Cognachrome Vision System User's Guide

Manual Edition 2.0
Documents software version 26.0
June 3, 1996

Anne Wright, Randy Sargent, and Carl Witty
Jeremy Brown

Newton Research Labs
Evergreen Park
16504 NE 37th St. #X2073
Redmond, WA 98052
E-mail: vision@newtonlabs.com
http://www.newtonlabs.com

1 Introduction

The Cognachrome 2000 color vision system is a small (2.5"x6.25"x1.25") video-processing computer system with special hardware to track multiple objects in the visual field (discriminated by color) at a full 60 Hz frame rate, with a resolution of 200x250 pixels. The board is also capable of low-resolution 24-bit RGB frame grabs at 64x48 pixel resolution. Its input is standard NTSC, allowing much flexibility in video input source, as well as making it very simple to see the real-time video image (just hook up a standard NTSC monitor). The board also outputs a 60 Hz NTSC image of the objects it is tracking, to help in debugging.

The Cognachrome system can output tracking data to another computer, or can be used stand-alone, using spare I/O and CPU resources to directly control a robot base or similar hardware.

1.1 Modes of Use

The Cognachrome 2000 system allows for several modes of use:

- Using our pre-programmed algorithms, the system can be set up to output tracking data over serial ports to another computer. Currently two software packages are available—*stand alone* and *Pioneer*. If you are using the system bundled with the Pioneer robot, you probably want to use the Pioneer software; see Chapter 3 [Pioneer Quick Start Guide], page 7. Otherwise, you probably want to use the stand alone software; see Chapter 2 [Stand Alone Quick Start Guide], page 3.
- The vision board itself may be programmed in C to directly control a robot or various actuators. Your controlling code can run as a completely separate process from the vision tracking, or be called after each frame. You can choose whether to treat the data as a frame buffer, thresholded segments, or processed blobs. The 68332-based computer board has free I/O ports which may be used for interfacing to other devices. These ports include digital I/O lines, a bus with software-definable chip selects, and two asynchronous and one synchronous serial port.
- You may compile your own processing algorithms and place them on the board, and make this data available either to a remote machine or to other processes operating on the board.

At this time, this manual only documents the use of the system with the preprogrammed algorithms. If you want to write custom software for the vision sysem, you may want to purchase the ARC C environment, also available from Newton Labs. This environment allows compilation and interactive debugging of programs on the board.

Programs are compiled using the GNU C cross-compiler (freely available for FTP from '`prep.ai.mit.edu`'), well-known for its stability and excellent optimization. The ARC environment provides a multi-tasking kernel and libraries for the 68332 board, plus the ARC interaction program which runs on a host machine and allows you to easily download and debug programs running on the 68332 board by communicating over a serial port. See '`http://www.newtonlabs.com/arc`' for more information about the ARC Development System.

Once you have purchased ARC, you can write your own programs and interface them to vision libraries which provide the functionality described in this manual. Contact Newton Labs if you wish to pursue this option (see Appendix D [Reaching Newton Research Labs], page 105).

1.2 Components of the System

This system is built of two small, stacking printed circuit boards, measuring a total of only 2.5"x6.25"x1.25" when stacked. The top board is the color processing board, which digitizes incoming NTSC into 24 bits of RGB, and then sends this RGB through a lookup table. The board can be trained to recognize up to three different colors at once. Objects are tracked or recognized using the color lookup table. Multiple objects of a given color may be tracked at once. Tracking runs at full frame rate (60 Hz) for sufficiently simple processing. More complex operations, such as orientation calculations, may reduce the update rate somewhat. We use a folding scheme to reduce the 16 million table entries required for 24-bit RGB, while still keeping the full 24 bits of color resolution.

The bottom board is the processor board. It is a standard Motorola 68332-based computer, with 256K of RAM (expandable to 1MB on board). The 68332 is unique in its inclusion of a microcodable timer coprocessor, which is heavily used in processing thresholded images. The board has I/O ports left over after the interface to the vision processing board. This includes digital I/O lines, a bus with software-definable chip selects, and two asynchronous and one synchronous serial ports. It is therefore possible for a user to directly control a robot or other actuators from the board, or to interface to a network of other computers.

2 Stand Alone Quick Start Guide

The stand alone quick start guide is intended to help you get your board installed and operational using the stand alone mode software. It assumes that you will be communicating with the board over a serial port. Generally, you will want to use the ARC interaction program to configure and download new software to the vision system. Once the system is configured, it will automatically start transmitting tracking information on startup. You will probably want to write your own program to process this tracking data and use it in a control task. This chapter contains the information needed for you to start using the vision system in this manner.

If your cognachrome vision system came bundled with a Pioneer robot, the interaction will be different and this chapter will not be relevant to you. See Chapter 3 [Pioneer Quick Start Guide], page 7, for information specific to that system.

After you're up and running, you will probably want to read the other sections of this manual, which explain alternate hardware setups, and give more detail on how to configure and operate the software.

2.1 Hooking Up the Board and Cables

Identify the boards

Your vision system consists of two boards (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). Notice now which board is the vision board, and which is the processor board. Normally these boards come stacked with the vision board on top, and the processor board on the bottom. If they are already stacked, you do not need to do anything further for this step.

If the boards are not already stacked, you can recognize which is which in the following manner:

- The vision board has two video connectors and many LEDs at each end.
- The processor board has four large rectangular chips (the memory), and one large square chip (the processor).

Stack the boards together with the vision board on top, and the processor board on the bottom. Verify the orientation is correct by seeing that the mounting holes on the four corners of each board line up. Press the boards together firmly.

Connect power to the vision system

You should see two molex connectors (plasic housings with wires going to them) coming from your power supply. They connect to opposite ends of the top board. The board can potentially be damaged by applying power to one side but not the other, so we recommend connecting these while the power supply is turned off or unplugged. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

Plug the board in now and confirm that all three red LED's light. If none light, you may want to check that the power supply is plugged into an active outlet, then confirm the voltages at the molex connectors with a voltmeter. If the red +5V Digital Power LED does not light, but the other two red LED's do, you may have a blown fuse. If the red +5V Analog Power LED does not light, you may have a bad analog power connector or a bad 5V power regulator. Check that the +5V Analog Power pin of the 6-pin connector has 5V going into it. See Section B.1 [Status Lights and Beeper], page 85, for more general troubleshooting information.

Connect the serial cable

Connect a serial cable from your host computer to Port A on the vision system. This is one of the two 3-pin male molex connectors on the processor board. Port A is on the right, and port B is on the left. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

See Section B.3 [Serial Port Troubleshooting], page 87, for more in-depth information on the topic of serial ports.

Connect the camera

Connect an NTSC color camera to the white RCA jack (the kind of coaxial connector common on modern VCR's and TV's). If you look closely you will see a small "IN" label next to this jack on the PC board. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

Connect a video monitor

It is not necessary to hook up a video monitor for the system to work, but it is recommended for initially testing the system, and for training the system to track new colors. The vision board outputs standard black and white video—a color monitor is not required. You can use any NTSC (North American video standard) video monitor with an RCA *Video In* connector. This includes most modern TV's, VCR's, and camcorders.

Once you have located a suitable monitor and RCA video cable, connect the debugging video output from the board (red RCA jack) to the monitor. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

2.2 Using ARC with a Stand Alone Vision System

ARC is a serial interaction program which allows you to use a host computer to communicate with the vision system. If you have purchased the full ARC programming environment, you can write software that executes on the vision system processor itself. Even if you have not purchased the full ARC development system, purchase of a vision system entitles you to a *Mini-ARC* license.

Mini-ARC allows you to interact with the vision system for configuration and allows you to download new software releases, but does not include the libraries or the full compilation and interactive debugging environment. Mini-ARC is sufficient to do everything described in this section. See Appendix A [Mini-ARC Installation Guide], page 79, for installation instructions. In the rest of this manual, ARC is used as the generic term to refer to the serial interaction program in either a full ARC Development System installation, or a Mini-ARC installation.

Before starting ARC, you'll need to know the name of the serial port you intend to use. If you do not know which port to use, see Section B.3.1 [Serial Port Names], page 87.

- First, start ARC to talk to the appropriate serial port, as discussed above.

  ```
  arc -port portname
  ```

  Note: The stand alone vision software defaults to having the serial port run at 38400 baud. This is the default baud rate for ARC, so you should not need any other flags.
- Next, turn on the power supply for your vision board, the camera, and your debugging monitor, if you have not already done so.
- Press and release the vision system's RESET lever (corner of the top board). See Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

You should hear a beep from the board, and see a boot message from the board on the ARC window.

If you do not see a boot message from the board, don't be dismayed—usually it is just a problem with the configuration of the serial port, or a simple wiring problem. If any of the three red LED's didn't come on, you have some sort of power problem. If you heard a beep but didn't see anything on the ARC screen, you probably have a problem with the serial connection or gave the wrong serial port name to ARC. See Appendix B [Troubleshooting], page 85, and feel free to call us for help if you can't find your problem quickly (see Appendix D [Reaching Newton Research Labs], page 105).

The board will immediately enter *Tracking Upload Mode* (see Section 6.2 [Tracking Upload Mode], page 49). If the camera is connected, the board will stay in this mode and immediately start uploading tracking information. If the camera is not connected or not working, the system will print '`Aborting vision`' and enter TPUVIS Prompt Mode (see Section 6.3 [TPUVIS Prompt Mode], page 50).

ARC will bring up two windows, one with the title "Stream #3" for TPUVIS interaction, and one with the title "Stream #28" which will display the tracking data.

2.3 Trying it Out

Your system should arrive with tracking channel A pre-configured to track a bright orange test beacon.

Please note that even though the system has been pre-trained to track the orange beacon, you'll definitely want to re-train the system to your camera and lighting conditions before doing any serious work (see Section 6.3.4.2 [Training and Configuring Channels Detail], page 52). Hopefully, though, the pre-configured training will be enough to get a reasonable response from your system while you try it out.

To see how well the color thresholding is working in your environment, look at the debugging monitor you hooked up to the board. It shows, in real-time, which regions of the video image match the programmed lookup table for channel A.

If the beacon shows up only spottily, it's most likely that you need to retrain the system (see Section 5.2 [Color Training], page 27), but you can deal with that later. Also make sure that the lighting conditions around the beacon are reasonable (see Section 5.1.3 [Choosing Lighting], page 26), and visually double-check the image coming out of the camera by connecting it directly to a monitor (see Section 2.1 [Hooking Up the Board and Cables], page 3). Depending on the camera, you might want to focus or adjust the iris.

The data being displayed over the serial port should give the coordinates of the centroid of the largest orange object detected. Move the beacon and observe how the data changes.

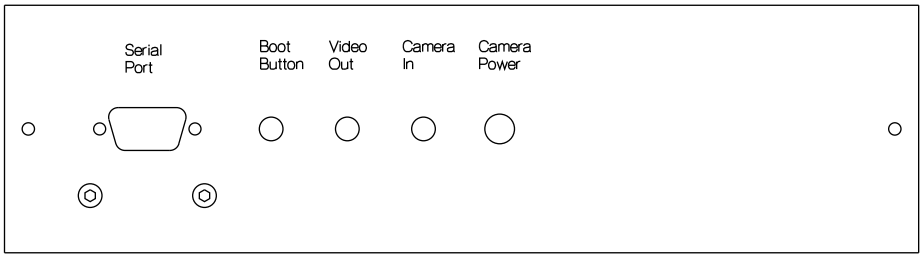
Congratulations! You've gotten your system up and running. Please read the other sections of the manual to learn how to configure the system to track multiple colors or several objects of the same color, measure size and orientation of objects, and do frame grabbing and processing.

3 Pioneer Quick Start Guide

The Pioneer quick start guide is intended to help you get started using the vision system hardware and software bundled with the Pioneer robot. This outlines the hardware and software needed to get you started.

3.1 Pioneer Vision Interface Panel Diagram

The following is a diagram of the vision interface panel. This panel allows you to connect to the vision system inside the Pioneer robot. If you thought that your robot has a vision system, but it is not equipped with such a panel, stop now and contact ActivMedia. The rest of this manual assumes that your system is equipped with such a panel.



3.2 Setting Up the Pioneer Vision Hardware

These instructions assume that you have a Pioneer robot with a vision system, camera, and vision interface panel already properly installed and hooked up. If this is not the case, see Appendix C [Pioneer Vision System Hardware Installation], page 101, first, then come back to this section when you are done.

Connect the camera

The camera has two connectors: an RCA video plug, and a power plug. Connect the power plug to the *Camera Power* connector on the vision interface panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7). Connect the RCA video plug to the *Camera In* connector on the vision interface panel.

Connect the serial

If you are using a radio modem, the robot end of the modem should be hooked to the vision system already. Follow the instructions in the Pioneer manual for how to connect the host end of the radio modem to your computer.

If you are not using a radio modem, you should connect a serial cable from your computer to the 9-pin serial connector on the vision interface panel. This is a standard IBM PC compatible serial connector.

DO NOT use the 9-pin serial connector on the top of the Pioneer. Connecting to this port would cause contention between the vision system's serial port and the serial port on your host computer.

Connect a video monitor

It is not technically necessary to hook up a video monitor for the system to work, but it is recommended for initially testing the system, and for training the system to track new colors. The vision board outputs standard black and white video—a color monitor is not required. You can use any NTSC (North American video standard) video monitor with an RCA *Video In* connector. This includes most modern TV's, VCR's, and camcorders.

Once you have located a suitable monitor and RCA video cable, connect the monitor to the *Video Out* jack on the vision interface panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7).

Turn on the power

The vision system and camera power can be enabled or disabled by using a switch on the top of the Pioneer. You can't turn on the vision system power when the rest of the robot is off, but it is possible to power the rest of the robot, but not the vision system.

Towards the back of the robot from the main power switch are two other switches. Of these two, the switch closer to the center of the robot is the one which controls the vision system power. Be sure that this switch is turned on (the switch is pushed toward the back of the robot) before you try to use the vision system.

It should be the case that if you switch this switch off, then back on, you will hear some beeps and see a change in the video output monitor. If you plan to always use the vision system with the Pioneer, you can go ahead and leave this switch always in the ON position, and use only the main power switch to turn the robot on and off.

Try it out Congratulations! This is the end of the hardware setup instructions. You should now be ready to try out your robot. You should be able to start Saphira as usual, and have access to vision system features as well as the usual features of the robot. See Appendix B [Troubleshooting], page 85, if this does not work. If all else fails, contact ActivMedia. It is important that this step works before you go on.

3.3 Interacting Directly with the Vision System using ARC

In the normal course of events, you will probably be using the Saphira host program to interact with the robot. In this case, the interaction between the host computer and the vision system will be transparent. However, there are reasons that you may need to communicate with the vision board using ARC.

- It may be necessary to upgrade the software in the vision system. This requires using ARC.
- At this time, not all of the possible vision modes are built into Saphira. If you want to use certain modes now, it may be necessary to interface directly to the vision system. See Section 6.4 [Saphira Interaction Mode], page 56, for more information.
- Interacting with the vision board directly allows access to certain diagnostic routines which are unavailable from Saphira. See Section 6.3.3 [TPUVIS Command Summary], page 50, for information on the commands available.

WARNING: Be sure that ARC and Saphira are not both trying to talk to the robot at the same time. This will not work! Exit one before starting the other.

3.3.1 Getting Set Up to Use ARC

ARC is a serial interaction program which allows you to use a host computer to communicate with the vision system. If you have purchased the full ARC programming environment, you can write software that executes on the vision system processor itself. Even if you have not purchased the full ARC development system, purchase of a vision system entitles you to a *Mini-ARC* license.

Mini-ARC allows you to interact with the vision system for configuration and allows you to download new software releases, but does not include the libraries or the full compilation and interactive debugging environment. Mini-ARC is sufficient to do everything described in this section. See Appendix A [Mini-ARC Installation Guide], page 79, for installation instructions. In the rest of this manual, ARC is used as the generic term to refer to the serial interaction program in either a full ARC Development System installation, or a Mini-ARC installation.

Before starting ARC, you'll need to know the name of the serial port you intend to use. If you do not know which port to use, see Section B.3.1 [Serial Port Names], page 87.

3.3.2 Starting ARC for use with the Pioneer Vision System

When using ARC to communicate with the Pioneer vision system, you should invoke it with either the `-pioneer` or the `-saphira` flag, as well as the `-port` *portname* flag to specify which serial port to use (see Section B.3.1 [Serial Port Names], page 87). For example, `arc -port /dev/ttya -pioneer`.

Either flag will set the baud rate to 9600 baud and allow you to download new programs. The difference is that the `-pioneer` will cause the system to exit Saphira mode, whereas the `-saphira` flag will not.

When ARC is invoked with the `-pioneer` flag, you will be prompted to start the system, if not already running, and press RETURN to exit Saphira mode. When you press RETURN, ARC will send a packet to the system to make it exit Saphira mode and enter TPUVIS Prompt mode (see Section 6.3 [TPUVIS Prompt Mode], page 50).

