot. The Gator Auto is essentially a platform which carries a laptop and a Logitech 3000 pro webcam. It's purpose is to use the webcam to emulate a real automobile, by using line following and basic image processing.

MAIN BODY

Integrated System

This section will give a complete organizational description of the Gator Auto system.

The robot is composed of the following components

i. Atmel Mega128 Microcontroller
ii. Servos attached to wheels
iii. Two Sharp Infra-Red Sensor
iv. Laptop coupled with a Web Cam
v. Platform/Frame
vi. Battery Pack

The most interesting interface is that between the laptop and the micro-controller. Three pins are connected between the laptop's parallel port and the laptop. These pins have different codes for what direction if any the robot should move. The micro-controller continually polls these pins and controls the motors accordingly. For example, [001] implies that the robot needs to turn right. When the laptop recognizes that the appropriate angle of rotation has been reached it may write 0x00 to the parallel port register, [000], which means to stop. So, in fact the real autonomous "smarts" of the robot is done in a C# program in the laptop.

The micro controller drives the movement of the robot and the constantly checks the state of the IR Sensors and the Laptop Pins. In this design the laptop system is considered to be no more then a very heavy sensor. The other type of sensor is the digital IR component.

The battery pack powers every component except the laptop which has its own battery. Everything runs on 5V except the servos which require their own voltage regulators. Two L7806CV voltage regulators were used for the servos. Each servo needs its own voltage regulator (or another way to think of it is two regulators in parallel), because of the potential current they can source. Each regulator is only 1.5A. Together they supply up to 3A. A L7805CV voltage regulator was used to supply 5VDC to the micro controller, LCD Display, and IR Sensors.

Mobile Platform

This section will describe the specifications and objectives of the platform. Also, any lessons learned.

The objectives of the platform are

i. Robustness for two semester lifespan
ii. Structurally sound regardless of the laptop payload
iii. Design which will allow easy removal of the laptop
iv. Metal of some form
v. Design which resembles an automobile

The design of the platform was done as simply as possible. The frame consists only of a rectangular panel with several screw holes. Everything attaches to this flat panel either below or above it. For example, the circuitry is all on a pc board which screws in below the panel. The LCD and camera are above near the front of the panel, etc. Clearly, the laptop sits on top.

Actuation

The actuation section will cover the scope, specifications, and objectives for the robot's movement.

The Gator Auto has only one type of actuation—movement of the platform. This is done using the following components

i. Two Hitec HS-700BB servos from http://www.servocity.com
    a. Ball Bearing
    b. Operating Speed is 0.22sec/60° at 4.8V
    c. Output Torque is 9.5kg·cm or 133oz·in at 4.8V
    d. Size is 59x29x52mm
    e. Voltage range is 4.8V to 6V
    f. Requires 3-5V peak to peak square wave pulse of duration from 0.9mS to 2.1mS as center. Pulse refreshes at 50Hz (20mS)
    g. Current drain is 8mA idle and 230mA no load operating at 4.8V
ii. Two 3.5" car-like tires from http://www.lynxmotion.com
iii. One traditional furniture caster from Lows
iv. Kokam USA, Lithium Polymer Battery from http://www.fmadirect.com/

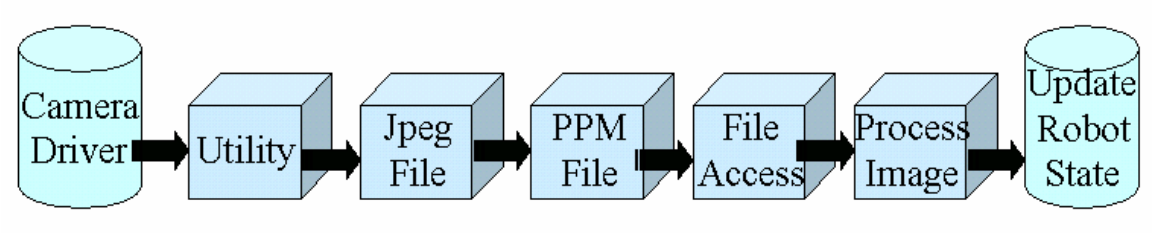
Sensors

The following will describe the scope, specifications, and objectives of the two onboard sensors.

