Encoder
The Newsletter of the Seattle Robotics Society

Implementing Vision Using a GAME BOY® Camera on the MRM

Author:	**Dafydd Walters <mailto:dafydd@walters.net>**
Level:	**Intermediate**
Revision:	**1.0**
Date:	**January 1, 2002**

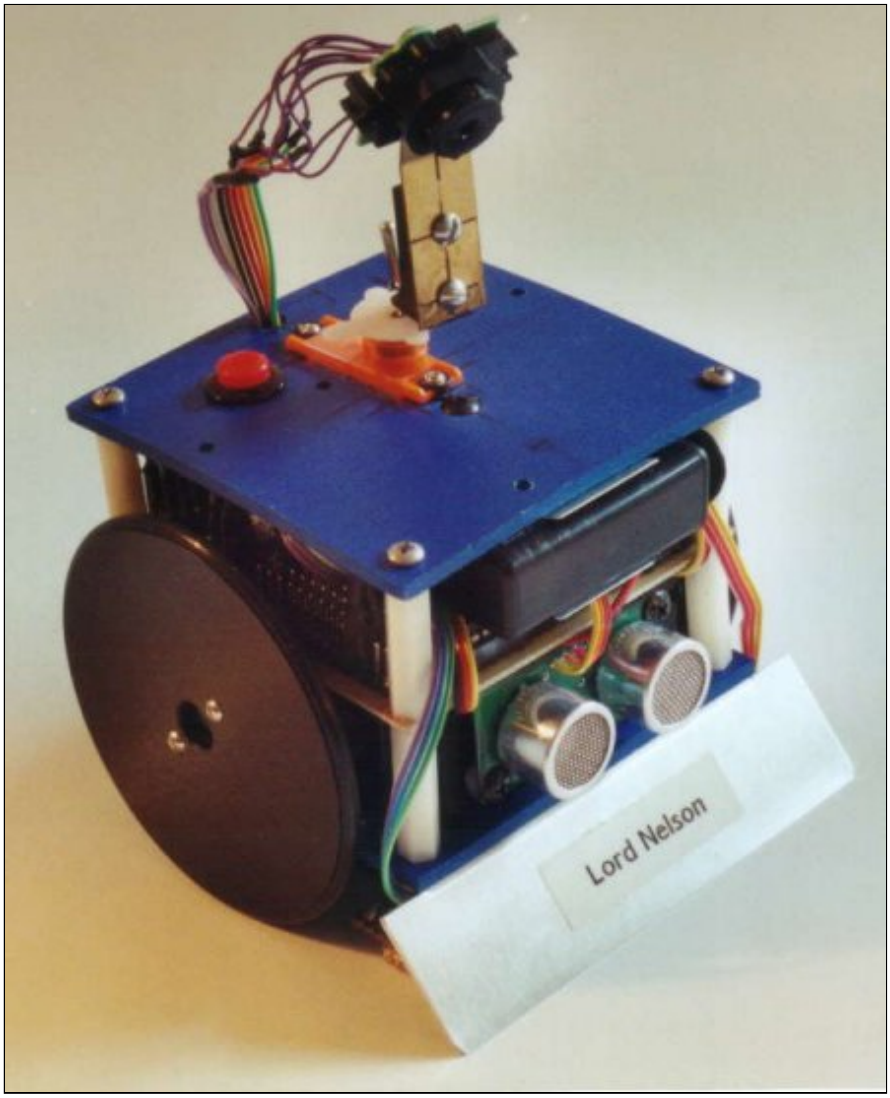

Figure 1. Lord Nelson - Mini Sumo Robot.

1. Abstract

One of the factors that has limited the prevalence of vision systems in amateur robotics has been the lack of affordable processing power in a small package. Another has been the difficulty of obtaining a low-cost vision sensor that is easy to interface with a controller board.

This article describes how to implement a simple monochrome vision system on a robot powered by the *Mini RoboMind* (MRM). The MRM is a small, powerful, inexpensive 68332-based robot controller designed by Mark Castelluccio. The vision sensor is a CMOS image sensor taken from the GAME BOY® Camera, an accessory for the GAME BOY® handheld gaming device.

This article covers the basics of interfacing the image sensor with the controller board, and includes code samples for both the MRM and a host PC (connected via serial port) to allow captured images to be viewed on the host PC. Image processing techniques are beyond the scope of this article, but I've included a few pointers.

The host PC code for this article is written in Java™, and has been tested on Windows® and Linux® platforms. In theory, it should also run on any other operating system that supports Java™ 2, as long as there is also a working implementation of the Java™ COMM API available (the Java™ COMM API is a platform-independent programming interface that Java™ applications can use to access communication devices such as serial ports and parallel ports).