When ARC is invoked with the `-saphira` flag, it will bring up a window for Saphira mode interaction. While the board is in Saphira mode, this window will have a with a `saphira>` prompt. You can type commands in this window to test out various features of the Saphira mode interaction of the vision system. Type `help` in this window for a list of possible commands.

Also, when ARC is invoked with the `-saphira` flag, it will prompt you whether you want to exit Saphira mode when you issue a download or baud rate change command. Generally, you should answer yes to these questions.

3.4 Trying Out your Vision System with ARC

This is a small walkthrough to make sure that your system is properly set up, and to demonstrate some of the features of the Cognachrome Vision System.

First, make sure that ARC is installed correctly (see Section 3.3.1 [Getting Set Up to Use ARC], page 9). Also, you'll need to have some sort of monitor hooked up to the Pioneer's *Video Out* port.

We'll be running the system in Saphira mode (see Section 6.4 [Saphira Interaction Mode], page 56). If you are running the Saphira host program, exit it before starting ARC. Run ARC with the `-saphira` flag, as well as with any other necessary flags (such as the `-port` flag, to specify which serial port to use).

```
arc -saphira -port portname
```

You should get an '`arc>`' prompt in the window where you ran ARC. Also, ARC should bring up a new window with a '`saphira>`' prompt. It may also bring up another window at this time which will be used for TPUVIS Prompt interaction (see Section 6.3 [TPUVIS Prompt Mode], page 50).

Turn on the power for the Pioneer robot and the vision system, or press the *Boot Button* on the vision interface panel (see Section 3.2 [Setting Up the Pioneer Vision Hardware], page 7). You should get a few lines of messages printed to the '`arc>`' window.

Now the system should be ready to go. Type `v` at the '`saphira>`' prompt. (Then press `ENTER` or `RETURN`. You need to do this after every line you type, even though I won't mention it again through the rest of this section.)

```
saphira> v
pioneer 26.0;vision 26.0
```

The exact version numbers you get may be different, but you should get something formatted much like the above. If you don't, here are some things to check:

- Make sure that Saphira is not running, and that no other software is trying to use the serial port you've connected the Pioneer to (see Section B.3.4 [Serial Port Usage Conflicts], page 91).
- Make sure that ARC is set to use the correct serial port, and that the serial port is correctly wired to the Pioneer. If you saw the Pioneer printing messages when you hit RESET, then this is probably set up correctly.

If you have checked these things and it still does not work, see Section B.3.6 [Pioneer-Specific Serial Troubleshooting], page 97.

If the version did print correctly, we're ready to try out some more features.

First, let's try to grab a frame of video.

```
saphira> G0
frame sent
```

If printing the version string worked, but the board did not respond with a `frame sent` response, then your camera is probably not connected or not powered on (see Section 3.2 [Setting Up the Pioneer Vision Hardware], page 7). Check this now.

After about 10 seconds, a window should pop up showing a low-resolution picture of what the robot is seeing. (You may have to move the mouse into the window to see the correct colors. You can hit '`q`' in this window to make it go away.)

You can replace the '`0`' with '`1`' or '`2`' for faster, but lower quality, uploads.

```
saphira> G2
frame sent
```

Now let's check on the video tracking abilities. First, let's try training the system. (This is only an overview of the training process. See Section 5.2 [Color Training], page 27, for more detailed instructions.)

Make sure you have a monitor hooked up to the Pioneer's video out port. Place a large object of some bright, uniform color (we like bright orange) in the center of the Pioneer's field of view (you can use '`G#`' to check this). To train the system on this color, type:

```
saphira> sa
retrained
```

When you type this, the monitor should lose sync for a second, and then display the newly trained tracking output. You should see the silhouette of the object you trained on; if there are other objects of that color or a similar color in the Pioneer's field of view, you will be able to see them also.

If the tracking output looks grainy (black dots inside the object), you can improve the training with the '`j`' command.

```
saphira> ja
incrementally trained
```

After you've entered this command, move the object around slightly while the screen is flickering.

Now you can see the color tracking output on the monitor. See Section 6.4.3 [Saphira Mode Channel Commands], page 58, for more commands to adjust the color training and view the tracking output of the other tracking channels.

Now let's look at the edge detection capabilities.

```
saphira> E0
edge-detected frame sent
```

This will take a few seconds. You should see a window pop up which contains a gray-scale image showing the edges in the Pioneer's field of view.

The vision system has a *visual sonars* mode, in which the distance to obstacles in different directions is estimated using the edge-detection information. To see a display of this information, use the 'D' command:

```
saphira> D
distmap sent
```

You should see a graph showing eight horizontal segments of varying height. Each segment represents the vision system's estimate of the distance to the nearest obstacle in the given direction. To see a running display of this plot,

```
saphira> DB
distmap started
```

Now you can play with this display, putting various obstacles in front of the Pioneer to see how it reacts. Use 'DE' to stop.

```
saphira> DE
distmap stopped
```

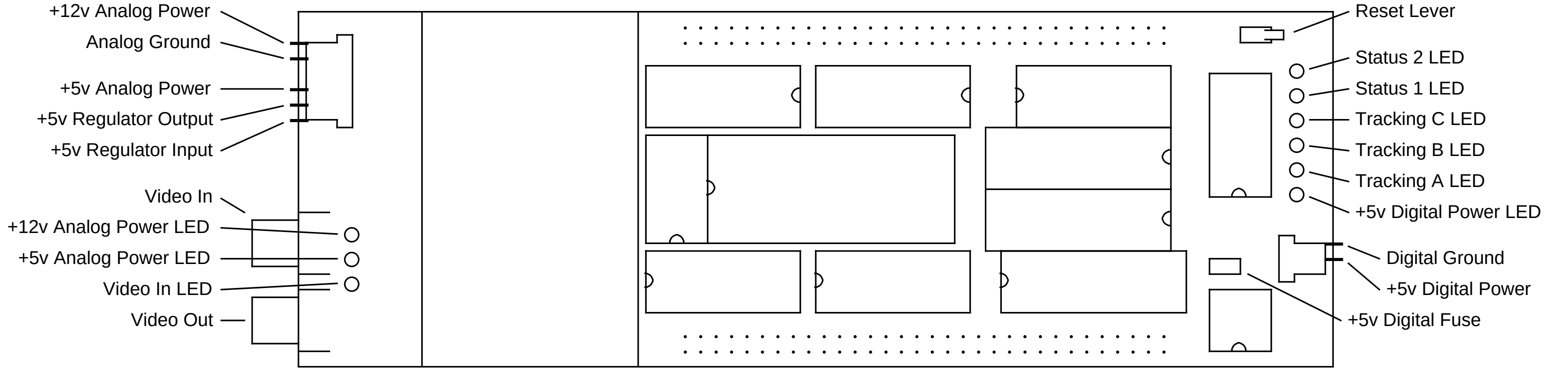
That's it for this walkthrough. If you've gotten to this point, you should be confident that the vision system is working correctly, and that you understand some of its basic capabilities.

4 Vision System Hardware

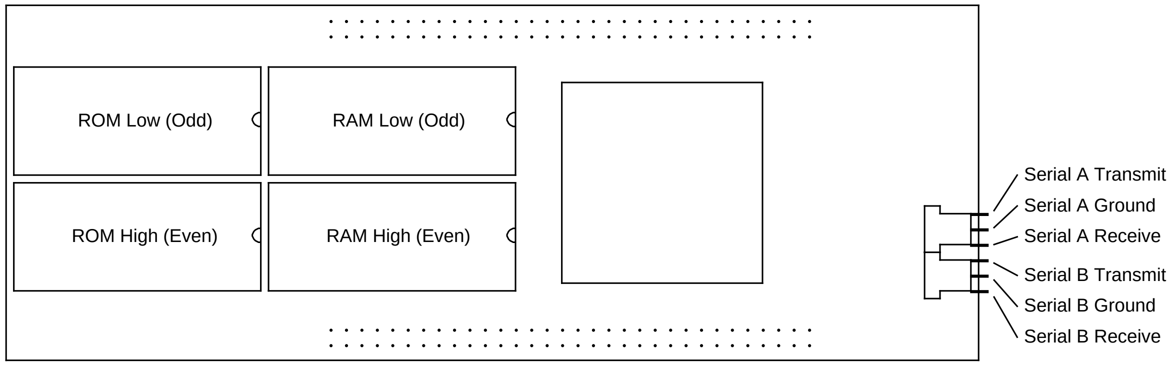
This chapter addresses hardware issues relating to the vision system. This includes a discussion of power, serial, and video I/O. Many sections are more relevant if you are using the system in stand alone mode, rather than as part of a Pioneer robot. See Appendix C [Pioneer Vision System Hardware Installation], page 101, for information more specific to the Pioneer.

If you choose to write your own code to run on the vision system processor, you can control other hardware directly from the vision system, such as motors and additional sensors. This also allows you to gain access to hardware resources which are not required by the vision system. These resources include spare digital I/O pins, timer lines, and a high-speed synchronous serial port. If you wish to do this, please contact us directly (see Appendix D [Reaching Newton Research Labs], page 105).

4.1 Cognachrome Vision System Board Diagram



Cognachrome Vision Board



Vesta 68332 CPU Board

4.2 RS-232 Serial Ports

The Cognachrome vision system has two RS-232 compatible serial ports, labeled "A" and "B". These are the primary means for getting data to and from the vision system.

4.2.1 RS-232 Port Pinouts

The two serial ports are accessible at the two 3-pin molex connectors side by side on the processor board. For location and pinouts of the two serial ports, see Section 4.1 [Cognachrome Vision System Board Diagram], page 16. *Receive* refers to data going *to* the vision system, and *transmit* is data coming *from* the system.

See Section B.3.5 [Serial Port Pinouts], page 96, for information on pinouts of the serial ports of various host machines.

4.2.2 Using the RS-232 Ports

Port A is considered to be the primary port, as it is connected directly to the 68332 processor's hardware asynchronous serial port, and thus takes a minimum of overhead for data transmission and reception. Port A runs reliably at 38400 baud and slower; it may work at higher baud rates but has not been thoroughly tested (please contact us if you wish to use higher baud rates).

Port B is connected to the general-purpose Timer Processor Unit, which runs special microcode to implement asynchronous serial (in addition to the various vision capture tasks it must perform). Thus, while Port B is transmitting or receiving characters, vision system performance may degrade slightly. Port B is only reliable at 9600 baud or slower speeds.

The stand alone mode software (see Chapter 2 [Stand Alone Quick Start Guide], page 3) uses port A by default. The Pioneer software (see Chapter 3 [Pioneer Quick Start Guide], page 7) uses port A to talk to the host computer, and port B to talk to the 6811 board which controls the Pioneer robot. Either port A or port B can be used to interactively tune color recognition, change modes and parameters, and to download new executable programs to your vision system.

4.2.3 RS-232 Port Protocol

In all modes except Saphira interaction mode (see Section 6.4 [Saphira Interaction Mode], page 56), the serial ports use a protocol to multiplex several streams over the same serial line. When using ARC, you do not need to worry about this protocol, as ARC will automatically separate the various streams of interaction into different windows. However, when interfacing to the vision system serial port yourself, you'll need to know how this works.

The protocol is extremely simple and does not support error detection or correction. Each direction on each serial port has an independently maintained *current stream*. When either the board or ARC receive a *stream break* character, they interpret the next character as a *stream number*, and change the current reception stream on that port to be the specified stream number. When the board or ARC want to send a character to a different stream than last time, they send a *stream break* followed by the new *stream number*, followed by the data they want to send to that stream. Note that the input side and the output side are completely independent; a stream switch on one side does not affect the other side at all.

The stream break character is decimal 254 (hexadecimal 0xfe). Stream numbers are in the range of 0 to 31. It is possible to send decimal 254, but it must be escaped by sending two 254's in a row. This is similar to the way backslashes are escaped in C strings, and percent symbols are escaped in `printf` format strings.

For example, in the following sequence, *hi* is sent to stream 5, and *there* is sent to stream 6. Non-printable characters are represented as the hexadecimal value in square braces (i.e., stream break is represented as [0xfe]).

```
[0xfe][0x05]hi[0xfe][0x06]there
```

In the next example the characters [0xf8] to [0xff] are sent over stream 7. Notice how [0xfe] is escaped by sending it twice.

```
[0xfe][0x07][0xf8][0xf9][0xfa][0xfb][0xfc][0xfd][0xfe][0xfe][0xff]
```

In the stand alone software, the TPUVIS prompt interaction is on stream 3, tracking data is sent over stream 28, and frame uploads are sent over stream 30.

4.3 Video I/O

There are two RCA jacks for NTSC video, one for video input, and the other for debugging output. RCA is the coaxial cable standard used by most modern VCR's, TV's, stereo equipment, and camcorders. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16, for the position of these connectors.

4.3.1 Video Input

The white RCA jack on the vision system is labeled *IN* and should be connected to your video source. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16, for the position of this connector. This channel is terminated by a 75 ohm termination within the vision system, so no other termination is needed.

4.3.1.1 Preserving Video Signal Quality

Unfortunately, NTSC is not an ideal standard for transmitting color information. Due to the encoding of hue and saturation on a 3.58 MHz carrier, noise and/or poor signal quality tend to cause problems in hue and saturation much more than in intensity. Here are things you can do to fight signal degradation:

- Use good RCA video cables with 75 ohm impedance. Many RCA cables are intended for audio—don't use these! If you are unsure, use 75 ohm (RG-59) BNC cables with RCA adapters at the ends.

  If you do use BNC cable, use RG-59. Do *not* use RG-58, because it has a 50 ohm impedance. This is a common mistake, since most computer labs have lots of RG-58 lying around (RG-58 has the correct impendance to be used for Ethernet networking). Most BNC cables have RG-59 or RG-58 labeled on their insulation if you look closely enough, so check before you use.
- Keep cables short when possible.
- Understand termination. Your camera should have a 75 ohm output impedance, the video cables should have 75 ohm impedances, and your termination on the other end should have 75 ohm input impedance. Since the vision system has a 75 ohm input impedance, it works as a terminator. So if you simply hook up your camera through a cable to your vision system, this works great.
- If you want to hook up a color video monitor to view the output of the camera directly, be careful of impedance and termination issues. Video monitors normally have 75 ohm input

impedance (for termination). Double termination of a video signal, such as by hooking up both a terminated video monitor and the vision system to the same video signal, can result in signal degradation. This is because the two input impedances act in parallel, which increases the load on the camera and attenuates the signal.

If you can find a video monitor which can be switched to *high-impedance* mode, then you can safely view the video on the monitor at the same time as it is input to the vision system. Switch the monitor into high impedance mode—usually there is a switch labeled *75 ohm* on one side and *Hi-Z* (high impedance) on the other. Then, hook up the monitor *between* the camera and the vision board by using a shielded RCA Y adapter. Hook the Y adapter directly to the input of the monitor—do not use a third piece of cable to connect it. The reason the monitor must be between the camera and vision system, and not have a cable coming from the Y adapter, is to signal reflection (ghosting).

If you do not have a high impedance video monitor (they are fairly rare), do not plug same video signal into both a monitor and the vision system at the same time (if you do, don't complain to us about poor thresholding results). Other video inputs, such as the input to a VCR or camcorder, are usually also 75 ohm terminated, and their use follows the same restrictions as video monitors.

4.3.1.2 Visually Inspecting the Video Signal

We recommend that you connect a color video monitor in place of your vision board, and look closely at the image. Humans are good at filtering out noise, and you may be used to watching broadcast TV and ignoring the fairly noisy picture. Try to turn off your filtering, and look closely for graininess in the color, or things like color fringes.

If the scene is not lit well, the colors may not look very good in the video image. Try to look at the monitor, and forget what color you know things are supposed to be. If objects on the monitor look indistiguishably brownish to you, they will look indistinguishly brownish to the vision system as well. See Section 5.1.3 [Choosing Lighting], page 26.

4.3.1.3 Adjusting the Camera

Auto *White Balance* **Adjustment**

If your camera has optional *white balance* adjustment, turn it off. This is important.

White balance correction tries to adjust the hue of the image in real-time to make it look more *normal*. It turns out that different lighting conditions (incandescent vs.

fluorescent vs. sunlight) have fairly different spectra, so the camera tries to compensate for this. But the camera doesn't know what you're pointing at, so it just ends up adjusting the hue until the different colors balance nicely (to the camera's way of seeing).

Unfortunately, white balance correction interferes with color thresholding. If you are seeking an orange beacon with white balance on, the orange the camera transmits will be significantly different depending on the predominant color of the background. Or worse, if the beacon starts filling much of the field of view (say your robot is trying to approach it), the white balance will notice that the picture is too orange and wash out the orange color.

Iris Adjustment

Although it's no longer very common, some cameras come with lenses that have manual irises. The iris can be set to limit the amount of light coming to the camera, and to increase the depth of focus somewhat.