Two sensors were used in this design. The most notable of which is the laptop coupled with a Logitech Web Cam.

Each image is processed in a C# program and a state is determined for output to the parallel port. The state can be (1) Turn Left (2) Go Straight (3) Stop (4) Turn Right

Initially the author tried using a web cam utility which outputs 2 jpeg images a second to the hard disk. This approach was inefficient because of all the stages involved and the poor capture rate of the shareware utility:

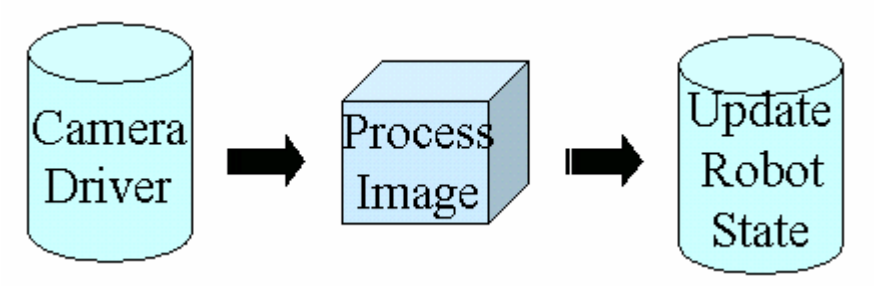


Soon, the author realized that converting from jpeg to ppm and the slow capture rate of the utility would make the frame rate too slow for the image processing to be able to react to the line or "road."

The solution is to access the camera directly with Windows DirectShow. Visual Studio is available free to all engineers at http://www.msdnaa.eng.ufl.edu/

C# was chosen for development because it is basically Microsoft's version of Java and arguably the easiest most modern OOP language available. A tutorial which captures a single image from any windows compatible web cam may be found at: http://www.codeproject.com/csharp/webcamservice.asp

Now the frame rate is far more efficient:



As described there are two sensor components

i. Laptop/Web Cam
   a. IBM 570 Laptop 3.7LB, datasheet at http://www.ibm.com
   b. Windows 98 Second edition for easy access to output pins
   c. Logitech 3000 pro Web Cam
   d. Parallel Port Output—3 pins represent sensor state
   e. Image processing done in Microsoft C#

ii. Sharp GPD15 Infra-Red Sensor from http://www.junun.org
   a. Digital output which reads high within 24cm

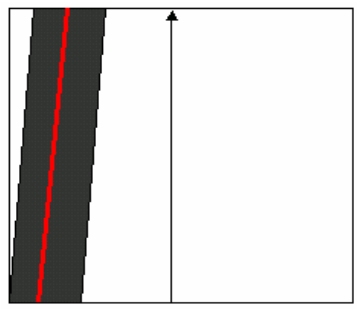
Behaviors

This behaviors section will describe the scope, specifications, and objectives of the Gator Auto behaviors.

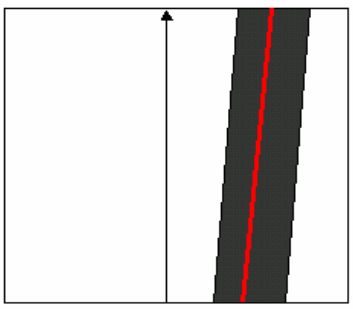
The Gator Auto has three behaviors

i. Follows a small scale road using the web cam and image processing
ii. Stops at a miniature scale deer crossing when an object is present
   The digital Sharp IR sensor is used for this behavior
iii. Follows traffic regulations at a miniature stoplight using the web cam

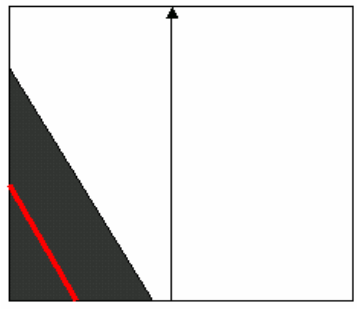
The algorithm for following a line is actually very simple. A vertical line drawn at the center of the image may be used as a reference. A best fitting line for the road is determined. If the two lines converge, the robot continues to move forward. If not, it either turns left or right to correct. For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