2. Hardware

This article assumes that in addition to basic tools and a host PC, you have the following components:

- A GAME BOY® Camera
- An MRM with serial cable
- A few inches of ribbon cable (9 wires)
- Some narrow-bore shrink-wrap tubing
- Some 0.1" pitch female headers for the MRM

2.2 Obtaining the GAME BOY® Camera

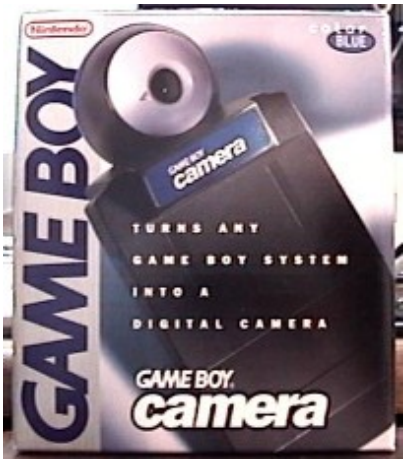

Figure 2. GAME BOY® Camera.

Although it seems increasingly difficult to obtain the now obsolete GAME BOY® Cameras, it is still possible to mail-order them from some retailers at the time of writing. Try navigating to MySimon.com, and enter **GAME BOY Camera** in the search field. There are also a few people selling them on the eBay auction site too.

2.3 Obtaining the MRM

The MRM is available by mail order from the RoboMinds web site in a variety of different configurations and options.

At the time of writing, the basic MRM is available for under $100. The RAM and processor upgrades are well worth the money, and the package bundles represent good value. The *MRM MAX RS232* package is a great starter package, and is ideal for this project.

For some nitty-gritty technical details on the MRM, consult the article "*Three 68332 Designs Presented*" included in the Links section at the end of this article.

2.4 Extracting the Mitsubishi M64282FP Artificial Retina

At the heart of the GAME BOY® Camera is the M64282FP CMOS image sensor from Mitsubishi. You can download the data sheet for this device from the Links section at the end of the article.

This device, which requires only a single 5-volt power supply, has an effective resolution of 128 pixels across by 123 pixels vertically. The M64282FP interfaces with the host micro via an interesting combination of digital and analog I/O. All control and clocking signals are digital, but the intensity of each image pixel is output as an analog voltage. The MRM's built-in fast analog to digital converter is perfect for this application, enabling the sensor to be connected to the MRM with zero "glue" components.

Figure 3. M64282FP CMOS Image Sensor - Lens Removed.

To access the image sensor, you will need to open the case by removing the four screws in the back (warning: this will obviously void any warranty you may have had when you purchased the GAME BOY® Camera). This is not as easy as it sounds, as these are special screws with three slots. I have managed to get them out with difficulty, using a small flat blade screwdriver, but it's much easier with the proper tri-wing bit tool.

Radio Shack sells a set of security bits that includes the special tri-wing bit for around $13. The part number is 910-5247. You can mail-order this from radioshack.com. Alternatively you can mail-order a suitable screwdriver from A. I. Trading Ltd in Hong Kong (click on *GBA*, then scroll down to *Screw Driver*). This costs $7 plus $5 shipping. It's a bit expensive for a small screwdriver, but it works well. Delivery is about a week.

Figure 4. GAME BOY® Camera - Front and Back.

After removing the back of the case, the eyeball may be unplugged from the main board. Whether or not you remove the sensor from the eyeball is up to you; the same awkward three-slot screws hold the eyeball case together. You can see from the picture of my Mini Sumo robot that I removed the eyeball case - this was to keep the weight of the robot down.

If you decide to remove the image sensor PCB from the eyeball housing, you will see that the image sensor is mounted on one side of the board (covered by the plastic lens housing), and the 9-way connector is on the other (see Figures 5 and 6).

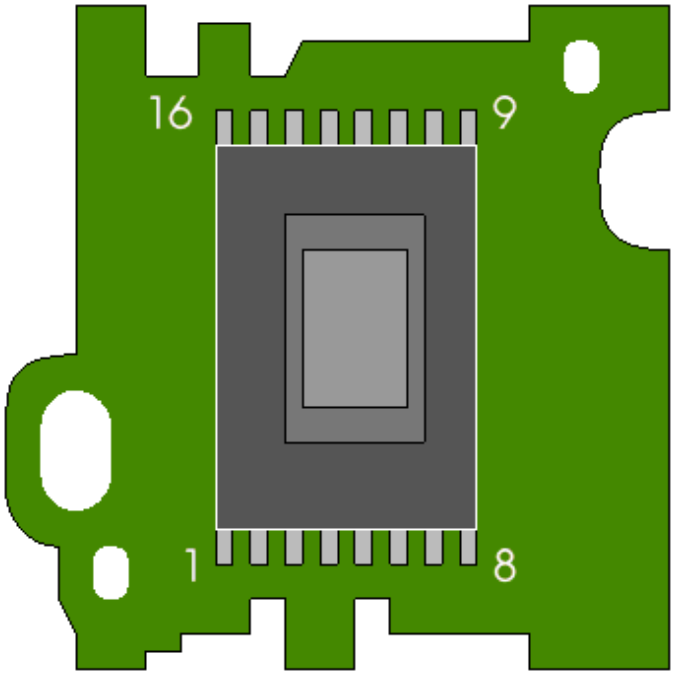

Figure 5. Layout of Image Sensor Circuit Board - Sensor Side (M64282FP pin numbers shown)