A manual iris is useful when a camera does not have adequate electronic gain control to maintain proper brightness under different lighting conditions. Since the iris will let in the same fraction of light regardless of light levels, cameras with only a manual iris need to be adjusted whenever the light levels change—they cannot adapt on their own.

When the iris is set to let in too little light, the signal will look very grainy. We have seen setups where the iris was inadvertently set to reduce light by 75%, causing the video image to be grainy, thus causing tracking to be quite poor. Opening up the iris fixed the tracking problems.

We recommend that if you have a manual iris, open it all the way, letting the maximum possible light through. If this causes *blooming* (bright areas start to cause stripes) or *fadeout* (bright colors start to look white), reduce the iris the only enough to get rid of these artifacts.

4.3.2 Video Output

The red RCA jack on the vision system is labeled *OUT* and can be connected to a video monitor. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16, for the position of this connector. The output on this channel is always black and white, so you do not need a color monitor.

It is very important to watch this monitor while training the system to detect new colors. Also, while it is not strictly necessary for tracking operation, it is useful for monitoring the quality of color tracking.

This channel expects a 75 ohm impedance, but it can handle higher or lower impedances.

4.4 Power Connections

Although your vision system was shipped with a power supply, you may need to create your own power supply for it. For example, you may wish to run your system off batteries on a mobile robot base. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16, for the position of the power connectors.

4.4.1 Power Requirements

The Cognachrome system requires the following voltage sources:

- +5 volts digital @ 750ma (approximate)
- +12 volts analog @ 150ma (approximate)
  (By default, +5v analog is regulated down from the +12v analog. The +12v current requirement can be reduced by supplying the +5v analog yourself. Do not use the +5v digital for this though—noise on the analog supplies can ruin color tracking.)

The analog and digital grounds are not directly connected on the vision board; instead, they are coupled with a resistance of approximately 5 ohms. If you can supply separate voltages without a common ground, this will increase noise immunity between the digital and analog portions of the vision system.

4.4.2 Digital Power Connector

For the location and pinout of the digital power connector, see Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

There is a small, socketed fuse between the +5v digital connector and the board's +5v supply. It looks a bit like a resistor. If your power supply works, but you do not see light from the red *+5v digital power* LED (the one next to the digital power connector), check continuity across the fuse.

The most common reason for the fuse to blow is that power and ground are inadvertently shorted by a metallic object touching the male bus interface which protrudes from the top of the vision board. Your system comes with several fuses; if you misplace them or somehow run out, replace with a fast-acting 2 amp fuse (such as DigiKey F828-ND). It is possible to just put a wire in place of the fuse, but this is highly discouraged. It can result in damage to components and board traces. We will not repair any boards which were damaged due to lack of a fuse in this position.

4.4.3 Analog Power Connector

For the location and pinout of the analog power connector, see Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

The analog subsystem has its own +5v regulator, so in normal usage +12v is supplied to both the *+12v Analog Power* and *+5V Regulator Input* pins. Note that the +5v voltage regulator is not automatically connected to the +5v analog line; you must wire the *+5V Regulator Output* pin back to the *+5v Analog Power* pin yourself. (See the cables that originally came with your system for an example.)

If you wish to supply separate analog +5v and +12v signals (possibly to reduce the +12v current requirement), you do not need to connect to the *+5V Regulator Input* or *+5V Regultor Output* pins at all. Just connect directly to the *+12v Analog Power*, *+5v Analog Power*, and *Analog Ground* pins.

4.5 Digital I/O and Synchronous Serial Connection

At this time, interfacing to these resources requires writing custom programs for the vision system. If you wish to do this, please contact us directly (see Appendix D [Reaching Newton Research Labs], page 105).

4.6 IR Remote Control Input

Some configurations of the vision system include an IR sensor. With this feature, the system can be controlled using a Sony TV or VCR remote control.

The system beeps whenever it receives an IR remote command. Each such command is translated into a character, which the vision system processes as if it had been received over the serial port in

TPUVIS mode. See Section 6.3 [TPUVIS Prompt Mode], page 50, for more information on these commands.

The current translation table is given below; remote control equivalents are also given in the documentation for individual commands.

`play` — `r` Start tracking.

`stop` — `q` Exits various submodes, such as tracking mode.

`0` — `s` Retrain.

`1`...`9` — Reserved for user extensions

`enter` — `j` Incrementally adjust the color training.

`record` — `S` Save the current color training and settings to EEPROM.

`rewind` — `R` Load color training and settings from EEPROM.

`volume up` — `+` Crosshairs on.

`volume down` — `-` Crosshairs off.

`channel up` — `>` Grow color training.

`channel down` — `<` Shrink color training.

5 Object Tracking

By default, when the Cognachrome Vision System starts operation, it goes into object tracking mode. In this mode, the incoming video stream is digitized to a resolution of 200 by 250, and each pixel is checked for membership in the three independent color look-up tables ("LUT" for short). Connected regions of recognized pixels are merged, and various statistics about the regions can be calculated:

- Centroid
- Area
- Orientation of major axis
- Relative sizes of major and minor axes

When running the Pioneer software, Saphira configures the tracking mode for each channel by setting persistent variables (see Section 6.4 [Saphira Interaction Mode], page 56), and the data is uploaded in the form of Saphira packets.

When running the stand alone software, the protocol string provides a very flexible way to configure what data is calculated and transmitted by the board (see Section 5.3 [Protocol String], page 31).

The protocol string, the color lookup tables, and other configuration information are all stored in the configuration EEPROM.

5.1 Choosing Tracking Targets

What you'll read in the following sections are suggestions for how to pick targets to optimize tracking. But don't take all the suggestions as hard-and-fast rules: use them as guidelines to help you optimize your tracking setup. We suggest experimentation to test which suggestions you can get away with ignoring in any given setup.

5.1.1 Choosing Colors

When deciding what colors to use when marking objects for tracking, the main issue is finding colors which are different from those typically in your environment. We find that highly saturated, "neon" colors often work well. When we set up for tracking in a new environment, we will try several different fluorescent colors and see which one works best for the background and lighting conditions.

5.1.2 Use Matte not Glossy

When possible, use targets with matte rather than glossy surfaces. Glossy materials are much more affected by the angle the target's surface presents relative to the light source and camera. For example, if you look at a ball that's glossy, you will see reflections of lights, and in bad cases, even be able to see reflections of the surroundings. The big problem is that when glossy materials reflect, they usually reflect the whole spectrum—a reflection of a white light, for example, looks white. This degrades the ability of the color thresholding to cleanly detect such a target.

Most materials are between matte and glossy. This means that the reflection of a white light on most surfaces looks whiter than the material itself, not just brighter—the color saturation goes down. The ideal matte surface does not change color saturation when lighting sources are presented at different angles; it just changes brightness.

5.1.3 Choosing Lighting

We first suggest using lots of light. Most cameras start giving grainy pictures as light levels fall.

Because surfaces are never perfectly matte, color saturation will always depend somewhat on the angle of the target's surface relative to the light source and camera. This causes the worst problems when your lighting is a point source. To minimize the problems of glossiness, ideally, you want to use indirect lighting from many directions.

To avoid shadows across the target, you may want to use lights on the same side of the object as the camera. It is bad to light your target only from behind.

Finally, for some colors and some cameras, we have had slightly better performance illuminating targets with fluorescent light rather than with incandescent light. This appears to be because the spectrum of some incandescent lights contains very little green or blue component, thus reducing the effective range colors in the image.

For other pointers on getting a high-quality video input, see Section 4.3.1 [Video Input], page 19.

5.2 Color Training

Once you have chosen a target object, you will want to train the vision system to threshold for only objects of that color. The vision system has three color channels: `a`, `b`, and `c`. You can set each of these channels to threshold for a different color.

5.2.1 Overview of Video Thresholding

Each pixel of the video input is digitized into a 24-bit RGB value. This value consists of 8 bits each of the red, green, and blue components of the light which reached the camera at that point in the image. The goal of the thresholding stage of the vision system is to classify each point in the image as being interesting, meaning that it may be part of a target you want to track, or uninteresting, meaning that it probably is not part of a target. By looking at the debugging video output from the vision system, you can see how various regions of the image are classified at any given time. The interesting parts of the image are shown as being white, and the rest is left black.

Pixels are classified as interesting or uninteresting in hardware by comparing the RGB value of the pixel to a set of values which were defined as interesting when the system was trained. This is done by using the value as an address into a lookup table which maps RGB values into a single bit for each channel: 1 if it is interesting, and 0 if it is not.

The space of possible RGB values contains 16,777,216 distict points. If one bit were stored for every possible value which could come from the video digitizer, this would require over 16 megabits of thresholding information per channel, times 3 channels. Building an inexpensive system with a lookup table this large is not feasible, so a folding scheme is used to reduce this number to a reasonable amount. Instead of defining an arbitrary volume of 3 dimensional RGB space, two 2 dimensional regions of the RG and BG planes are defined to be interesting. The projection of an incoming RGB value into the RG and RB planes must be contained in the interesting region in order to classify that value as interesting.

The set of RGB values in an image of a given object depend on many factors, including characteristics of the particular camera, lighting, and material in any given situation. Human eyes and brains are very good at adapting to changing circumstances. We interpret a given object as appearing constant and distinct from the background in widely varying conditions. However, the actual RGB values which various cameras would provide for the same object in the same set of varying conditions would vary considerably. Therefore, we find that it is best to train the vision system to recognize certain targets in certain situations from empirical measurements. Methods for doing this are described in the next section.

It is possible to train the system to detect an object in a wide range of lighting conditions. In order to do this, you should follow the directions for training on the color of your target, then incrementally train the system through a range of different conditions. However, the range of conditions in which a given target can be accurately distinguished from the background is limited. It must be the case that the range of values acquired for the target does not intersect with the range of values for the background. Otherwise, you may detect background objects when they enter the range of trained target values. This is not necessarily a fatal problem—so long as you can always distinguish the actual target from false positives in the background you should be fine. However, such false positives will make the recognition problem more difficult, and may adversely affect performance.

For example, say that you want to track a light blue target in a dark blue room. For a limited range of lighting levels, this may work fine because the set of values the target and walls occupy may be disjoint. However, when the lights dim, the values for the target may overlap with the values the walls had when the scene was brighter. This problem also arises between more and less saturated colors. For example, an orange object in dim lighting becomes indistinguishable from brown.

There is therefore a tradeoff between the range of lighting conditions and the properties of the background. If everything in the background is sufficiently different from the target, you can train for a wide range of lighting conditions with no loss of performance. However, if aspects of the background are close to the color of the target, you may only be able to get good performance in a small range of lighting conditions.

See Section 5.1 [Choosing Tracking Targets], page 25, for guidelines on how to choose a target so as to minimize this problem. Then, if it is possible, the best solution is to modify the background to get rid of troublesome elements. Otherwise, you can try to control the lighting. For example, put up curtains if the sun makes the lighting conditions (both in terms of intensity and spectral characteristics) change according to time of day. The final option is to determine a way to recognize which objects are the ones you are interested in, and accept that false positives may occur.

5.2.2 Training the Vision System to See a Target Color

It is possible to train the vision system for a certain color either from TPUVIS mode or, if you are running the Pioneer software, from Saphira interaction mode. The training procedure and commands are the same in either case, but the method of entering them and specifying which channel to affect is different.

In TPUVIS mode, single character commands are typed in the window displaying the TPUVIS prompt (see Section 6.3 [TPUVIS Prompt Mode], page 50). Which channel is affected by a given command is modal and depends on what the current channel is set to. You can tell what the current channel is by looking at the prompt; the current channel is shown in brackets:

```
tpuvis[a]>
```

In Saphira mode, commands are issued by typing at the command line with the `saphira>` prompt, then hitting RETURN. Most commands consist of multiple characters, including a channel specification. See Section 6.4 [Saphira Interaction Mode], page 56.

Sample the Color of the Target Object

The first step in training the system to detect a certain color is to place an appropriately colored training target in the center of the camera's viewing area and sample that color by using the lower case '`s`' command.

This command will grab a single frame of video, find the color at the exact center of the image, and use that color as the seed color for the training. It will then move out from that point in the image and try to find the outline of the object. If you are using this command from the TPUVIS prompt, a window containing this outline should appear. Once it has found the outline of the object, it will take all of the RGB values from the inside of that area of the image, and classify all of those values as interesting. This set of values will usually be sparse. As the final step, it will fill in the gaps to leave a smooth region.

This step works best when the training target is large in the image, solid colored, convex, and contrasts well with the background. If the training target does not have these characterisics, the outlining step may not work well.

The criteria used for detecting the outline of the training target is controlled by the `diffthresh` persistent parameter. This specifies how different a color can be from the center color and still be considered to be part of the same object. The default value is 5000. Increase this value if the system is not classifying a large enough region as belonging to the training target. Decrease this value if the system is classifying too large a region as belonging to the object. See Section 8.3 [Setting and Storing Persistents], page 72, for information on how to change this value.

You may want to try this several times with different values of `diffthresh` until you get a good initial training. Don't worry about getting it perfect—there are more steps to go. However, you should be sure that the largest block of thresholded pixels in the image corresponds to the object you are interested in.

Grow or Shrink the Region

In the case that the sample step described above resulted in some of the background being incorrectly thresholded high, you can shrink the edges of the training set by using the '<' (less than sign) command. This will nibble away the edges of the thresholding region, hopefully leaving the training target strong while removing any false positives.

If instead the sample step resulted in no false positives, but only ended up accepting a small part of the target object, you can use the '>' (greater than sign) command. This will increase the size of the thresholding region, while leaving the center the same.

How much is removed from or added to the edges of the thresholding set is controlled by the `color_grow` persistent. Setting this value is somewhat analogous to choosing what grit of sandpaper to use. Increasing this number makes each command have a larger effect, and decreasing this number makes each command have a smaller effect. The default value is 5. See Section 8.3 [Setting and Storing Persistents], page 72, for information on how to change this value.

Incrementally Train the System

Once you have trained the system to detect roughly the correct color, you can use the lower case '`j`' command to track the target object, and incrementally add the color of the center of it to the thresholding set. The target object does not need to be as constrained for this step as it was in the first step: it can be smaller and you can move it around.

For each frame of the incremental tracking, a small box centered around the RGB value of the center of the tracked target is added to the training volume. The size of this box is set by the `boxsize` persistent value. The larger this size is, the greater the effect of each incremental training frame. The default value is 20. If you set this number to be too large, you may start to get false positives in the background. Use the shrink command to correct this, then come back to do more incremental training.

Save the Training and Parameters to Serial EEPROM

Once you are happy with the color settings and persistent parameters, you should save them to non-volatile storage. The vision system contains a small dedicated serial EEPROM for this purpose. When you are ready to save the settings, use the capital '`S`' command. This will save the training of all three channels, and all of the persistents to the serial EEPROM.

IMPORTANT The stored training values are reloaded from the serial EEPROM on every reset. If you are happy with new settings, you should be sure to issue this command before resetting the board. The persistent values are slightly different. On reset, it will check if the persistent values stored in RAM are still valid. If not, it will use the values in EEPROM to reinitialize the table. However, if the values are still valid, it will not overwrite the values in RAM with the values stored in EEPROM.

5.3 Protocol String

The protocol string is a way of customizing how much processing is done on each channel and what format the data is uploaded over serial while the system is in tracking upload mode (see Section 6.2 [Tracking Upload Mode], page 49). It allows customization without recompilation, so long as the data is sent in a format supported by the board's version of `printf` (i.e., ASCII encoded decimal or hex, or a one- or two-byte value encoded in base 254.)

The protocol string is actually made up of four substrings, the header, channel A, channel B, and channel C strings. These substrings are set by using the TPUVIS '`p`' command (see Section 6.3 [TPUVIS Prompt Mode], page 50). When running the default tracking algorithm which uploads data according to the protocol string, data is uploaded by calling `printf` with format and arguments calculated from the protocol strings.

By default, data is sent only for the largest blob in each channel. However, the maximum and minimum number of blobs per channel can be set with the '`n`' TPUVIS command (see Section 6.3 [TPUVIS Prompt Mode], page 50). Tracking data will be sent for the N largest blobs, where N will be no fewer than the set minimum, and no greater than the set maximum.

- If there are fewer than the specified minimum number of blobs, the remainder of the data will be 0's.
- If the maximum and minimum are set to be the same, then the data uploaded in each frame will be the same length; this simplifies parsing the results.
- If the minimum is set to 0, then only as much data will be sent as there are visible blobs.

In the case where the maximum and minimum are set to be different values, there will have to be some marker in the protocol strings to let the application determine how many blobs were sent for each channel.

5.3.1 Protocol Codes Summary

Certain capital letters have special meaning in a protocol string and are used to determine what the arguments to printf will be, and how much processing needs to be done on the given channel. These special characters are removed from the string in order to create the format argument to printf. The following is a list of all the special characters defined so far and the values they generate. The other capital letters are reserved for future use, so you should avoid the use of capital letters for the formatting part of the protocol string.