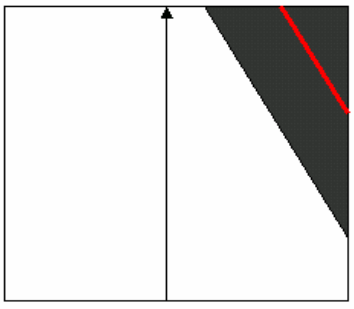
Forward



Turn Right



Turn Left



Forward

Experimental Layout and Results

The layout and results were done by constructing a miniature oval shaped track representing the road. The road is constructed by using interlocking ceiling tiles bought from Lows for about $20. Each square ft tile has a piece of the road on it. . Transporting the tiles can be done easily and constructing the road is as simple as putting together a toy train track. The material used for the road is called Flexi-Foam. It was chosen for it's low reflectivity. Electrical tape, which was originally used, is not desirable because at certain angles from a light source, electrical tape is perceived as white from a camera. The Flexi-Foam can be purchased at a Michael's crafts store and is manufactured by FibreCraft.

CLOSING

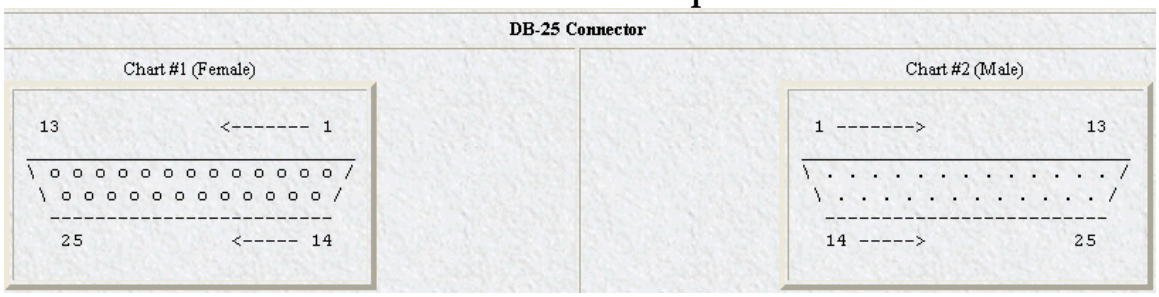
Conclusion

This should be a good start for any project which carries a laptop or involves image processing.

If you are anything like the author, you don't read lengthy reports like this too often. Thank you for taking the time to take a peek into his world and feel free to contact him if it was somehow useful to you. This project was successful and rewarding.

Appendices

Windows 98 Parallel Port Specifications



Input to the laptop can be done using pins [11, 10, 12, 13, 15]

They can be read by a program on the pc at memory location 0x3BD (base + 1)

The mapping is as follows

pins[11|10|12|13|15] ⇔ 0x3BD[11|10|12|13|15|X|X|X]

Bit 11 is inverted

C# Image Processing Source

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

         For this code to work, you need DShowNet.dll
         The purpose of this code is to continuously input
         images from a windows device driver compatible
         Web Cam and find the best fitting line for black
         pixels representing a road.  It then determines
         if a robot should go straight, turn left, or
         turn right in order to stay on the road.
*****************************************************/

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;

		// path of printer port output application
		private string PrinterPortAppPath;

		private DsDevice dev;

		// command line parameters
		private int centerPartitionSize,
					sectorPotentcy,
```

```
                                maxRecSlope,
                                blackBound,
                                redRedUpBound,
                                redRedLowBound,
                                redGrnUpBound,
                                redGrnLowBound,
                                redBlueUpBound,
                                redBlueLowBound,
                                ledPotentcy;

        public Capture(    string arg0,
                                        string arg1,
                                        string arg2,
                                        string arg3,
                                        string arg4,
                                        string arg5,
                                        string arg6,
                                        string arg7,
                                        string arg8,
                                        string arg9,
                                        string arg10,
                                        string arg11)
        {
                PrinterPortAppPath = arg0;
                centerPartitionSize = int.Parse(arg1);
                sectorPotentcy = int.Parse(arg2);
                maxRecSlope = int.Parse(arg3);
                blackBound = int.Parse(arg4);
                redRedUpBound = int.Parse(arg5);
                redGrnUpBound = int.Parse(arg6);
                redBlueUpBound = int.Parse(arg7);
                redRedLowBound = int.Parse(arg8);
                redGrnLowBound = int.Parse(arg9);
                redBlueLowBound = int.Parse(arg10);
                ledPotentcy = int.Parse(arg11);