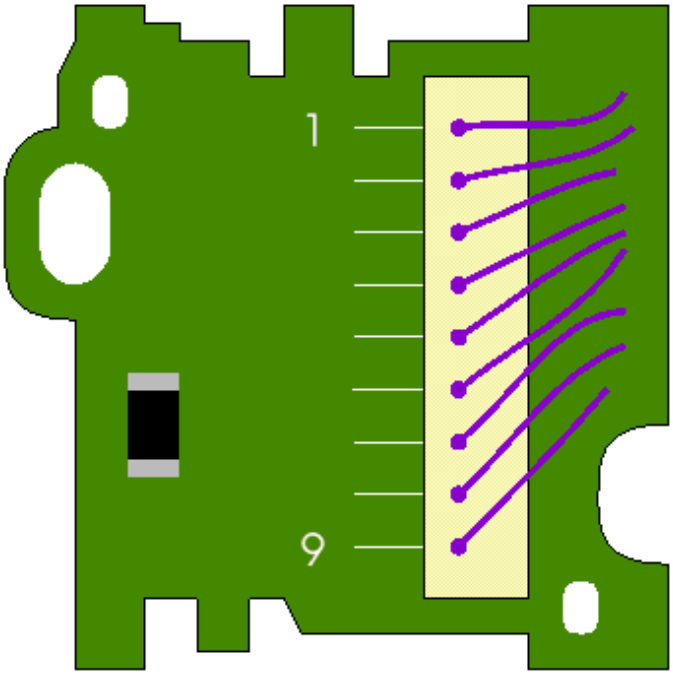

Figure 6. Layout of Image Sensor Circuit Board - Back (wire numbers shown)

2.5 Connecting the Image Sensor to the MRM

This is probably the trickiest step of the whole operation, but with a little care, this can be accomplished successfully.

Refer to Figure 7 and Table 1 to see how the wires need to be connected to the MRM.

You may think of a better way of doing this than I, but I found that cutting off the connector one wire at a time (so that I could keep track of which wire I was connecting), and soldering directly to ribbon-cable worked well. I used fine-bore shrink-wrap tubing to insulate and strengthen the joints, and I made a careful note of the color of the ribbon-cable wire as I connected each one.