'F'	Frame count mod 65536 (range 0-65535)
'G'	Frame count mod 256 (range 0-255)
'N'	Square root of area of blob (range 0-255)
'R'	Row number of centroid (range 0-255)
'C'	Column number of centroid (range 0-200)
'Y'	Row coordinate of centroid multiplied by 128 (range 0-32640)
'X'	Column coordinate of centroid multiplied by 128 (range 0-25600)
'A'	Angle of blob (range 0-180)
'S'	Aspect ratio of blob times 10 (range 10-255)
'W'	Width of the bounding box of blob.
'H'	Height of the bounding box of blob.

5.3.2 Protocol Codes Detail

'F' or 'G' The frame count increments once for every field of video. This number is calculated on interrupts and is an accurate way of telling whether any fields have been missed. This number is reset to 0 every time the tracking is started. 'F' is the frame count modulo 65536, so it will wrap after 18 minutes. 'G' is the frame count modulo 256, so it will wrap after 4.2 seconds.

'N' The size of the largest blob, calculated by taking the square root of the pixel area of the blob. This is roughly equivalent to the diameter of the blob, and would be the length of the side of the blob if it were square.

'R' Row number of the center of the largest blob for the given channel. The row number starts at 0 at the top of the screen and increases towards the bottom. You should calibrate the row and column numbers; the aspect ratio of the vision system is not square.

'C' Column number of the center of the largest blob for the given channel. The column number starts at 0 on the left side of the screen, and increments towards the right. You should calibrate the row and column numbers; the aspect ratio of the vision system is not square.

'Y' or 'X' These are exactly like 'R' and 'C', respectively, except that they are multiplied by 128. This allows for more precision in computing the centroid.

'S' (capital)
: Aspect ratio of the blob, times 10. This is the square root of the ratio between of the second moments around the major axis and minor axes, times 10. Intuitively, this gives you an "aspect ratio" for the blob. If you are looking at a circle (or a square), you'll get 1. If you look at an oval (or rectangle) that's twice as long as it is wide, you'll get 2. Three times as long, and you'll get 3. Note that this value is output is scaled by a factor of 10 and then truncated to avoid sending floating point numbers; thus, a circle would output 10, while a rectagle 2.5" x 1" would give you 25.

  Note: Currently the blob aspect ratio is *not* compensated for the aspect ratio of the vision system. This adjustment cannot be done easily (in the general case) as a post-processing step. We plan to allow users to input calibration data and have the system aspect ratio automatically compensated in a future release.

'A'
: Angle of the major axis of the blob. This number is only meaningful if the tracked object is elongated. Therefore, consult the aspect ratio to get an idea of how much to believe the angle.

  Note: Currently this angle is *not* compensated for the aspect ratio of the vision system. This adjustment cannot be done easily (in the general case) as a post-processing step, so we will soon release a new version of the software that will allow users to input calibration data and have the aspect ratio automatically compensated.

'W'
: The width of the bounding box of the blob.

'H'
: The height of the bounding box of the blob.

5.3.3 The %w printf Code

The board's version of `printf` has an additional nonstandard feature. You can use the '`%w`' printf code to print a value in base 254, using either one or two bytes. This is useful because it will have a fixed number of bytes regardless of ARC's stream protocol. See Section 4.2.3 [RS-232 Port Protocol], page 18, for more information.

Like standard `printf` formatting codes, you can specify a field width by placing a number between the '`%`' and the '`w`'. For example, `%1w` specifies a field width of 1 character. Unlike standard `printf` formatting codes, the `%w` code treats the field width as an exact number of bytes to send, rather than just a minimum. This is good because it means that the `%w` code will produce a fixed number of characters regardless of the value. If the field width is unspecified, the default width of 2 characters is used.

Field width = 1

The argument is printed as a single byte whose value is between 0 and 253. If the arument is greater than 253, the number modulo 254 will be sent. If the argument is less than 0, the results are undefined.

Field width = 2

The argument is printed as two bytes: *b1*, then *b2*. The value is *b1**254+*b2*. If the argument is greater than 64515, the argument modulo 64516 will be sent. If the argument is less than 0, the results are undefined.

Field width > 2

At this time field widths greater than two are not supported. The results are undefined.

5.3.4 Summary of printf Formatting Codes

The following table summarizes the strengths and weaknesses of using various encoding methods for transmitting serial information for machine consumption.

Formatting Code	Encoding Method	Efficiency	Debugability	Unique Framing Characters	Range	# Bytes Required
%c	Binary	Fastest	Poor	None	0-255	1 byte
%w	Binary, Base 254	Fast	Poor	255 (\377 in C string)	0-64515	1 or 2 bytes (log base 254)
%x	Encoded Hexidecimal	Medium	Good	All except 0-9, a-f, or A-F	unlimited	log base 16
%d	Encoded Decimal	Slowest	Best	All except 0-9	unlimited	log base 10

See Section 5.3.3 [The %w printf Code], page 33, for more information on the `%w` formatting code. Refer to any C reference manual for more information on the other formatting codes.

Flags, field width, precision, and size modifiers can be specified between the '`%`' and the formatting code. These are supported by the version of `printf` on the processor board. You should not need to use the size modifier flags ('`l`', '`L`', or '`h`') as all of the arguments to `printf` generated by protocol codes are cast to be 4 byte integers (which is the default if no flag is specified).

Note that in accordance with the ANSI specification, the field widths parameter only specifies a minimum and not a maximum field width. For example, if a number larger than 255 were formatted using the `%02x` code, it would contain more than 2 characters. See Section 5.3.1 [Protocol Codes Summary], page 31 for information on the range of values each protocol code generates.

5.3.5 Controlling the Amount of Processing

As well as specifying the upload format, the protocol string is also used to determine how much processing should be done on each channel. Only the minimum amount of processing required to calculate the information requested in the protocol string is done. The processing level is determined by which formatting code in the current protocol string appears lowest on the following list.

(none) No processing

'F', 'G' Edges acquired only

'N' Edges acquired, sizes calculated, and largest found

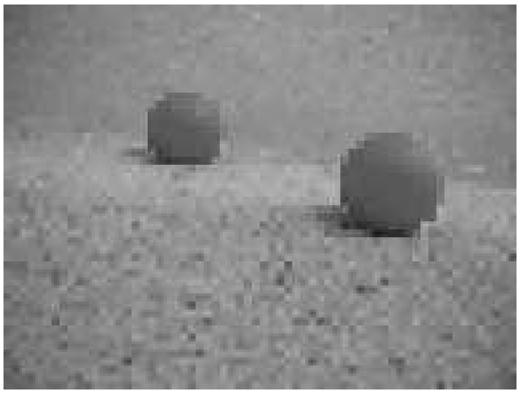
'R', 'C', 'X', 'Y', 'W', 'H'
Edges acquired, sizes calculated, largest found, and center of largest blob calculated.

'A', 'S' (capital)
Edges acquired, sizes calculated, largest found, and center, moments, angle, and aspect ratio of largest blob calculated.

5.3.6 Single Color Protocol String Example

In this example, we want to track the positions of each of two orange balls. Channel A is trained to detect orange, and the other channels are not used. The maximum number of blobs is set to be 2 using the 'n' TPUVIS command (see Section 6.3 [TPUVIS Prompt Mode], page 50). The tracking data shown is for the following static sample image.

If you do not see this image in color, you can see it at:
'http://www.newtonlabs.com/cognachrome/cognachrome_5.html#protocol_example1'.



5.3.6.1 Human Readable Protocol String Example

For nice, human readable output, you could use the TPUVIS 'p' command (see Section 6.3 [TPUVIS Prompt Mode], page 50) to set the protocol string to the following:

```
Header: "frame %3dG:\n" Channel A: "\ta: diameter %3dN, cg (%dC,%dR)\n"
Channel B: "" Channel C: ""
```

Using the 'r' TPUVIS command (see Section 6.3 [TPUVIS Prompt Mode], page 50) to test the new protocol string on the sample image yields:

```
frame   1:
        a: diameter  41, cg (189,115)
        a: diameter  31, cg (105,82)
frame   3:
        a: diameter  41, cg (189,115)
        a: diameter  31, cg (106,82)
frame   5:
        a: diameter  41, cg (189,115)
        a: diameter  31, cg (105,82)
frame   7:
        a: diameter  41, cg (189,115)
        a: diameter  31, cg (106,82)
frame   9:
        a: diameter  41, cg (189,115)
        a: diameter  31, cg (106,82)
```

Note that the frame numbers show that the data was not sent for every frame. Video contains 60 frames per second, and the frame number reported by the 'F' and 'G' commands count every frame. Using this protocol string yields around 80 characters of data uploaded per frame.

At a baud rate of 38400 baud, you can send data at a rate of 3840 characters per second (baud measures bits per second, there are 10 bits per character). This works out to 64 characters taking a full sixtieth of a second—the same amount of time which it takes to receive a full frame of video. Therefore it is not possible to send the data for every frame using this protocol string. In this situation, the system skips sending data for frames until the serial buffer is clear. The alternative would have been to send every frame with increasing delays until the serial output buffer filled, then throw out the overflow characters. This decision was made in order to minimize latency, which is a very important factor in stably controlling dynamic systems.

5.3.6.2 Machine Readable Protocol String Example

If you have a program parsing the data stream, you do not have to format the data in a way which a human would find easy to read. Instead, you want to make the information easy for a program to interpret, and you want to pack a large amount of information into a small number of bytes of transmission data. In this example, the same information is sent in the same order as in the human readable example, but it is intended to be read by a machine.

The highest information density can be achieved by using the `%c` or `%w` binary formatting codes (see Section 5.3.3 [The %w printf Code], page 33). However, the `%x` hexadecimal formatting code has some advantages, even though it requires approximately twice as many characters:

- The problem of synchronization and framing (determining which characters mean what) is easier with hexadecimal encoding of the numbers. You can use any characters except numeric digits and letters from `a` to `f` for framing and never confuse them with the numbers.

  This problem is much more difficult when using the `%c` binary formatting code because there are no characters which are guaranteed not to appear as part of a number. Also, while it is possible to use 255 as a unique framing character when using the `%w` format code, the processing of the protocol string does not yet support numeric literals (i.e. \377 for 255). Therefore there is no convenient way for 255 to be encoded into the protocol string. This will be fixed in future versions.
- Using the binary formatting codes will almost certainly result in non-printable characters being sent in the serial stream. This makes it very difficult to even display the data, much less read it.
- It is often challenging to write parsing software (software which extracts the information you want from a known format). It is a big advantage if you can read the data and verify that your software is generating the expected result.

In this example, hexadecimal encoding is used. Also, you will notice that the full formatting specification is `%02x`. The '`0`' (zero) specifies that zeroes should be used for padding instead of spaces. The '`2`' specifies that the minimum field width is 2 characters. See Section 5.3.4 [Summary of printf Formatting Codes], page 34, for more information.

Only the '`G`', '`N`', '`C`', and '`R`' protocol codes are used in this example—all of which are specified not to exceed 255 (see Section 5.3.1 [Protocol Codes Summary], page 31). Therefore the data for each object will contain a constant number of characters.

This example uses the TPUVIS '`p`' command (see Section 6.3 [TPUVIS Prompt Mode], page 50) to set the protocol string to the following:

```
Header:         "$%02xG"
Channel A:      "#%02xN:%02xC:%02xR"
Channel B:      ""
Channel C:      ""
```

Using the 'r' TPUVIS command to start tracking with this protocol string yields the following results. Carriage returns have been added here to make it more readable. The actual data sent contains no carriage returns.

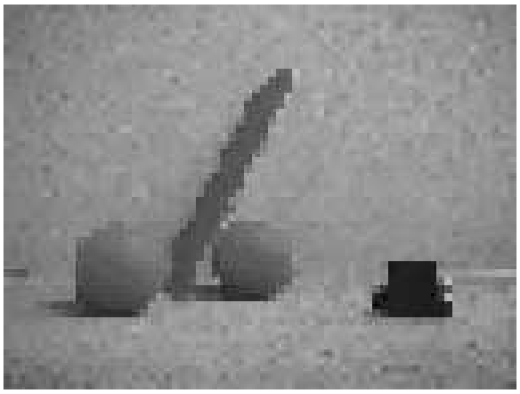
```
$01#29:bd:73#1f:6a:52
$02#29:bd:74#1f:6a:53
$03#29:be:74#1f:6a:52
$04#29:bd:74#1f:6a:53
$05#29:be:74#1f:6a:52
$06#28:be:74#1e:6a:52
$07#29:bd:73#1f:6a:52
$08#28:bd:74#1f:6a:53
$09#29:bd:73#1f:6a:52
$0a#29:bd:74#1f:6a:53
```

Note that while frames were skipped in the human readable example, all frames are present in this example. This is because only 21 characters are being sent per frame. It only takes 5 milliseconds to send 21 characters at 38400 baud. Each frame takes about 17 milliseconds, so it is possible to send all of the data without skipping any frames.

5.3.7 Multiple Colors Protocol String Example

In this example, we want to track the positions of each of two orange balls; the position, aspect ratio, and angle of a blue piece of cardboard; and the position, and bounding box dimensions of a dark green serial adapter. Channel A is trained to detect orange; B is trained to detect blue; and C is trained to detect dark green. The maximum number of blobs is set to be 2 for channel A, and 1 for channels B and C. The tracking data shown is for the followint static sample image.

If you do not see this image in color, you can see it at:
'http://www.newtonlabs.com/cognachrome/cognachrome_5.html#protocol_example2'.



The protocol string is set up to be the following:

```
Header:          "$%02xG"
Channel A:       "#%02xN:%02xC:%02xR"
Channel B:       "*%02xN:%02xC:%02xR:%02xS:%02xA"
Channel C:       "&%02xN:%02xC:%02xR:%02xW:%02xH"
```

Using the 'r' TPUVIS command to start tracking with this protocol string yields the following results. Carriage returns have been added here to make it more readable. The actual data sent contains no carriage returns.

```
$01#21:50:a8#1d:86:a2*23:7b:74:64:47&1d:c5:bd:22:25
$03#21:50:a7#1d:86:a2*23:7b:73:64:47&1c:c6:bc:21:26
$05#22:50:a8#1d:86:a2*23:7b:74:64:47&1c:c6:bd:21:25
$08#22:50:a7#1d:86:a1*23:7b:74:64:47&1c:c6:bb:1f:24
$0a#22:50:a7#1d:86:a1*23:7b:75:64:47&1c:c6:bc:21:26
```

You can see that frames were skipped in this example. Here 51 characters are sent per frame, so it takes 13 milliseconds per frame at 38400 baud to upload the information. Since each frame takes about 17 milliseconds, it would be possible to send this much serial data without missing any frames. However, processing three channels with angle information takes long enough that some frames must be skipped. See Section 5.4.3 [Multiple Channel Tracking Plus Angle Performance Example], page 41, for a more in-depth analysis of the calculation time used by this example.

5.4 Tracking Performance

You can use the 'm' TPUVIS command to obtain rate and latency information when no serial data is sent (see Section 6.3.4.3 [Debugging and Miscellaneous Detail], page 55). Which channels are acquired and how much processing is done for each is set by the protocol string. This is so that the information can easily be compared against tracking performance in normal operation.

Each frame for which the data is available is processed as usual, but no data is sent over the serial port. Instead, the amount of time which passed between when the data started to become available and the time when the processing was completed is recorded. A fixed number of frames are processed, and the number of frames which had to be skipped is calculated. The output of this command is formatted in the following manner.

tpuvis [a]> m Missed M over N frames: avg latency L ms

N The total number of frames which were processed

M The total number of frames which were not processed

$(N+M)$ The total number of frames which arrived from the camera

L The number of milliseconds which passed on average from when data started to become available for the frame (measured at the end of the vertical retrace) to the time when the processing of the frame was completed.

The system has two edge acquisition buffers. Generally one is being filled while the other is being processed. When a new frame starts, whichever buffer is not being processed is filled with the incoming data. If the most recently processed buffer has not been released yet when it would be time to refill it, the buffer which was most recently filled will be refilled, and data for the previous frame will be lost.

The camera provides 60 frames per second, so the total time during which frames were acquired is $T = (N+M)/60$ seconds. During this time N frames were processed. This leads to an output frame rate of $R = N/T$, or $R = 60*N/(N+M)$ Hz (frames per second).

From this equation, you can see that if zero frames are missed, $R = 60$ Hz. As more frames are missed, the frame rate degrades smoothly.

Another way to look at the data uses the latency figures. It takes 17 ms from start to finish for a single frame to arrive from the camera. It then takes some additional time for the system to finish processing the information, and release the buffer to be refilled. This extra time is $E = L-17$ ms. If $E > 17$ms, then frames will definitely be missed. It is difficult to say more than that about the relationship between latency and frame rate.

The following examples give performance specifications for the same scene as the multiple colors protocol string example (see Section 5.3.7 [Multiple Colors Protocol String Example], page 38) at different levels of processing.