                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;
                // 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 );

        /* algorithm attribues */
        double black_count = 0;             // running sum of black pixels in a row
        double black_sum = 0;               // running sum of x coordinates for black pixels
        double point_count = 0;             // same as black count but only used for partitions
        double point_sum = 0;
        int position;                       // x coordinate of a pixel
        int cursor = 0;                     // current position of image byte array
        int ledColorCnt = 0;                // running sum of "red" pixels
        double d;                           // average x coordinate for all black pixels in the image
        int red, green, blue;               // byte values from 0-255 representing rgb
        int partition = 0;                  // index to current horizontal partition of the image
        double[] points = new double[24];   // holds the average x coordinate for each partition
        int point_pos = 0;                  // index to current parition, essential index for points[]
        double rise, run, slope;            // for slope of perceived road, best fitting line for all black
        String lastTurn = "Forward";        // default behavior for error
        for(int i = 1; i <= 240; i++ )
        {
                for(int j = 1; j <= 320; j++ )
                {
                        blue = (uint) savedArray[cursor++];
                        green = (uint) savedArray[cursor++];
                        red = (uint) savedArray[cursor++];

                        // makes the center of the screen the origin in x axis
                        position = j - 160;

                        // is this pixel black?
                        if( red <= blackBound && green <= blackBound && blue <= blackBound )
                        {
```

```
                                point_count++;
                                point_sum += position;

                                black_count++;
                                black_sum += position;
                        }

                        /* detecs red in the image, for stop light novelty */
                        if( red >= redRedLowBound && red <= redRedUpBound &&
                                green >= redGrnLowBound && green <= redGrnUpBound &&
                                blue >= redBlueLowBound && blue <= redBlueUpBound )
                        {
                                ledColorCnt++;
                        }
                }

                if(i > (partition+9))
                {
                        // need to use only potent sectors of black
                        if(point_count < sectorPotentcy)
                        {
                                point_sum = 0;
                                points[point_pos] = 0;
                        }
                        else
                        {
                                points[point_pos] = point_sum / point_count;
                        }

                        //re-init
                        point_pos++;
                        point_count = 0;
                        point_sum = 0;
                        partition += 10;
                }
        }

        // find best points to make slope
        int topPoint = 0;
        int bottomPoint = 0;

        for(int i = 0; i < 24; i++)
        {
                if( points[i] != 0 )
                {
                        bottomPoint = i;
                        break;
                }
        }

        for(int i = 23; i >= 0; i--)
        {
                if( points[i] != 0 )
                {
                        topPoint = i;
                        break;
                }
```

```
}

// calculate rise and run
rise = 5+10*(23-topPoint) - 5+10*(23-bottomPoint);
run = points[topPoint] - points[bottomPoint];

// thinks a stop light is present, there is enough "red" in the image
if(ledColorCnt > ledPotentcy)
{
        execute("Stop");
}

// CHECK FOR DIVISON BY ZERO
else if(run == 0)
{
        // Console.WriteLine(lastTurn);
        execute(lastTurn);
}
else if(black_count == 0)
{
        execute(lastTurn);
}
else
{
        d = black_sum / black_count;

        // calculate slope
        slope = rise/run;

        // Console.WriteLine("Slope:"+slope);

        double pi = System.Math.PI;

        double desiredSlope = System.Math.Tan((pi*d)/640 + pi/2);

        // case for vertical line
        if(slope > maxRecSlope || slope < -maxRecSlope)
        {
                if(d >= centerPartitionSize)
                {
                        // Console.WriteLine("Turn Right");
                        execute("Right");
                        lastTurn = "Right";
                }
                else if(d <= -centerPartitionSize)
```