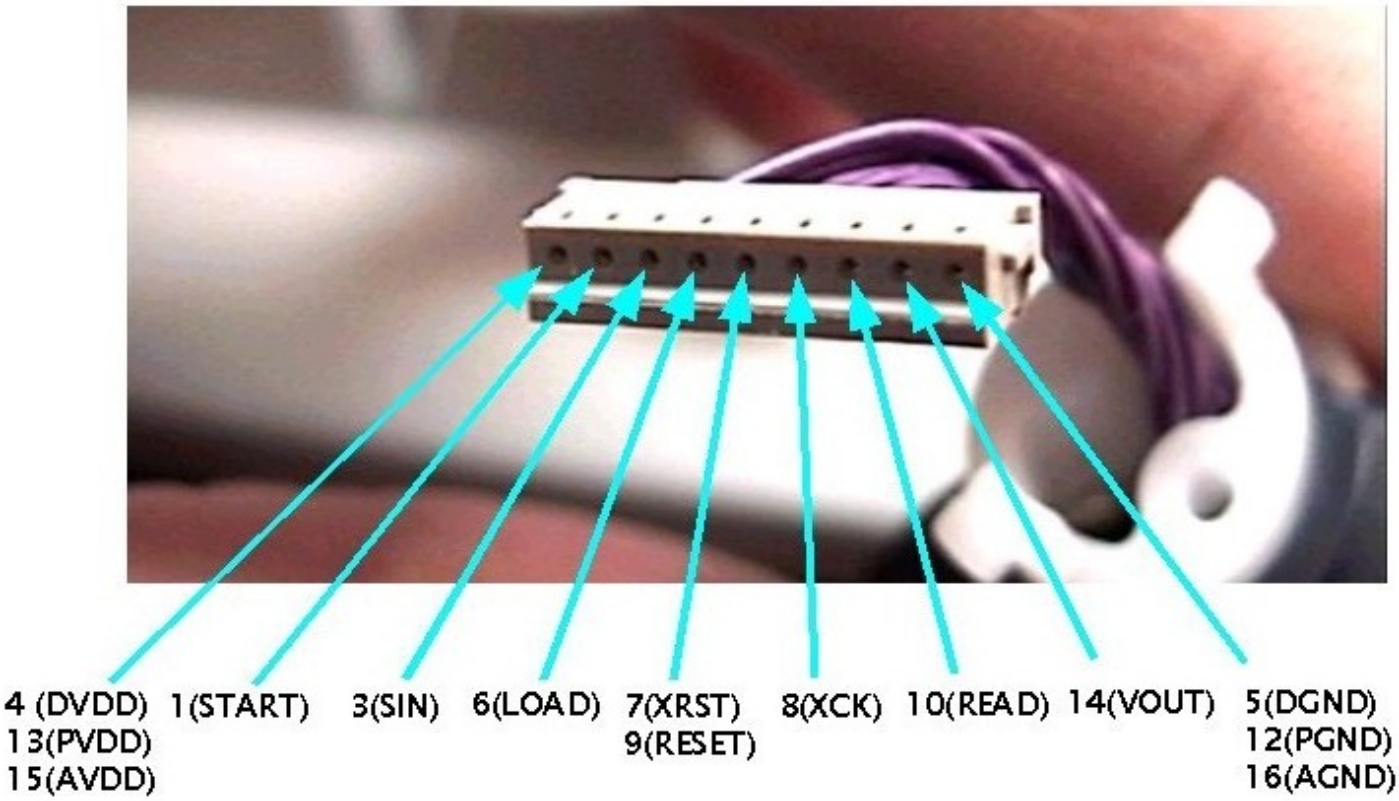

Figure 7. Eyeball Connector Pin-out (M64282FP pin numbers shown)

Wire	M64282FP Pin(s)	Symbol(s)	Function	MRM Pin
1	4,13,15	DVDD,AVDD1,AVDD2	+5V Power Supply	Power (+5V)
2	1	START	Start *Input*	Port E Bit 5
3	3	SIN	Data *Input*	Port E Bit 4
4	6	LOAD	Data Set *Input*	Port E Bit 3
5	7,9	Xrst,RESET	System+Memory Reset *Input*	Port E Bit 2
6	8	Xck	System Clock *Input*	Port E Bit 1
7	10	READ	Read Image *Output*	Port E Bit 0
8	14	Vout	Analog Signal *Output*	ADC Port 0
9	5,12,16	DGND,AGND1,AGND2	Ground	GND (0V)

Table 1. Eyeball to MRM Connections

At the MRM end of the ribbon cable, you will need to solder or crimp the wires to 0.1" female headers which will plug into the MRM. Figure 8 shows the pins on the MRM on which the headers will be inserted. The numbers in the red circles are the port bit numbers (in the case of Port E), and the analog to digital converter channel (in the case of the A/D port). For Power and GND, you are spoiled for choice.

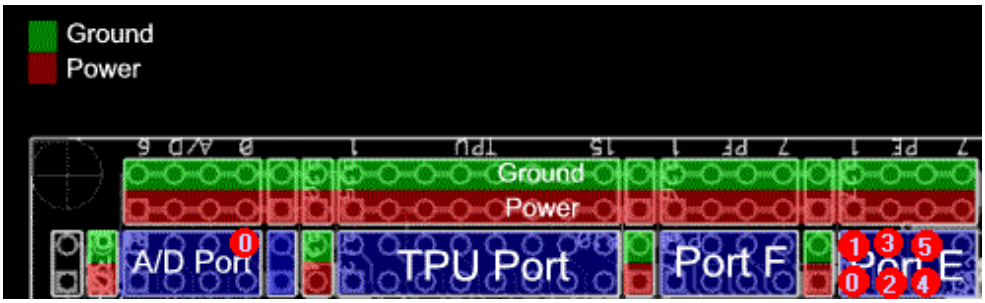

Figure 8. MRM Connections

3. Software

The code featured in this article leverages the Open Source software libraries of the *MotoRobots* project (see the Links section at the end of this article for details). The MotoRobots libraries are an attempt to create a powerful, programmer-friendly collection of C functions (and later C++ classes) to make the MRM (and similar controller boards) more accessible to beginners, and more productive for experienced developers. These libraries are distributed under the GNU LGPL.

The GNU m68k cross-development toolchain, suitable for building the code samples in this article, has been packaged into convenient downloadable installation packages for the MotoRobots project (Windows® and Linux® platforms - Intel x86). These are freely available from the downloads page of the MotoRobots project web site.

3.1 Target Software

Listing 1 shows the main program. This will run on any MRM variant (32KB, 512KB, 16MHz, 25MHz). After initializing the processor clock speed and serial port BAUD rate, the program initializes the camera I/O ports. The program then enters a loop, looking for characters received from the serial port. The program will respond to certain characters as follows:

- **s** - sets the "shutter" speed (exposure time)
- **p** - take a picture and upload it to the host PC
- **x** - end the program and return control to the monitor