5.4.1 Single Channel Tracking Performance Example

Processing the scene and calculating the largest blobs on the first channel only yields the following:

```
tpuvis [a]> m
Missed    0 over  500 frames: avg latency   20 ms
```

Here $M = 0$, so the frame rate is a full 60 Hz. It was possible for the acquisition buffer to be switched every time, so no data was lost.

5.4.2 Multiple Channel Tracking Performance Example

Processing the same scene and calculating the center of the largest blobs on all three channels without calculating angle provides the following results:

```
tpuvis [a]> m
Missed    0 over  500 frames: avg latency   23 ms
```

Again $M = 0$, so the frame rate is a full 60 Hz. Both of the last two examples would result in the same frame rate, but this would have slightly more latency than the previous one.

5.4.3 Multiple Channel Tracking Plus Angle Performance Example

Here the protocol string from the protocol string example (see Section 5.3.7 [Multiple Colors Protocol String Example], page 38) was used. The center of the largest blobs on all three channels, plus the aspect ratio and angle on channel B were processed:

```
tpuvis [a]> m
Missed  249 over  500 frames: avg latency   29 ms
```

Here $M = 249$ and $N = 500$. The frame rate $R = (60*500)/(500 + 249) = 40$Hz. This tells us that the processing buffer was not ready to be refilled about a third of the time. This was caused because the angle and aspect ratio calculations are quite expensive compared to just tracking the center of the objects. In this case, the added time for computing angle on one channel was 6 ms—a 26% increase.

6 Vision System Software

6.1 Software Overview

This section provides an overview of the vision system software. This includes relevant behavior of the system software running on the vision system's processor board. Topics covered include software startup behavior, how to download new software, and how to interact with the vision system specific software.

6.1.1 Startup Behavior

When the 68332 processor board starts up, it goes through several stages. Each stage can be recognized by the information it sends over the serial port.

Bootstrap Startup

First the bootstrap starts up and sends the following message at 38400 baud.

BOOTSTRAP <*version number*>

This message will always be at 38400 baud, regardless of what baud you are running at. Therefore, if you are running at 9600 baud it will appear as some garbage characters. Do not be alarmed by this.

Kernel Startup

Next the Kernel (the operating system on the processor board) starts up, sets the baud rate to be what it was last running at (in the case of the Pioneer vision software, this will be 9600 baud, otherwise it will be 38400 baud), checksums itself, and loads custom microcode. It prints a message like the following:

```
Checksum OK
Restoring all backups unless RAM programs are newer
Microcode loaded from 17f28 to fff000, 800 bytes

Anarchboot Multitasking 68332 Kernel Version 0.986
Copyright Anne Wright 1993

Compiled May  6 1996 at 15:20:43
```

The actual boot message will change with different versions of the kernel, but it will generally be in a similar form to that shown above.

Check for Programs to Run on Reset

Next it checks whether there are any programs which should be run on reset. If it detects any, it will give you an opportunity to keep them from running (see Section 6.1.2 [Inhibiting Reset Run of Programs], page 45).

```
Checking for reset run programs
Control-c to inhibit reset run...10...9...8...7...6...5...4...3...2...1...
```

Library Initialization

If you did not prevent programs from being run on reset, it will start to run them at this time. First, it will initialize any libraries it finds. In the case of the software for both the stand alone and Pioneer vision system software, it should run the library 'vislib-xx_x' (where 'xx_x' refers to the version number of the vision library).

The vision library will beep, read the color training and persistent parameters which had been saved to the serial EEPROM on the vision system (see Section 5.2.2 [Training the Vision System to See a Target Color], page 28), then beep again.

```
Initializing library vislib-26_2
Cognachrome Vision System
Software version 26.2   [beep]
Channel 0: Restored 393 bytes
Channel 1: Restored 1035 bytes
Channel 2: Restored 6 bytes
Restored a total of 1434 bytes
Restored protocol string: "$    %04xF:%04xR:%04xC#" from EEPROM [beep]
```

Application Program Startup

Finally it runs any application programs that are loaded. In the case of the stand alone software, the application program should be 'vision-xx_x' (where 'xx_x' refers to the version number). In the case of the Pioneer software, this should be 'pioneer-xx_x'.

```
Running program vision-26_2 as process 99
vestaboot.986>
```

If this sequence does not proceed according as described above, determine the first step at which the actual behavior diverged.

- If you did not see any information from the serial port whatsoever, see Section B.3 [Serial Port Troubleshooting], page 87.
- If you did see the bootstrap startup message (remember that it appears as garbage at rates other than 38400 baud), but did not see anything else, you need to download a new vision installation package. See Section 6.1.4 [Downloading Programs to the Vision System], page 47.
- If you saw the kernel startup messages, but it did not find any programs, you need to download both the vision library ('vislib-xx_x') and application code ('vision-xx_x' or 'pioneer-xx_x'). See Section 6.1.4 [Downloading Programs to the Vision System], page 47.

- If you saw the library initialization messages, but it did not start any application programs, you need to download the application code (‘`vision-xx_x`’ or ‘`pioneer-xx_x`’). It may also be necessary to download a different copy of `vislib-xx_x` in order for the versions to be compatible. See Section 6.1.4 [Downloading Programs to the Vision System], page 47.

6.1.2 Inhibiting Reset Run of Programs

When the board is reset, the kernel will normally initialize libraries and run application programs automatically (see Section 6.1.1 [Startup Behavior], page 43). However, there is an opportunity to prevent it from performing this step. This is particularly useful if you are writing your own software and are putting the serial port into raw mode, or if you are using the Pioneer software and wish to prevent the vision system from entering Saphira interaction mode (which runs with the serial port in raw mode).

Do the following to inhibit programs from being run on reset:

- Press the vision system RESET button.

  For vision systems installed in Pioneer robots, do not confuse the reset for the vision system and the robot itself. The vision system reset button is the one labeled *Boot Button* on the vision interface panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7).
- Press Control-C after it has started printing dots counting down from 10 to 0 immediately after reset:

  ```
  Checking for reset run programs
  Control-c to inhibit reset run...10...9...8...7...6...5...4...3...2...1...
  ```

There is only a small window of opportunity, as pressing Control-C before or after the dots will not work. If you succeed at pressing Control-C in the appropriate interval, it will say:

```
Reset running of programs inhibited
```

If nothing appears when you press reset, check the power and serial connections, and check that you are using the correct serial port (see Appendix B [Troubleshooting], page 85). If you get reasonable boot messages (see Section 6.1.1 [Startup Behavior], page 43), but it does not print the above message, most likely there are no programs loaded. See Section 6.1.4 [Downloading Programs to the Vision System], page 47.

6.1.3 Interaction Modes

The behavior of the vision system is governed by the *interaction mode* it is in at a given time. The mode determines what processing it will perform, what information it will send over its serial port, and what commands it will respond to or ignore. The following table provides an overview of these modes. They are described in more detail in later sections.

	Tracking Upload Mode	TPUVIS Prompt Mode	Saphira Interaction Mode
Purpose	Upload tracking data according to protocol string	Interact with user to adjust parameters and debug system	Interact with Saphira robot control software
Software required	Any	Any	Pioneer software only
Serial interaction	Uploads data to ARC tracking stream (28)	Interacts with user on ARC TPUVIS interaction stream (usually 3 or 4)	Only sends and receives PSOS packets (Saphira or ARC saphira> prompt)
ARC commands available	All	All	None
Entering	Stand alone software enters on reset, otherwise 'r' TPUVIS command	'q' command from other modes	Enters on reset, or '9' TPUVIS command (Pioneer software only)
Exiting	'q' or 'Q' to tracking stream or interaction stream	'r', 'Q', or '9' (Pioneer only) TPUVIS command	PSOS packet containing string "q" (Saphira or ARC saphira> prompt)
Vision processing	Data acquired, processing specified by protocol string	None	Data acquired, processing specified by persistent variables

Tracking Upload Mode

In Tracking Upload Mode, the vision system processes and uploads tracking data. The format of the tracking data is configurable by setting the *Protocol String* (see Section 5.3 [Protocol String], page 31). It will continue to upload data until it is stopped (see Section 6.2.2 [Exiting Tracking Upload Mode], page 50).

TPUVIS Prompt Mode

In TPUVIS Prompt Mode, the vision system interactively accepts commands which can affect configuration or perform diagnostics. You should use ARC to interact with the vision system in this mode. The interaction is performed in a separate window from the tracking uploads.

Saphira mode

This mode is only available if you are using the Pioneer software. In Saphira mode, the vision system runs at 9600 baud and only responds to PSOS packets, which is the data format used by the 6811 board on the Pioneer and the Saphira robot control program (see Section 6.5 [Saphira Packet Protocol], page 61). In this mode, the vision system will act as a bridge between Saphira and the robot processor, handle PSOS packets from the host directed to the vision system, and upload vision data to the host formatted as PSOS packets.

You must exit this mode in order to download programs or use regular ARC commands.

In all modes except Saphira Interaction Mode, the vision system interacts with the host over a multi-streamed protocol. This allows ARC to interact simultaneously with multiple processes on the board, and present each interaction as a separate window. It also allows ARC to communicate with the board to download programs, set the baud rate, etc. (see Section 6.1.4 [Downloading Programs to the Vision System], page 47).

6.1.4 Downloading Programs to the Vision System

In any mode except Saphira mode, you can download programs using the regular download commands. If you are in Saphira mode, exit first before downloading (see Section 6.4.2 [Exiting Vision System Saphira Interaction Mode], page 58). This section describes the minimal downloading instructions, which is all you should need. See the *ARC User's Guide* for more details. It is available at:
'`http://www.newtonlabs.com/arc/arc_1.html`'

The vision code is distributed in two formats:

S19 Files
: Installation and upgrade packages for the vision system software contain the operating system, libraries, and application code (either stand alone or Pioneer). These are distributed as Motorola S-Record files (`.s19` suffix) containing multiple executables.

Executable Files
: These files are in `a.out` executable format, which is output by the compiler. If you are writing your own programs using the full ARC Development system, you will mainly be downloading files in this format.

WARNING: Downloading either kind of package to non-volatile memory only works for processor boards which have been specially modified to use EEPROM. All processor boards shipped with vision systems sold after March 31, 1996, are modified to use EEPROM. See Section B.2 [Processor Board Jumper Configuration], page 86, for more information.

6.1.4.1 Downloading S19 Packages

To download these packages to EEPROM, start ARC and use the `kernelload` command.

`arc>` *kernelload packagename.s19*

ARC will prompt you to turn off the board and hit RETURN. After you hit RETURN, turn the system back on. Downloading will begin automatically.

It is normal for ARC to pause while it reads the file, so be patient. However, if it does not start downloading after about 30 seconds, there is probably a problem. Double check that serial interaction works in general (see Section B.3 [Serial Port Troubleshooting], page 87).

These `.s19` distribution files contain the bootstrap. This is necessary to allow the same file to work whether downloaded by ARC or a device programmer. As a result, when downloading using ARC, writes to the first 16K of EEPROM will print a message that they are being skipped so as not to overwrite the current bootstrap. This is normal, and is nothing to worry about.

The download will fill the entire EEPROM (to address 0x40000), so it takes a while.

When the download completes, press the vision system reset button. For vision systems installed in Pioneer robots, do not confuse the reset for the vision system and for the robot itself. The vision system reset button is the one labeled *Boot Button* on the vision interface panel(see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7).

6.1.4.2 Downloading Executables

Executables output by the GCC cross compiler are in a.out format. If you are writing your own programs using the full ARC Development System, you will mainly be downloading your programs in this format.

To download these files:

- Exit Saphira mode if necessary (see Section 6.4.2 [Exiting Vision System Saphira Interaction Mode], page 58).
- (optional) If you are using the Pioneer software, or your own software which sets the baud rate slower than 38400 baud, you may want to set the baud rate to 38400 for faster downloading: type `baud 38400` at the `arc>` prompt.
- Decide whether to download the program to EEPROM or RAM. EEPROM is non-volatile, so it has the advantage that you can unplug your board for as long as you want without losing your program. RAM is volatile, so you will lose your program if you unplug the board for too long.

- Unload any programs you want to get rid of. You can get a list of the programs which are currently loaded by using the `list` command. Programs are unloaded by using the `unload` *filename* command at the `arc>` prompt.
- Download the program.

  To EEPROM

  `arc>` *romload filename*

  To RAM

  `arc>` *ramload filename*

- If you changed the baud rate to 38400 for faster downloading, but you were running at a different baud rate before, set the baud rate back to the old value now. In particular, if you are using the Pioneer code, you should set the rate to 9600 baud, as this is required for using the Pioneer software for the vision system.

  ```
  arc> baud 9600
  ```

- To start the program after download, either type `run` *filename* at the `arc>` prompt, or press the reset button. For vision systems installed in Pioneer robots, do not confuse the reset for the vision system and for the robot itself (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7).

6.2 Tracking Upload Mode

In Tracking Upload Mode, the vision system will upload tracking data over stream 28 on serial port A. The format of the tracking data is configurable by setting the *Protocol String* (see Section 5.3 [Protocol String], page 31). It will continue to upload data until it is stopped (see Section 6.2.2 [Exiting Tracking Upload Mode], page 50).

6.2.1 Entering Tracking Upload Mode

With Stand Alone Mode software installed
Press the RESET button.

With Pioneer software installed
Press RESET, exit Saphira Interaction Mode, press '`r`'.

General From TPUVIS prompt, press '`r`'.

6.2.2 Exiting Tracking Upload Mode

Once in Tracking Upload Mode, do one of the following to exit:

- Send '`q`' to stream 3 (TPUVIS interaction stream) or stream 28 (Tracking upload stream)
- Press stop on IR remote control (if IR hardware is installed).

6.3 TPUVIS Prompt Mode

6.3.1 Entering TPUVIS Prompt Mode

With Stand Alone Mode software installed

Do one of the following:

- With the camera connected and on: Press the RESET button. Wait for the tracking output to start. Exit Tracking Upload Mode (see Section 6.2.2 [Exiting Tracking Upload Mode], page 50.
- With the camera disconnected or off: Press the RESET button. Wait for the TPUVIS Prompt to appear.

With Pioneer software installed

Press RESET, exit Saphira Interaction Mode.

6.3.2 Exiting TPUVIS Prompt Mode

Most other modes can be entered from TPUVIS Prompt Mode by pressing a key. See the instructions for entering the other modes for instructions.

With either the stand alone or Pioneer software, if you want to quit the program from the TPUVIS Prompt, press capital '`Q`'. If you do this, the program will no longer be running, and you will have to reset or type `run` *filename* at the `arc>` prompt to run it again.

6.3.3 TPUVIS Command Summary

From the '`tpuvis>`' prompt, you can retrain the three tracking channels and change the parameters used during tracking, as well as other miscellaneous operations. To execute a command, simply type its letter (don't press return). Case is important: some commands are lowercase, while some are uppercase.

6.3.3.1 Turning Tracking On and Off Summary

'r' (remote control: 'play') Start tracking.

'q' (remote control: 'stop') Stop tracking (and give 'tpuvis>' prompt).

6.3.3.2 Training and Configuring Channels Summary

'a' Set current tracking channel ('a', 'b', or 'c').

's' (remote control: '0') Clear current channel, and train on object in center of screen.

'j' (remote control: 'enter') Add to training of current channel, based on currently tracked objects.

'>' (remote control: 'channel up') Grow recognition region for current channel.

'<' (remote control: 'channel down') Shrink recognition region for current channel.

'S' (remote control: 'record') Save trained colors, protocol string, and persistents to EEPROM.

'R' (remote control: 'rewind') Restore trained colors and protocol string from EEPROM.

'f' Configure signal filter.

'p' Change computed parameters and serial transmission format.

'n' Change min/max number of tracked objects per channel.

'c' Go to advanced "colorplot" menu.

'A' Adjust NTSC->RGB converter parameters.

6.3.3.3 Debugging and Miscellaneous Summary

'?' List available commands.

'v' Print version information.

'P' Change persistent variable.

- '0' Track for one frame (for testing protocol string).
- 'm' Benchmark frame acquisition and latency.
- 'l' Track largest object in current channel, with human readable diameter, row, and column output.
- 'o' Track largest object in current channel, with human readable diameter, row, column, and orientation output.
- '+' (remote control: 'volume up') Turn crosshair display on.
- '-' (remote control: 'volume down') Turn crosshair display off.
- 'b' Show, in ARC/X windows graphics mode, the segments seen by the color thresholding circuitry for the current channel before processing.
- 'd' Dump, in text, the segments seen by the color thresholding circuitry for the current channel before processing.
- 'T' Configure the behavior of the system on startup.
- 'Q' Kill the vision process.

6.3.4 TPUVIS Command Explanations

This section describes the individual 'tpuvis>' commands in detail.

6.3.4.1 Turning Tracking On and Off Detail

- 'r' (remote control: 'play') Run color tracking. The amount of processing and format of output is governed by the protocol string. See Section 5.3 [Protocol String], page 31.
- 'q' (remote control: 'stop') Stop tracking if running. In general, if something is running and you want the prompt back, you can type 'q' to get it back.