Before capturing the image, the **gbcam_shoot( )** function initializes the M64282FP by writing a set of default values to its registers (please refer to the M64282FP data sheet in the Links section at the end of this article for details of these registers). The default settings will do for most simple image capture applications, but if you want to tweak them, you can alter the values of the members of the structure **gbcam_registers** (not show in Listing 1 - see the source code file **gbcam.h**) before calling **gbcam_exposure( )** or **gbcam_shoot( )**.

```
// camera.c
// gbcam library usage example
//
// Copyright (C) 2001 Dafydd Walters
// dwalters@users.sourceforge.net
//
// Permission is hereby granted, free of charge, to use,
// modify and distribute this example source code without
// restriction.  Please note that such unrestricted
// permission does NOT apply to the MotoRobots libraries
// themselves.  The MotoRobots libraries are distributed
// under the terms of the GNU Lesser General Public License.
//
// THE AUTHORS AND THE CONTRIBUTORS TO THIS WORK
// DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE
//
```

```
#include <68332/qsm.h>
#include <68332/sim.h>
#include <68332/mrm/gbcam.h>

#define BAUD_RATE 19200

int main(void)
{
    char c = 0;
    int i;

    set_processor_speed(CLOCK16MHZ);
    sciinit(BAUD_RATE);
    gbcam_init();

    while (c != 'x')
    {
        if (havebyte())
        {
            // Receive byte from serial port
            c = inbyte();

            switch (c)
            {
                case 's' :  // Set exposure time

                    gbcam_exposure();
                    outbyte('s');
                    break;

                case 'p' :  // Take a picture

                    gbcam_shoot();
                    for (i = 0; i < gbcam_IMAGE_SIZE; i++)
                        outbyte(gbcam_imageBuffer[i]);
                    break;
            }
        }
    }

    // return control to CPU32BUG monitor
    return 0;
}
```

Listing 1. Target Program

The full target source code is packaged here for download in *zip* format for Windows®, and *tar.gz* format for Linux®. The packages include the source code in Listing 1, and the dependent MotoRobots libraries. The compiled binary program in S19 format is also provided, ready to load into the MRM.

Downloads

- Target binary - MRM (.s19)
- Target source code - Windows (.zip)
- Target source code - Linux (.tar.gz)

The MotoRobots libraries contain ready-made linker scripts for the different MRM configurations (these are provided in the **startup/68332/mrm** directory). The target code provided here is linked with **mrm_rom_32k.ld**, which is the script for the *32KB* version of the MRM that locates the program code in *Flash ROM*. The 32KB script also works with the 512KB MRM. However, if you have the expanded MRM and you are planning on

expanding on the example code provided here, you may wish to edit the Makefile (in **examples/68332/mrm/gbcam**) to use **mrm_rom_512k.ld** and rebuild (using **make**).

If you have the 25MHz version of the MRM, you may wish to edit **camera.c** and change the line **set_processor_speed(CLOCK16MHZ);** to read **set_processor_speed(CLOCK25MHZ);** and then rebuild.

The MotoRobots libraries are being constantly developed, so by the time you read this, the version included here could well be out of date. Please check the MotoRobots web site for the latest version. However, the code provided here has been well tested, and is known to work, so before downloading any later versions, I would recommend that you eliminate any hardware or wiring bugs using the code provided with this article first.

3.2 Host Software

Listing 2 is the program for the host PC. This is written in Java™ and just requires a Java™ 2 runtime environment (version 1.2 or later) and an implementation of the Java™ COMM API to run. I've successfully tested it using the following configurations:

Platform	SDK	Java™ COMM API Implementation
Windows® 2000 Professional	Java™ 2 SDK Standard Edition version 1.3.1	Java™ Communications for Windows® version 2.0
Red Hat® Linux® 7.2	Java™ 2 SDK Standard Edition version 1.3.1	RXTX version 1.4.13

Downloads

Both the source code and compiled binaries of the Camera Image Viewer host software are included in this **jar** archive file.

- Host software - Platform-independent (.jar)

To extract the source code files from the archive, you can use

```
jar xf GBCamView.jar GBCamView.java
```