6.3.4.2 Training and Configuring Channels Detail

- 'a' This command changes the current channel—that is, the channel affected by the other training and configuration commands in this section. It takes a second one-character parameter specifying which is the new channel to use, 'a', 'b', or 'c'. The channel most recently chosen is shown in square brackets in the prompt itself (when the prompt is 'tpuvis [b]>', for example, it means channel 'b' is the current channel).

'`s`' (lower case s)
: (remote control: '`0`') Clear LUT and start with object in center. This is the most convenient way to change what color is being tracked on a given channel. Put a large object of the color you want to track in the center of the screen and type lower case '`s`'. This will clear out what used to be in the LUT and replace it with a region of the color cube containing the color it found in the center and around it in the region seen as belonging to the same object.

  The border of the object is assumed to be where the distance in the color cube from the color in the center to another color exceeds the persistent value `diffthresh` (which defaults to 5000). The scale of `diffthresh` is pretty arbitrary, but when it gets bigger it is more generous about how different a color can be from the center color and still be accepted as belonging to the same object, and when it gets smaller it gets more stingy.

'`j`'
: (remote control: '`enter`') Add tracked color to the LUT. Use this, after getting a reasonable lock on the color you want by using lower case '`s`', in order to add the colors resulting from different distance, lighting, or angle of the target object. The '`j`' command uses the blob tracking capability to find where the target object is, then takes a frame grab and adds the color it finds to the LUT.

'`<`' or '`>`'
: (remote control: '`channel down`' or '`channel up`') Shrink or grow region in LUT. This command takes the region of the LUT which is thresholded high and shrinks or expands it by the persistent amount `color_grow` along all edges. `color_grow` defaults to 5. Increasing this number increases the effect of '`<`' and '`>`', and decreasing it decreases the effect.

'`S`' (capital)
: (remote control: '`record`') Save the current values in the LUT for thresholding all three channels, the protocol string, and the persistents to the EEPROM. It is important to remember to do this after you are satisfied with the effect of retraining the color or changing the protocol string and before the next reset, as on reset the values last stored to the EEPROM are restored.

'`R`'
: (remote control: '`rewind`') Restore the old values of the thresholding LUT and protocol string from the EEPROM for all three channels. This is useful when you've tried to change the settings and you aren't happy with it.

'`f`'
: Change the amount of filtering done. The amount of filtering is the same for each channel; it cannot be set separately for the channels. 0 is the least filtering (none), while 3 is the most filtering. Filtering is only done in the horizontal direction.

  - Filter level of 0 allows the system to recognize objects only one pixel wide.
  - Filter level 1 means an object must be at least 2 pixels wide to be detected.
  - Filter level 2 means an object must be at least 3 pixels wide to be detected.
  - Filter level 3 means an object must be at least 4 pixels wide in order to be detected.

Filtering is typically used to make up for things like a cluttered environment, where small objects in the background might be incorrectly detected.

In typical systems the filtering is usually set to 1. The primary problem with increasing filtering is that it increases the size of the smallest object detectable by the system. (Sometimes making targets that are wide but short can help meet a constraint of small target but large filtering). The primary advantage of increasing the filtering is that the processing time per frame is proportional to the number of transitions in the thresholded image. Increasing the filtering typically reduces the number of transitions by removing the spurious ones, which decreases the processing time per frame, which speeds up the entire system. Therefore, it is best to keep the filter level as high as you can, given the size of the smallest object you want to be able to track.

Be sure to remember the capital '`S`' command to save to EEPROM if you want the new filtering level to persist when you turn off the board.

'`p`' Change a channel's *protocol string*. The amount of processing, which data are output, and the format of the output are all governed by the protocol string. See Section 5.3 [Protocol String], page 31, for a full explanation of the protocol string's format, and what it does.

When you execute this command, you will be given a choice of which protocol string to change, or if you want to modify the header: type '`h`' to change the header, or '`a`', '`b`', or '`c`' to change a particular channel. Be sure to try tracking out afterwards to see if your change had its desired effect—protocol strings can be tricky. You can use the capital letter '`O`' command to track a single frame; this is very useful for protocol string testing.

Remember that commands capital '`S`' and '`R`' save and restore the protocol strings as well as the colors being tracked. If you change the protocol string and are happy with it, be sure to use '`S`' to save it to EEPROM.

'`n`' Change the minimum and maximum number of tracked objects per channel; see Section 5.3 [Protocol String], page 31, for more explanation of this feature.

'`c`' Go to the *colorplot* submenu. The colorplot submenu provides advanced commands for training colors and analyzing color training. Hopefully you will not need to use these features, as we have tried to provide all necessary operations at the top level menu. This submenu is not yet documented; if you need more information, please contact us (see Appendix D [Reaching Newton Research Labs], page 105).

'`A`' Adjust the NTSC to RGB converter on the Sony daughterboard. There are four customizable parameters, each of which ranges from 0 to 255 (with a default of 127). Normally the default works well.

The four parameters are `sony_apl`, which adjusts the *auto picture level* (auto gain control); `sony_sharpness`, which adjusts the sharpness; `sony_saturation`, which adjusts the saturation; and `sony_hue`, which adjusts the hue. See Section 8.4 [Important Persistents], page 73 for more information.

6.3.4.3 Debugging and Miscellaneous Detail

'?' This prints a list of the available commands. The list is likely to be more up to date than this document, so you may want to refer to it even if you don't need your memory refreshed.

'v' This prints the software version that's running on the board.

'P' This lets you change a persistent (see Chapter 8 [Persistent Variables], page 71). First, it prompts you for the persistent to set; you can type '?' to get a list of persistents, or just hit RET to abort the command. Then, it prints the old value of the persistent, and asks for the new value; pressing just RET leaves it unchanged.

'O' This command prints one frame of tracking information in the current window. It prints the same information as the 'r' command (see Section 5.3 [Protocol String], page 31), but only for one frame.

'm' Benchmark tracking and latency. This command performs exactly the same processing as would be done by using the 'r' command, but instead of uploading the data over serial it only takes statistics and prints the accumulated statistics at the end. See Section 5.4 [Tracking Performance], page 39, for more information and examples.

'l' Track largest blob with human readable output. This function tracks the diameter and center of the largest blob and prints it in human readable format like this. (Note that since a large amount of data is printed per frame, you may not receive 60 Hz tracking using this command).

```
Frame   70: N=     27, cg= ( 58,201)
Frame   71: N=     27, cg= ( 58,202)
Frame   72: N=     25, cg= ( 57,201)
Frame   73: N=     27, cg= ( 59,201)
Frame   74: N=     25, cg= ( 56,201)
```

'o' Track largest blob with orientation and human readable output. This command is just like 'l' but it also does angle and aspect ratio calculations. (Note that since a large amount of data is printed per frame, you may not receive 60 Hz tracking using this command).

```
Frame   83: N=     27, cg= ( 60,201), angle=  60, aspect ratio =  16
Frame   85: N=     28, cg= ( 57,202), angle=  53, aspect ratio =  16
Frame   87: N=     27, cg= ( 59,201), angle=  56, aspect ratio =  14
Frame   89: N=     27, cg= ( 58,202), angle=  48, aspect ratio =  15
```

'+' (remote control: '`volume up`') Enable the crosshairs. All the default tracking commands set the position of the crosshairs to be the center of the largest item they are tracking.

Crosshairs are intended to be used primarily as a debugging aid. Because the timer processor generates the crosshairs in software, tracking performance will degrade somewhat when the crosshairs are on, so we recommend leaving crosshairs off during normal operation.

Also, you may notice unusual streaks to the right of the object being tracked while crosshairs are on. This doesn't indicate a problem in tracking, but is rather an artifact of minimizing crosshair's impact on tracking performace (the timer processor gives higher priority to recording the incoming tracking data, so it may be sluggish turning off the crosshair signal if there are a lot of incoming data points to record).

'-' (remote control: '`volume down`') Disable the crosshairs. If the crosshairs are showing, this will make them disappear.

'b' Acquire raw data with debugging enabled. This command prints the data for each blob individually and outlines it in a Tektronix window. This is nice for low level debugging, but it requires that you are running ARC under X in order to work.

'd' Dump raw data to serial port. This is the lowest level debugging available. It prints the row, start column, and end column (in that order) for all the segments captured by the hardware:

```
84: 94 -> 108
85: 88 -> 110
```

'T' Configure the behavior of the system on startup. A menu is printed with three choices:

0 don't run '`vision`'

1 run '`vision`' but don't start tracking

2 run '`vision`' and automatically start tracking

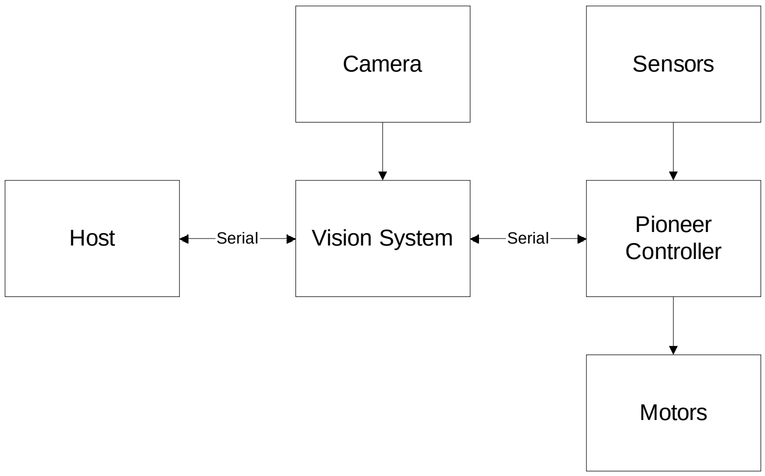
'Q' Kill the vision process.

6.4 Saphira Interaction Mode

This section is only relevant if you are using the Pioneer software (see Chapter 3 [Pioneer Quick Start Guide], page 7). The commands described in this section are not available if you are using the stand alone software (see Chapter 2 [Stand Alone Quick Start Guide], page 3).

When the Pioneer software starts up, it automatically enters Saphira interaction mode (see Section 6.1.1 [Startup Behavior], page 43). In this mode, the vision system runs at 9600 baud and communicates using PSOS packets—the data format used by the 6811 board on the Pioneer and the Saphira robot control program (see Section 6.5 [Saphira Packet Protocol], page 61). In this mode, the vision system will act as a bridge between Saphira and the robot processor, handle PSOS packets from the host directed to the vision system, and upload vision data to the host formatted as PSOS packets. It will not exit this mode unless it receives a PSOS packet instructing it to. See Section 6.4.2 [Exiting Vision System Saphira Interaction Mode], page 58.

The following block diagram shows the communications between the components of a vision-equipped Pioneer.



The vision system has two serial ports. In Saphira mode, one communicates with the host and the other communicates with the 6811-based Pioneer controller board. The host will typically be running the Saphira host program (the graphical robot-control environment for the Pioneer written by Kurt Konolige). You may choose instead to run ARC with the '`-saphira`' flag on the host (see Section 3.3.2 [Starting ARC for use with the Pioneer Vision System], page 10).

PSOS packets from the host contain an address field which tells the vision system whether it should handle the packet itself, or just pass it along to the Pioneer controller. For example, commands for controlling the motors or reading the sonars are passed along to the Pioneer controller unmodified. On the other hand, commands for configuring the vision system are handled locally and are not passed on to the Pioneer controller at all.

This section discusses the commands which Saphira can send the vision system. In normal use, you should not need to issue these commands directly, as Saphira will do it for you. However, in some cases you may wish to issue these commands to the vision system directly without using Saphira. See Section 3.3 [Interacting Directly with the Vision System using ARC], page 9, for information on setting up to communicate with the vision system in Saphira mode from ARC.

WARNING: Be sure that ARC and the Saphira host program are not both trying to talk to the robot at the same time. Exit one before starting the other.

6.4.1 Entering Vision System Saphira Interaction Mode

With Stand Alone Mode software installed
NOT AVAILABLE

With Pioneer software installed
When the Pioneer software starts up, it automatically enters Saphira interaction mode (see Section 6.1.1 [Startup Behavior], page 43). If you have exited Saphira mode using ARC and are currently in TPUVIS mode, you can use the TPUVIS '`9`' command to reenter Saphira mode.

6.4.2 Exiting Vision System Saphira Interaction Mode

There are several ways to make the vision system exit Saphira interaction mode:

- Reset, then prevent the Pioneer application from running (see Section 6.1.2 [Inhibiting Reset Run of Programs], page 45).
- Start ARC using the `-pioneer` flag, turn on or reset the board, and press RETURN. ARC will send a PSOS packet to the board to cause it to exit Saphira mode.
- After starting ARC using the `-pioneer` or `-saphira` flags, issue the `saphira_stop` command at the `arc>` prompt. ARC will send a packet to the board to cause it to exit Saphira mode.
- After starting ARC using the `-saphira` flag, type '`q`' RET at the `saphira>` prompt.

In the first case, the vision code will not run at all. In all of the other cases, the vision system will now be in TPUVIS Prompt Mode (see Section 6.3 [TPUVIS Prompt Mode], page 50). In either case, regular ARC commands will work at this point (the vision system's host serial port will no longer be communicating using PSOS packets).

6.4.3 Saphira Mode Channel Commands

These commands perform operations on the tracking channels. Replace '`%`' with '`a`', '`b`', or '`c`' to specify which channel to affect. These behave the same as their counterpart commands at the

TPUVIS prompt. See Section 6.3.4.2 [Training and Configuring Channels Detail], page 52, for details. These commands are case-sensitive, so be sure to match the case listed here.

'`s%`' Train on color in center of given channel.

'`j%`' Incrementally add to training for given channel.

'`<%`' Shrink accepted color set for given channel.

'`>%`' Expand accepted color set for given channel.

'`a%`' Display tracking output of given channel on *Video Out* port.

6.4.4 Saphira Mode Frame Commands

The frame commands perform a 64x48 pixel frame grab and upload information for display. Currently the frames uploaded by these commands are displayed using the program `xv`. In order to use these, be sure you have installed `xv`, and that it is in your path. The default settings are for an 8-bit display. For `xv` to display a frame with acceptable quality on an 8-bit display, it must construct an appropriate 8-bit colormap. `xv` creates a window with a private colormap, so move the mouse inside the window to see what the frame really looks like (with the mouse outside, you're likely to see some pretty strange artifacts).

Replace '`#`' with '`0`', '`1`', or '`2`' for compression amount. Currently, the only compression performed is by reducing the bits per pixel. In the future, other compression means (such as JPEG) may be incorporated.

'`0`' No compression, highest quality.

'`1`' 2x compression. Moderate quality.

'`2`' 4x compression. Low quality.

Replace '`%`' with '`a`', '`b`', or '`c`' to specify which tracking channel the command applies to.

These commands are case-sensitive, so be sure to match the case listed here.

'`G#`' Grab and display a color frame

'`E#`' Grab a frame, run edge detection, and display the result

'`T#%`' Grab a frame and display the part included by the color tracking for the given channel

6.4.5 Saphira Mode Visual Sonar commands

The vision system can analyze frame grabs to produce *visual sonars* data. This method calculates the distance to the nearest visible obstruction, just as sonars caculate the distance to the nearest object which reflects sound. Visual sonars can have high spatial resolution without the problems of cross-talk to which regular sonars are prone. They can also be more directional, and their speed does not depend on the maximum range as regular sonars do. They can recognize some types of obstacles that sonars miss, but can be fooled by things like changes in carpet color which would not fool regular sonars. Therefore both visual and normal sonars are useful.

The distance measurement is based on the assumption an obstacle touches the floor. In order to work, the camera must be angled towards the floor, and the floor must be a relatively solid color. First the image is processed with an edge detection filter (see Section 6.4.4 [Saphira Mode Frame Commands], page 59). Then the system scans up in columns from the bottom of the image to find the vertical position in the frame of the lowest obstruction.

Obstructions are assumed to contrast with the floor. While scanning up from the bottom, the first horizontal line in the column with a large enough value in the edge-filtered image is assumed to belong to the nearest obstacle. You can adjust the sensitivity of the obstacle avoidance by setting the persistent variable `edge_thresh` to a value between 1 and 255. 100 is the default. Lower values increase sensitivity, and higher values decrease sensitivity. (Persistents are viewed and set with the '`P`' command, described in the next section).

These commands display the visual sonars data using the plotting program `gnuplot`. A statically-linked version of `gnuplot` is included with ARC for this purpose. Be sure that it is installed and in your path to use these commands.

These commands are case-sensitive, so be sure to match the case listed here.

'`D`' Grab a frame and display visual sonars data for that frame.

'`DB`' Start continuously updating the visual sonars display.

'`DE`' Stop updating the visual sonars display.

6.4.6 Saphira Mode Persistent Commands

Persistent variables provide a method of preserving calibration variables even if the system is restarted. It is possible to view and modify any persistent variables, not just the ones particular to the Pioneer software, using the following Saphira interaction mode command. See Chapter 8 [Persistent Variables], page 71, for more information on how to use these persistent variables. Commands are case-sensitive, so be sure to match the case listed here.

'`Ppersist[=val]`'
: Query or set the value of a persistent variable.

  For example, to check the sensitivity of the edge detection, send '`Pedge_thresh`' in a PSOS packet (see Section 6.5 [Saphira Packet Protocol], page 61). It will respond with a packet containing '`edge_thresh = 100`'.

  Say that you now want to set the value to 200. Send '`Pedge_thresh=200`' in a packet. It will respond with '`edge_thresh = 200 (was 100)`'.

6.4.7 Miscellaneous Saphira mode commands

'`S`' (capital), '`R`'
: Save or restore current color settings and persistents to EEPROM. See Section 8.3 [Setting and Storing Persistents], page 72.

'`v`' Print software version info

'`q`' Exit Saphira mode, enter TPUVIS prompt mode.

6.5 Saphira Packet Protocol

When the vision system is in Saphira mode, it is controlled using PSOS packets. This is the data format used by the 6811-based robot controller board on the Pioneer and the Saphira robot control program. Here is the exact format of the packets that the vision system accepts and generates while in Saphira mode (see Section 6.4 [Saphira Interaction Mode], page 56).

We assume that you're already familiar with the PSOS packet protocol as described in *Pioneer 1 Software Manual*.

The basic operation of the vision system in Saphira mode is as follows: whenever the host sends a packet to the vision system, if the vision system does not recognize the packet type, the packet is forwarded to the 6811. Whenever the 6811 sends a packet to the vision system, the vision system forwards the packet to the host, and follows it with a `VISIONpac` status packet. In this way, the vision system is almost totally transparent between the host and the 6811 (except for some latency).

6.5.1 Host-initiated Saphira Packet format

The vision system understands two packet formats from the host: `VISION_COM` and `VISIONIO_COM`. The former is subsumed by the latter, and should not be used (it is retained only for backward compatibility).

`VISIONIO_COM` packets contain length-prefixed ASCII strings, in the format generated by the Saphira function `sfRobotComStr`. These strings are just the Saphira commands described above (see Section 6.4 [Saphira Interaction Mode], page 56).

```
#define VISIONIO_COM 100
```

2 bytes	sync chars
1 byte	packet length
1 byte	`VISIONIO_COM` (= 100)
1 byte	`sfARGSTR` (= `0x2B`)
1 byte	string length
N bytes	string
2 bytes	checksum

Each `VISIONIO_COM` command responds with a `VISIONIOpac` packet when it is done. Mostly these are short status messages (suitable for displaying to the user); they are subject to change. (If you'd like to parse them, let us know (see Appendix D [Reaching Newton Research Labs], page 105) and we'll decide on a set that won't be (as much) subject to change.) For now, the main purpose of the return messages is handshaking; you should get exactly one `VISIONIOpac` for each `VISIONIO_COM` you send. Note that the `VISIONIOpac` string is not NUL-terminated.

```
#define VISIONIOpac 0x80
```

2 bytes	sync chars
1 byte	packet length
1 byte	`VISIONIOpac` (= `0x80`)
1 byte	string length
N bytes	string
2 bytes	checksum

6.5.2 Robot-initiated Saphira Packet format

Each time the vision system forwards a status packet from the 6811 to the host (at approximately 10 Hz), it also sends a `VISIONpac`. The mode of each of the three color channels is set by using persistent variables (see Chapter 8 [Persistent Variables], page 71). Each color channel can either be placed in a line-tracking mode or one of two blob-following modes. A `VISIONpac` contains some status information about the current line-tracking settings, followed by a tracking data block for each of the channels.

The format of the tracking data block depends on the tracking mode of the channel. The tracking mode of the three channels are set using the persistents `pioneer_a_mode`, `pioneer_b_mode`, and `pioneer_c_mode`. Currently, the possible values for these persistents are `BLOB_MODE`=0, `LINE_MODE`=1, and `BLOB_BB_MODE`=2.

```
#define VISIONpac 0x50
```

2 bytes	sync chars
1 byte	packet length
1 byte	`VISIONpac` (= `0x50`)
2 bytes	current value of `line_bottom_row` persistent
2 bytes	current value of `line_num_slices` persistent
2 bytes	current value of `line_slice_size` persistent
2 bytes	current value of `line_min_mass` persistent
	channel A tracking data
	channel B tracking data
	channel C tracking data
2 bytes	checksum

6.5.2.1 BLOB_MODE Tracking data

In `BLOB_MODE`, the center of the largest blob in image is calculated and uploaded. If there is nothing of the tracked color in the field of view, the data consists of all zeroes.

1 byte	`BLOB_MODE` (= 0)

4 bytes	area of largest blob
1 byte	X-coordinate of the center of gravity of the largest blob
1 byte	Y-coordinate of the center of gravity of the largest blob

6.5.2.2 BLOB_BB_MODE Tracking data

In `BLOB_MODE`, the center and bounding box of the largest blob in image is calculated and uploaded. If there is nothing of the tracked color in the field of view, the data consists of all zeroes.

In `BLOB_BB_MODE`, the data consists of

1 byte	`BLOB_BB_MODE` (= 2)
4 bytes	area of largest blob
1 byte	X-coordinate of the center of gravity of the largest blob
1 byte	Y-coordinate of the center of gravity of the largest blob
1 byte	X-coordinate of left edge of largest blob's bounding box
1 byte	X-coordinate of right edge of largest blob's bounding box
1 byte	Y-coordinate of top edge of largest blob's bounding box
1 byte	Y-coordinate of bottom edge of largest blob's bounding box

6.5.2.3 LINE_MODE Tracking data

To understand the `LINE_MODE` data chunk, we need to understand the processing the vision system does for `LINE_MODE`.

In `LINE_MODE`, the vision system divides its field of view into horizontal chunks called *slices*. Each slice has `line_slice_size` rows. The bottom slice (slice 0) starts at `line_bottom_row` and extends to `line_bottom_row+line_slice_size`. Each succeeding slice starts `line_slice_size` lines higher. There are `line_num_slices` slices in all.

In each slice, the vision system checks how many pixels in the slice are of the currently-tracked color for that channel; if the number is greater than `line_min_mass`, then the vision system believes it sees the line in that slice.

1 byte	`LINE_MODE` (= 1)
2 bytes	the number of the nearest slice in which the line can be seen
2 bytes	the number of slices in which the line can be seen
1 byte	X-coordinate of the line in the nearest slice in which it can be seen

The idea is to give you an idea of what the line is doing both nearby and in the distance. You can tell what the line is doing nearby so that you can turn in the correct direction. The information about how far away you can see the line lets you slow down for curves.

6.5.3 Special Saphira Packet formats

In addition to the above packet types, some of the Saphira commands return special-purpose packets. The '`G#`' and '`T#%`' commands respond with a color frame upload (a series of `FRAME24pac`'s), the '`E#`' command responds with a grayscale frame upload (a series of `FRAME8pac`'s), the '`D`' command responds with a `DISTpac`, and the '`DB`' command puts the system into a mode where it sends `DISTpac`'s continually (at somewhat less than 5 Hz).

6.5.3.1 Frame Upload Saphira Packet formats

The color and grayscale upload packets are very similar. Each frame upload splits the frame into several pieces, with some number of rows in each piece, and sends one piece in each packet. The packet consists of the following:

```
#define FRAME24pac 0xb0
#define FRAME8pac  0xc0
```

2 bytes	sync chars
1 byte	packet length
1 byte	packet type: `FRAME24pac` (= `0xB0`) or `FRAME8pac` (= `0xC0`)
1 byte	frame height (height of the whole frame, before splitting)
1 byte	frame width
1 byte	beginning row number of this piece
1 byte	number of rows in this piece

1 byte	compression type (see below)
1 byte	number of bytes of pixel data in this packet
	pixel data
2 bytes	checksum

Note that frame uploads use a packet size that's slightly larger than the 200 byte maximum size mentioned in the PSOS packet documentation; this is necessary to allow an entire 64-pixel row to be uploaded in a single packet.

Currently, the compression schemes are very simple; they just throw away low-order bits. There are three compression types defined:

0	no compression
1	keep only the top 4 bits of each byte
2	keep only the top 2 bits of each byte

The remaining bits are packed somewhat oddly in a `FRAME24pac`. The pixel data in a `FRAME24pac` always consists of a byte of red, a byte of green, and a byte of blue, in that order. In compression mode 0 (no compression), these are just the 1-byte values from the board. In compression mode 1, however, each of these bytes contains the bits from two adjacent pixels, and in compression mode 2, each byte contains the bits from four adjacent pixels. (If the frame width is not a multiple of 2 for mode 1, or 4 for mode 2, the frame is effectively padded on the right with zeroes.) When data from multiple pixels is packed into a byte, the leftmost pixel goes to the most significant bits in the byte. The same packing scheme is used in a `FRAME8pac`; of course, there's no color interleaving in that case.

6.5.3.2 Visual Sonar Upload Saphira Packet format

Visual sonar data is sent as a `DISTpac` packet. It has the following format:

```
#define DISTpac 0xd0
```

2 bytes	sync chars
1 byte	packet length
1 byte	`DISTpac` (= `0xD0`)

1 byte	number of bytes of data
	distance data
2 bytes	checksum

Each byte of the distance data is the y coordinate of the nearest obstacle in that direction. Thus, 0 means the obstacle is far away, and 47 means that the obstacle is very close. As a special case, 255 means that no obstacle was found at all in that direction. Currently, the vision system sends 8 byte `DISTpac`'s, but this might be made configurable; it would be a good idea to make your code check the number of bytes of data to make sure it's what you're expecting.

7 Grabbing Frames of Video

In addition to the high-resolution, hardware-assisted tracking of objects on the basis of color, the Cognachrome vision system is capable of low-resolution, 24 bit RGB frame grabs. Standard frame grab resolution is 64x48 pixels.

Although the board's standard software is capable of grabbing frames of video and uploading them to a host for display, frame grabbing is mainly intended for debugging, or for use in systems where the user writes his own software on the board to process the frames. (The communication overhead is simply too great to have good performance in a system which must send every pixel over the serial port.)

7.1 Uploading Frame Grabs through Serial

If you are using the Pioneer vision system software, you can grab and display a frame at 64x48 resolution from Saphira prompt using the '`G`' command. See Section 6.4 [Saphira Interaction Mode], page 56, for details.

To grab and display a frame at 64x48 resolution from the TPUVIS Prompt, do the following:

`tpuvis[a]> c`	*Enter advanced color menu*
`colorplot> g`	*Grab a frame into buffer*
`colorplot> u`	*Upload a frame in hex over serial port*

The frame data will now be uploaded over the serial port.

`colorplot> q`	*Exit advanced color menu*
`tpuvis[a]>`	

The frame information is sent over stream 30 in the following format:

`V 64 48 rgb`	*64 cols, 48 rows, R, G, and B data*
`H 0 675a5b 6a5c5b ... 2f2f1b`	*Horizontal line #0, RGB data*
`H 1 6a5c5b 675a4f ... 2f2f1b`	*Horizontal line #1, RGB data*
`...`	
`H 47 604222 5a3f22 ... 0f0d05`	*Horizontal line #47, RGB data*
`Z`	*End of frame*

7.2 Writing On-Board Programs to Process Frames

See Chapter 9 [Programming the Vision System], page 77.

7.3 Higher Resolution Frame Grabs

The hardware limitation of 64 pixels per line can be overcome in two ways: interleave, and removing color channels.

In order to interleave, one divides the Y resolution by two to double the X resolution. Since the Y resolution can be as high as 250 (every single line), this often isn't a major problem. This interleave is performed by grabbing adjacent horizontal lines with a half-pixel offset. The two adjacent horizontal lines are then merged into a single horizontal line. Resolutions as high as 130 x 125 can be obtained using this method.

The hardware Cognachrome Vision System is capable of doing somewhat higher resolution grabs if you are willing to give up color channels. The first step is to move from 24 bits to 16 bits per pixel, which allows you to grab either red and green, or blue and green for each pixel. The second step would be to choose a single color channel (red, green, or blue) and only take 8 bits per pixel.

Note that in 8-bit mode the board can effectively grab a black-and-white image by simply reducing the color saturation adjustment on the NTSC to RGB daughterboard, and then grabbing one of the three channels.

None of the scenarios described below have been implemented in the current software. Contact Newton Labs if you need such capabilities (see Appendix D [Reaching Newton Research Labs], page 105).

8 Persistent Variables

The vision system uses several persistent variables. These variables are stored in a special place in memory which is not rewritten when the system is reset. In addition, they can be stored in the serial EEPROM on the vision board so they can survive power down (see Section 8.3 [Setting and Storing Persistents], page 72). Many behaviors of the vision system and ARC can be modified through use of persistents.

Some of the persistents pertaining to vision processing are set as side effects of various commands. For example, the filter level and startup behavior can be changed using special TPUVIS commands. However, there are some less commonly adjusted, though interesting persistents which must be set by name using the following commands.

8.1 Listing All Persistents

From TPUVIS prompt mode

Type '`P`', then type '`?`' RET at the TPUVIS prompt. If you decide not to change anything at this time, hit RET to exit. See Section 6.3.4.3 [Debugging and Miscellaneous Detail], page 55.

```
tpuvis [a]> P
Set which persistent? (? for list) ?
Persistents table:
          orient_subsamp = 0:
               baud_rate = 9600:
                    ...
              color_grow = 5:
Set which persistent? (? for list)      [hit RET]
tpuvis [a]>
```

From Saphira interaction mode

NOT AVAILABLE

From the ARC prompt

Use the ARC '`persist`' command.

```
arc> persist
          sony_sharpness = 127:
            serialb_baud = 9600:
                 boxsize = 20:
                .....
             frameborder = 5:
```

8.2 Examining a Particular Persistent

From TPUVIS prompt mode

Type 'P', then type the name of the persistent followed by RET at the prompt. This brings up a prompt at which you could type a new value. Just hit RET again to leave the value the same. See Section 6.3.4.3 [Debugging and Miscellaneous Detail], page 55.

```
tpuvis [a]> P
Set which persistent? (? for list) color_grow
New value for color_grow [currently 5]:
```

From Saphira interaction mode

Send a PSOS packet containing `P<name>`. For example, to check the sensitivity of the edge detection, send '`Pedge_thresh`' in a PSOS packet (see Section 6.5 [Saphira Packet Protocol], page 61). It will respond with a packet containing `edge_thresh = 100`. See Section 6.4.6 [Saphira Mode Persistent Commands], page 60.

From the ARC prompt

Type '`set` *persistent*'.

```
arc> set color_grow
color_grow = 5
```

8.3 Setting and Storing Persistents

From TPUVIS prompt mode

Type 'P' at the TPUVIS prompt, then type the name of the persistent followed by RET at the prompt. This brings up a prompt at which you can type a new value (followed by RET). See Section 6.3.4.2 [Training and Configuring Channels Detail], page 52.

```
tpuvis [a]> P
Set which persistent? (? for list) color_grow
New value for color_grow [currently 5]: 10
```

From Saphira interaction mode

Send a PSOS packet containing the string '`P<persistent name>=<value>`'. For example, if you want to set the value of `edge_thresh` to 200, send '`Pedge_thresh=200`' in a packet. It will respond with `edge_thresh = 200 (was 100)`. See Section 6.4.7 [Miscellaneous Saphira mode commands], page 61.

From the ARC prompt

Type '`set` *persistent value*'.

```
arc> set color_grow 10
color_grow set to 10
```

To store the current values of all persistents to the serial EEPROM on the vision system, use the capital ’S’ command. This works in either TPUVIS or Saphira interaction mode. Even without using this command, parameters will persist across resetting the system. However, persistent variables are not guaranteed to persist across power cycling the system unless saved to EEPROM.

This is true regardless of what mode the persistent was changed from. Therefore, while you can examine and modify persistents from the ARC prompt even if the vision software is not currently loaded, you can only save the parameters to the serial EEPROM when the vision software is loaded and running.

8.4 Important Persistents

Here are some important persistents. This is an incomplete list; see Section 8.1 [Listing All Persistents], page 71, for information on getting a complete list of persistents (without documentation).

`baud_rate` (default 38400).
: This controls the baud rate initially used for port A on system power-up. This is usually set as a side effect of the `baud` ARC command, or by a call to `set_baud_rate()` in the code. The Pioneer software will set this to 9600 every time the code is run (i.e., every reset and every time the program is run from the command line).

`boxsize` (default 20).
: For each frame of incremental color training, a small box centered around the RGB value of the center of the tracked target is added to the training volume. The size of this box is set by the `boxsize` persistent value. The larger this size is, the greater the effect of each incremental training frame. See Section 5.2.2 [Training the Vision System to See a Target Color], page 28, for more information.

`color_grow` (default 5).
: This affects ‘`<`’ and ‘`>`’ color training commands. The value controls how much is removed from or added to the edges of the thresholding set each time one of these commands is issued. Setting this value is somewhat analogous to choosing what grit of sandpaper to use. Increasing this number makes each command have a larger effect, and decreasing this number makes each command have a smaller effect. See Section 5.2.2 [Training the Vision System to See a Target Color], page 28, for more information.