The file is also an executable jar, meaning that you can run the program by simply invoking

```
java -jar GBCamView.jar
```

```
/*
 * GBCamView.java
 * gbcam library image viewer example for host PC
 *
 * Copyright (C) 2001 Dafydd Walters
 * dwalters@users.sourceforge.net
 *
 * Permission is hereby granted, free of charge, to use,
 * modify and distribute this example source code without
 * restriction.  Please note that such unrestricted
 * permission does NOT apply to the MotoRobots libraries
 * themselves.  The MotoRobots libraries are distributed
 * under the terms of the GNU Lesser General Public License.
 *
```

```
 * THE AUTHORS AND THE CONTRIBUTORS TO THIS WORK
 * DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE
 */

import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.awt.event.*;
import javax.comm.*;
import java.util.*;

public class GBCamView extends JFrame {

  JPanel buttonPanel = new JPanel();
  JButton exposureButton = new JButton();
  JButton captureButton = new JButton();
  ImagePanel imagePanel = new ImagePanel();
  String serialPortName = null;
  SerialPort port = null;
  InputStream in = null;
  OutputStream out = null;


  public static void main(String[] args) {
    GBCamView viewer = new GBCamView();
    viewer.pack();
    viewer.show();
  }


  public GBCamView() {
    super("Camera Image Viewer");
    try {
      initGUI();
    }
    catch(Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
    loadConfig();
    promptSerialPort();
    setupPort();
  }


  private void promptSerialPort() {

    Vector v = new Vector();

    // Obtain a list of serial ports
    Enumeration en = CommPortIdentifier.getPortIdentifiers();
    while(en.hasMoreElements()) {
      CommPortIdentifier cpi = (CommPortIdentifier) en.nextElement();
      if (cpi.getPortType() == CommPortIdentifier.PORT_SERIAL)
        v.addElement(cpi.getName());
    }

    if (v.size() == 0) {
      // No serial ports found
      JOptionPane.showMessageDialog(this,
        "No serial ports found", "Error", JOptionPane.ERROR_MESSAGE);
      System.exit(1);
    }

    serialPortName = (String) JOptionPane.showInputDialog(this,
      "Please confirm serial port selection", "Camera Image Viewer",
```

```
    JOptionPane.QUESTION_MESSAGE, null, v.toArray(), serialPortName);

  if (serialPortName == null)
    System.exit(1);
  else
    saveConfig();
}


private void setupPort() {

  try {

    CommPortIdentifier cpi = CommPortIdentifier.getPortIdentifier(
      serialPortName);

    port = (SerialPort) cpi.open("Camera Image Viewer", 3000);
    port.setSerialPortParams(19200,
      SerialPort.DATABITS_8,
      SerialPort.STOPBITS_1,
      SerialPort.PARITY_NONE);
    port.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
    port.disableReceiveThreshold();
    port.disableReceiveTimeout();
    in = port.getInputStream();
    out = port.getOutputStream();
  } catch (Exception e) {
    JOptionPane.showMessageDialog(this,
      "Unable to setup serial port: " + e, "Error",
      JOptionPane.ERROR_MESSAGE);
    System.exit(1);
  }
}


private void loadConfig() {
  Properties p = new Properties();
  try {
    p.load(new FileInputStream(System.getProperty("user.home") +
            System.getProperty("file.separator") + ".gbcamview.properties"));
    serialPortName =  p.getProperty("gbcamview.port");
  } catch (Exception ex) {
    serialPortName = null;
  }
}


private void saveConfig() {
  Properties p = new Properties();
  p.setProperty("gbcamview.port", serialPortName);
  try {
    p.store(new FileOutputStream(System.getProperty("user.home") +
            System.getProperty("file.separator") +
            ".gbcamview.properties"), "Camera Viewer Settings");
  } catch (Exception ex) {
    JOptionPane.showMessageDialog(this,
      "Unable to save settings", "Error", JOptionPane.ERROR_MESSAGE);
  }
}


private void initGUI() throws Exception {
  exposureButton.setText("Set Exposure");
  exposureButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      exposureButton_actionPerformed(e);
```

```
    }
  });
  captureButton.setText("Capture");
  captureButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      captureButton_actionPerformed(e);
    }
  });
  imagePanel.setBorder(BorderFactory.createEtchedBorder());
  imagePanel.setMinimumSize(new Dimension(266, 256));
  imagePanel.setPreferredSize(new Dimension(266, 256));
  this.addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(WindowEvent e) {
      this_windowClosing(e);
    }
  });
  buttonPanel.add(captureButton, null);
  this.getContentPane().add(buttonPanel, BorderLayout.SOUTH);
  buttonPanel.add(exposureButton, null);
  this.getContentPane().add(imagePanel, BorderLayout.CENTER);
}


void exposureButton_actionPerformed(ActionEvent e) {
  try {
    // make sure input buffer is empty
    while (in.available() > 0)
      in.read();
    // transmit 's' command character to set exposure
    out.write('s');
    // wait for acknowledgement character
    in.read();
  } catch (IOException ex) {
    System.out.println("Exception occurred: " + ex);
  }
}


void captureButton_actionPerformed(ActionEvent e) {

  try {
    // make sure input buffer is empty
    while (in.available() > 0)
      in.read();
    // transmit 'p' command character to take a picture
    out.write('p');
    for (int y = 0; y < 123; y++) {
      for (int x = 0; x < 128; x++) {
        int pixel = in.read();
        if (pixel < 0 || pixel > 255)
          throw new Exception("Illegal pixel value");
        imagePanel.pixels[x][y] = pixel;
      }
    }
  } catch (Exception ex) {
    System.out.println("Exception occurred: " + ex);
  }
  // paint the image
  imagePanel.repaint();
}


void this_windowClosing(WindowEvent e) {
  // Window is closing, so end the application
  System.exit(0);
}
```

```
}


// ImagePanel is a class that encapsulates the display of the image
// using an array of pixel intensities.
class ImagePanel extends JPanel {

  public int[][] pixels = new int[128][123];

  protected void paintComponent(Graphics g) {
    g.setColor(getBackground());
    g.fillRect(0, 0, getWidth(), getHeight());
    int ox = (getWidth() - 256) / 2;
    int oy = (getHeight() - 246) / 2;
    for (int y = 0; y < 123; y++) {
      for (int x = 0; x < 128; x++) {
        int pixel = pixels[x][y];
        g.setColor(new Color(pixel,pixel,pixel));
        g.fillRect(x*2 + ox, y*2 + oy, 2, 2);
      }
    }
  }
}
```

Listing 2. Host Program

Here are some instructions for installing the Java™ Communications API on Windows® and Linux® platforms.

Java™ COMM API Installation

Windows®

This assumes you are using Java™ 2 SDK 1.2 or later. In the steps listed below, **<jdk>** refers to the root directory of your Java™ SDK installation, so if you installed Java™ SDK in **c:\jdk1.3.1_02** replace all reference to **<jdk>** with **c:\jdk1.3.1_02**

- Download the Java™ Communications zip file, and extract to an empty directory
- Place the file **win32com.dll** in the **<jdk>\jre\bin** directory
- Place the file **comm.jar** in the **<jdk>\jre\lib\ext** directory
- Place the file **javax.comm.properties** in the **<jdk>\jre\lib** directory
- Do *not* alter the CLASSPATH environment variable
- To test the installation, open a command prompt window, change to the **commapi\samples\BlackBox** directory and run

```
java -cp BlackBox.jar BlackBox
```

Linux®

I must be honest; I had great difficulty in getting Java Communications working in Linux. After searching the web, the only implementations I found were IBM's COMM API implementation for their own Java runtime environment, and an open source package called RXTX. RXTX looks like it has great potential, but I had problems getting versions 1.4.15 and 1.5.8 working properly. I did, however, manage to get 1.4.13 working, but only when logged in as root. Note that versions 1.4.x and 1.5.x are fundamentally different from each other, in that 1.4.x requires you to obtain **comm.jar** from Sun's Java™ Communications for *Solaris*, but the 1.5.x version of RXTX theoretically contains everything needed for COMM API. This means that the installation techniques for these two versions are quite different, but as 1.4.13 is the only version I was able to successfully install on my Red Hat Linux 7.2 box, that is the one I will describe here.

This assumes you are using Java™ 2 SDK 1.2 or later. In the steps listed below, **<jdk>** refers to the root directory of your Java™ SDK installation, so if you installed Java™ SDK in **/usr/java/jdk1.3.1_02** replace all reference to **<jdk>** with **/usr/java/jdk1.3.1_02**

- Download this i386 binary package: rxtx-bin-1.4.13.i386.tar.gz
- Extract the files from the archive with

  ```
  tar -zxf rxtx-bin-1.4.13.i386.tar.gz
  ```

  This will create a subdirectory *rxtx-bin-1.4.13.i386* containing the extracted files.
- Download Sun's Java™ Communications for *Solaris*
- Extract the files from the archive with

  ```
  tar -zxf <archive-name>.tar.Z
  ```

  This will create a subdirectory *commapi* containing the extracted files.

Log in as *root* to perform the remaining steps, *and* thereafter to run any COMM API programs (including GBCamView and BlackBox).

- Place all of the **libSerial*** and **libParallel*** files from RXTX in the **<jdk>/jre/lib/i386** directory
- Place the file **jcl.jar** from RXTX in the **<jdk>/jre/lib/ext** directory
- Place the file **javax.comm.properties** from RXTX in the **<jdk>/jre/lib** directory
- Place the file **comm.jar** from *commapi* in the **<jdk>/jre/lib/ext** directory
- Do *not* alter the CLASSPATH environment variable
- To test the installation, open a command prompt window, change to the **commapi/samples/BlackBox** directory and run (as root):

  ```
  java -cp BlackBox.jar BlackBox
  ```

If you are feeling adventurous, you can build RXTX version 1.4.13 from the source code. Here is the full package: rxtx-1.4-13.tar.gz.

If you're feeling *really* adventurous and have plenty of time to waste, you could try to get one of the later versions of RXTX working. At the time of this writing, 1.5 was still experimental.

One final point. The serial port devices on some Linux distros (e.g. Red Hat 7.0 - 7.2) are owned by *root* and have permissions set so that ordinary users cannot open them. This is easy enough to change. To allow all users to access COM1 for example, enter the following (as *root*):

```
chmod 666 /dev/ttyS0
```

and for COM2:

```
chmod 666 /dev/ttyS1
```

Since you will be running the COMM API code as *root* anyway, this is probably moot, however.

Troubleshooting

- If you see an error such as this: "Exception in thread 'main' java.lang.NoClassDefFoundError: javax/comm/CommPort", this probably means that the COMM API files have not been installed in the correct directories. Carefully double-check the file locations described above. Beware that at the time of this writing, the documentation that comes with Java™ Communications is a little out of date, and describes the installation steps and file locations for java 1.1 (which are different to Java™ 2 SDK 1.2 and above, which I have described above).

Here are the steps for running the host PC program. This assumes you have downloaded the host program, installed Java™ and Java™ Communications, and tested that it works by running the BlackBox example program bundled with Java™ Communications.

Running the Host Program

Windows®

- From *Explorer*, or *My Computer*, double-click on the icon for the host program file (**GBCamView.jar**) you downloaded

Linux®

- Open a command prompt window
- Change to the directory where you downloaded the host program (GBCamView.jar)
- Run the host program as an executable jar file:

```
java –jar GBCamView.jar
```

With the MRM hooked up to the host PC via the serial port, point the camera at a subject, then click the *Set Exposure* button on the GUI. By metering the brightness and contrast of the scene, this will automatically set the exposure time. This takes around 15 seconds. Try to keep the camera and subject fairly stationary during this exposure-setting time, as the algorithm takes multiple exposures. After setting the exposure time, clicking the *Capture* button will then take a picture, download it to the PC, and display it. The download takes about 10 seconds. It is not necessary to reset the exposure time between successive image captures, provided the lighting conditions remain similar.

Troubleshooting

- You may have problems running the host program if there is more than one version of Java™ installed on your machine, and the version for which you installed COMM API is not the version you are using to run the host program. A symptom of incorrectly configured COMM API is an error issued by the host program as soon as it is launched. You may be surprised to find multiple versions of the Java™ runtime installed on your system (e.g. IE or Netscape may have a Java™ plug-in). If in doubt, search for the **java** executable on your hard drive (**java.exe** on Windows® systems) - there are typically one or two instances of java.exe for each version of the Java™ SDK or JRE. If you find multiple versions, you can follow the instructions above to install COMM API on the others as required.
- To make sure that the MRM is responding to commands from the host PC via the serial port, and that the correct serial port is selected, try opening a terminal session at 19200 BAUD (e.g. using HyperTerminal or Kermit), and issue the 's' and 'p' commands manually. You should see a single 's' echoed in response to the 's' command after a pause of about 15 seconds, and a large stream of seemingly random characters in response to the 'p' command.

3.3 Image Processing Applications

Capturing an image is only the first step in robot vision. For a robot to do something useful with the image, some processing must be done. The details are beyond the scope of this article, but here are some pointers

Statistical Analysis
To see an example of how statistical analysis can be used to identify the lines of a line-maze, have a look at the article "*Line Detection Using a Digital Camera*" in the Links section at the end of this article.

Stereo Vision
Although there is no reason why you couldn't connect two GAME BOY® cameras to the MRM, the MRM is not powerful enough to do *stereopsis* (extracting depth information from two or more images of the same scene) in real time. Something that certainly *is* feasible though, and that I am working on in fact, is building a simple stereo vision head using an MRM, two image sensors from GAME BOY® cameras and a pair hobby servos (for pan and tilt). This connects to a PC via a full-speed USB interface (the USB interface has to be implemented using external components - the MRM has no inbuilt USB port). The MRM then acts as a slave to the PC, servoing the head, and capturing genlocked stereo images. The PC (which is much more powerful than the MRM) applies a stereopsis algorithm on the captured images in real time to extract depth information from the scene. When (or if) this work ever gets to a point where I have some meaningful code or circuit designs to share, I certainly intend to make it public.

Intensity Profile
The technique that my mini-sumo robot uses to detect its opponent is to build a histogram of the sum of pixel intensities for each column, resulting in an array of 256 integer values, and look for clusters of sudden changes in intensity value next to each other. Since the background is so sterile in the mini-sumo arena, such changes tend to occur where there is an object. This is fairly primitive, and far from perfect, but since the technique is so simple, it works in real time (several times per second).

Open Computer Vision Library
If you are feeling adventurous, you might want to take a look at the code in the *Open Computer Vision* software library (see the Links section at the end of this article), and port some of this code to the MRM. This is an Open Source project, started by Intel and hosted by SourceForge. Their philosophy is stated thus:

> *Aid commercial uses of computer vision in human-computer interface, robotics, monitoring, biometrics and security by providing a free and open infrastructure where the distributed efforts of the vision community can be consolidated and performance optimized.*

4. Other CMOS Image Sensors

A quick search of the web will reveal that there are many places you can obtain CMOS camera modules at fairly affordable prices (although admittedly, not at the prices you can get used GAME BOY® Cameras at currently). A nice source of both color and monochrome devices I found based on OmniVision CMOS Image Sensors was from Amazon Electronics. They sell color camera modules with full digital interface and a choice of lens for under $60.

5. Links

- RoboMinds - home of the Mini RoboMind robot controller
- Mitsubishi M64282 Image Sensor Data Sheet (.pdf)
- MotoRobots Software Libraries for the MRM
- Line Detection Using a Digital Camera by Doug Kelley
- Three 68332 Designs Presented by Kenneth Maxon

- Open Computer Vision Software Library
- Author's website

6. Acknowledgements

Many of the photographs and illustrations in this article are courtesy of Daniel Herrington and Benjamin Green. Used with permission. Many thanks to both of them.

7. Bugs in this article

Please don't hesitate to email me and let me know if you've found any errors or omissions in this article.

---

dafydd@walters.net