`diffthresh` (default 5000).
: This affects the lower case ‘`s`’ color training command. The value specifies how different a color can be from the center color and still be considered to be part of the same object.

Increase this value if the system is not classifying a large enough region as belonging to the training target. Decrease this value if the system is classifying too large a region as belonging to the object. See Section 5.2.2 [Training the Vision System to See a Target Color], page 28, for more information.

`orient_subsamp` (default 0).
: This determines whether subsampling is performed on the edge data for purposes of computing angular orientation and aspect ratio. Since angular orientation determination is a relatively costly procedure, you might need to subsample the data if you wish to maintain 60 Hz performance with large numbers of edges (i.e. blobs that fill the screen vertically). This parameter does not affect other calculations, such as calculations of centroid or area.

  The options are:

  - 0) Adaptive subsampling. This is the default, and is recommended. Using this method, the software will adaptively subsample to reduce the number of processed edges to be less than 100.
  - 1) No subsampling. Orientation calculation uses every single data point in each tracked object for calculations.
  - 2) 50% sampling. Half of the edges will be chosen at random in each tracked object for use in orientation calculation.
  - 4) 25% sampling. One quarter of the edges will be chosen at random in each tracked object for use in orientation calculation.

`serialb_baud` (default 9600).
: This controls the baud rate initially used for port B on system power-up. Since port B is implemented using the 68332's TPU, this port is unreliable at speeds over 9600 baud (see Section 4.2.2 [Using the RS-232 Ports], page 17).

Sony Digitizer Settings
: These values affect the way the Sony digitizer converts the analog NTSC video from the camera into digital RGB values. If set using the regular method of setting persistents, these values will not take effect until the next reset. Use the TPUVIS '`A`' command to set them with immediate effect. To see the effect of changing these values, look at frame grabs of a fixed scene before and after the change.

  `sony_apl` (default 127, range 0 to 255).
  : This controls the amount of auto gain the Sony NTSC digitizer module will use. Decrease this number to make the Sony digitizer perform less autogain, and increase it to make it do more.

`sony_hue` (default 127, range 0 to 255).
: This modifies the hue adjustment applied by the Sony NTSC digitizer. Changing this will affect how a given color phase in the NTSC video input is converted to RGB hue. Generally you should not change this value.

`sony_saturation` (default 127, range 0 to 255).
: This modifies the saturation adjustment applied by the Sony NTSC digitizer. Changing this will affect how saturated the colors are when digigitized. At a value of 0, you get a grayscale image (this will not interact well with color thresholding). At a value of 255 colors are extremely saturated. High saturation is generally good for thresholding—however, if this is set too high, normally unsaturated colors such as browns and grays will show up as saturated colors (such as orange and blue) and interfere with tracking only the truly saturated colors. For this reason, we generally keep this setting at 127.

`sony_sharpness` (default 127, range 0 to 255).
: This modifies the sharpness adjustment applied by the Sony NTSC digitizer. Changing this will affect how crisp or fuzzy the image appears.

9 Programming the Vision System

Many users never have the need to change the board's built-in vision algorithms. But some of you may want to add to or modify the vision algorithms. And perhaps more will wish to add code to directly drive robots or other equipment through the extra digital I/O.

The vision software is set up to allow easy access to the various levels of processing; if you simply want access to the results of the current algorithms to control your robot base, you don't need to understand the internals of the algorithms. If you want to delve deeper and compute a new metric for a tracked object, you can easily get the list of "segments" from an individual object. Or if you want to go deeper still, you can process segments as they come in from the timer processor, before they have been sorted into various tracked objects.

The current interface to the vision algorithms is still undergoing changes, and has not been completely documented yet. Please contact us for help and information to get started. Or, if you're not interested in programming the vision system, you may be able to contract with us to do the work. See Appendix D [Reaching Newton Research Labs], page 105.

Appendix A Mini-ARC Installation Guide

A.1 Create a Main Directory for Mini-ARC

The first step in installing Mini-ARC on your system is to decide on a place in your directory structure for the programs. By following these guidelines, it will be easy to replace Mini-ARC with a full ARC installation if you decide to at some point.

- If you are installing on a stand alone machine or a small network, we recommend making the directory '`/usr/local/arc`' and using that as the primary directory for ARC.
- If you are on a large network and ARC is being used by multiple people in your project, it may be more appropriate to install arc in your project's main directory. This might be something like '`/projects/walker/arc`'.
- If you are the only person using ARC, you may want to put the main ARC directory in your home directory (as in '`~/arc`').

These suggestions are more for aesthetic purposes than functional ones. So long as your path is set up properly, proper functioning of the software is not affected by where the main ARC directory is located. However, it is good practice to keep software packages which are used by multiple people in a more central location than someone's home directory. On unix systems, '`/usr/local`' is a standard place to put such packages, so that will be the default in examples in the rest of the manual.

Once you have decided where the main '`arc`' directory should be, make the directory (ex. `mkdir /usr/local/arc`) and give it appropriate permissions (ex. `chmod a+rx /usr/local/arc`). This will be the directory in which the subdirectory for the Mini-ARC binaries for each platform you use will be placed.

A.2 Decide on Which Package You Need

The following is a list of the platforms to which Mini-ARC has been ported as of June 1, 1996:

- Sparc running SunOS
- IBM PC or compatible running Linux
- IBM RS6000 running AIX

- HP700 running HPUX
- NeXT hardware running NeXT OS

If your computer does not appear on the list, please contact Newton Labs (see Appendix D [Reaching Newton Research Labs], page 105). So long as it is possible to build a GCC cross compiler for your host platform, it is possible for us to build ARC for your needs. In general, any machine running UNIX will probably be acceptable as a host platform. Windows 95 and NT will work as a host platform soon. Macintosh, DOS, and Windows 3 are not likely to work as host platforms.

A.3 Copy a Mini-ARC Package to your Site

Mini-ARC is distributed over the Internet and can be copied using any Web browser. The programs available over the network are initially disabled, and require a license key before they will run. You should have received a license key for Mini-ARC with your vision system. If this is not the case, contact your dealer or Newton Labs (see Appendix D [Reaching Newton Research Labs], page 105).

Use your web browser to go to the ARC home page at '`http://www.newtonlabs.com/arc`'. Links from that page will get you to the ARC download site, as well as online version of the ARC manual and other related information. Follow the link to the ARC download page.

Select the appropriate Mini-ARC package, as discussed above. Instructions on the download page will help you to get the most current version. You may also want to check this site periodically for upgrades. Your original key will work with such upgrades, and you can follow these same instructions to install upgrades as well as the initial installation.

Click on the appropriate packages, and tell the browser to copy them to your main '`arc`' directory (see Section A.1 [Create a Main Directory for Mini-ARC], page 79).

A.4 Extract the Software from the tar Archive Files

The packages available for download are compressed tar archive files. These should be extracted in the main '`arc`' directory (see Section A.1 [Create a Main Directory for Mini-ARC], page 79). They will expand to create separate directories for each host platform, so you do not need to do anything special to install software for multiple host types.

The following example shows how to install ARC (this this case, version 2.0 for the Sparc). To install ARC, you would move the files, uncompress them, untar them, and remove the archived versions. At the end of this example, you would have created and filled the directory '`/usr/local/arc/sparcbin`' with the Mini-ARC binaries.

```
mv mini-arc_sparcbin-2.0.tar.Z /usr/local/arc
cd /usr/local/arc
uncompress *.Z
tar xvpf mini-arc_sparcbin-2.0.tar
rm mini-arc_sparcbin-2.0.tar
```

A.5 Install the license key

Once you have installed the distribution package, you need to install the license key before the software is enabled. This procedure requires that you have write access in the binary directory (ie. '`sparcbin`', '`linuxbin`', etc), and that you use a machine of the appropriate architecture. For example, it is not possible to install the key for the Sparc version while running on a Linux machine—you would have to be on a Sparc. If you do not have write permission in the directory, you may need to contact someone who does before continuing.

License keys have two parts: a string specifying the person or company which Mini-ARC is licensed to, and a numeric sequence containing the actual key. *Both parts must be entered correctly for the software to be enabled.*

Go into the appropriate binary directory and type `arc -config`. This will put you in Mini-ARC configuration mode. There are many configuration parameters which may be set in this mode, including setting the license key information.

Press return until you are prompted with the line:

```
Person, Company ARC is licensed to:
```

Backspace to the start of the line and type the text part of the key. Note that capitalization, whitespace, and punctuation are not significant. When you have finished, press return. You will next be prompted with the line:

```
License key:
```

Again, backspace to the start of the line and type the numeric part of the key. This will be a string of numbers separated by hyphens.

Other configuration options may be safely ignored for now. Hit return at each and they will retain their default values.

Run Mini-ARC again, but this time without the `-config` flag. If the key was installed properly, it will say:

```
Single user license for: Jane Doe
License key: XXXXX-XXXXX-XXXXX-X

Port /dev/cua0, baud 38400
    Terminal port # 9041
Term socket = <fd 4>
arc>
```

If the license was not installed properly, or was not a valid Mini-ARC license, it will say something like:

```
Missing license key.  Please contact Newton Research Labs for key, and install
keys by typing "arc -config" from arc's home directory.
```

Then it will exit. If this happens, check that you typed both parts of the key accurately. If the problem persists, contact `arc@newtonlabs.com`.

A.6 Set up your environment to use ARC

Finally, in order for Mini-ARC to work properly, you will need to add the appropriate binary directory to your path. Examples of setting the path for a Sparc installation of Mini-ARC are shown below. You should replace '`/usr/local/arc/sparcbin`', with the path appropriate to your architecture. Note that setting the path to include ARC will only affect the particular shell you execute it in—the path in other shells will not be affected

From csh or tcsh

set path=($path /usr/local/arc/sparcbin)

From bash

PATH=$PATH:/usr/local/arc/sparcbin

You need to put this path extension command in your file which handles automatic path initialization. If you do not know the appropriate file, here are some possibilities:

For csh '`~/.cshrc`'

For tcsh '`~/.cshrc`' or '`~/.tcshrc`'

For bash '`~/.profile`'

These changes to the initialization files will not take effect in pre-existing shells. If you are using '`csh`' or '`tcsh`' as your shell, you can load the changes into your current shell by using the `source` *filename* command. To change the path in all of your shells, you can kill and restart all the shells, or log out and back in again.

To check if you have set up your path, `cd` to any directory not containing the `arc` executable and find out whether it would work to invoke ARC from there:

From tcsh or csh

```
which arc
```

From bash

```
type arc
```

If it says the path you expected, you are done. Otherwise, recheck your path. This must work before Mini-ARC will work properly—invoking Mini-ARC with the executable in the current directory is not sufficient because it will spawn other processes which also need the path set properly.

Appendix B Troubleshooting

If your system seems not to be working like you want it to, you might find a solution to your problem in this appendix. We've tried to include as much of our experience as we can, but if you run into problems and you don't find the help you want from this section, please contact us (see Appendix D [Reaching Newton Research Labs], page 105).

B.1 Status Lights and Beeper

You can find out a lot about what's going on with your board by looking at the nine LED's and listening to the beeper. For a diagram showing the names and locations of the various status indicators, see Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

The beeper should beep during power-up or reset. If it doesn't, you might have a power problem, a blown fuse, a problem with your ROMs, or a serious hardware problem with your vision system.

The three red LED's indicate the status of the three power sources (digital +5v, analog +12v, and analog +5v). All three should be lit. (Do not run the board for extended periods of time without all three power inputs supplied as it will damage the board). If you see an LED which isn't lit, it probably indicates your power supply isn't connected properly (or maybe isn't turned on). If the digital +5v LED (leftmost in the bank of 6 LED's) doesn't light but its power supply seems to be working, you may have blown a fuse. See Section 4.4.2 [Digital Power Connector], page 22. For more information on tracking down analog power wiring problems, see Section 4.4.3 [Analog Power Connector], page 23.

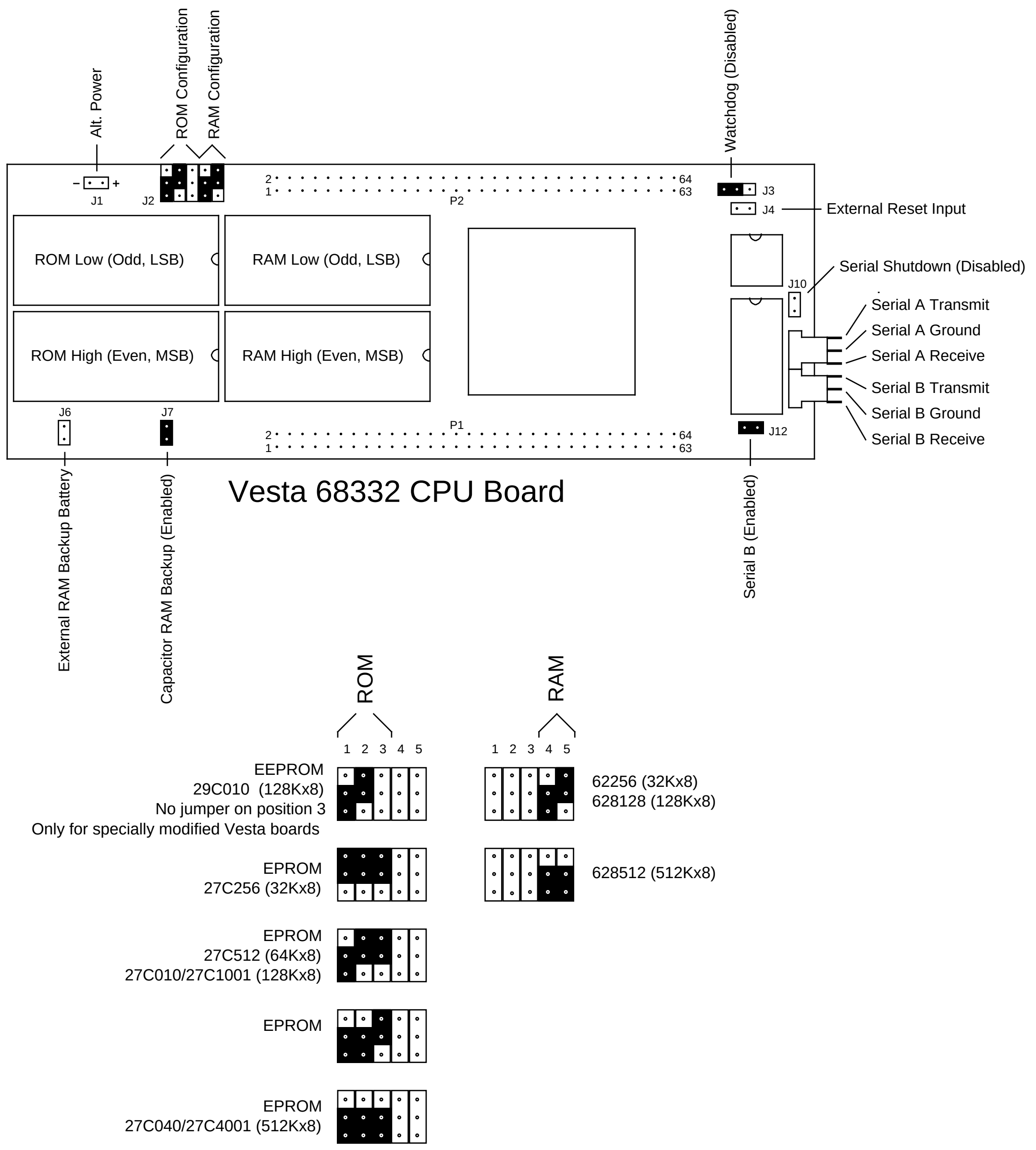
The green LED closest to the video connections tells whether horizontal and vertical sync signals are being detected from the incoming video. If this light isn't on, it's probably because your camera is off or disconnected.

The group of three green LED's are connected to the video recognition channels—you can tell by looking at them if each of the three channels is detecting anything.

The group of two yellow LED's are connected to internal lines on the board. As part of a normal boot sequence, you should see them flicker for around 5 seconds.

B.2 Processor Board Jumper Configuration

Hopefully you received a properly configured processor board with the jumpers all in the right places. However, if the processor board does not work quite right, it may not be configured properly. Check the jumper configuration against the following diagram:



Caution: Any Vesta board modified to use EEPROM rather than EPROM must use the 128Kx8 EEPROM jumper settings, regardless of whether EEPROM or EPROM is installed. Do not install a jumper in position three of J2 on these boards or damage may result.

B.3 Serial Port Troubleshooting

Before you can sucessfully interact with the vision system, you'll need to know the name of the serial port you intend to use. This is a multi-step process. First you must determine what the names of the serial ports are on your system. Once you have determined this, you must determine what name corresponds to the port you physically hook your cable to. It may also be necessary to change the permissions on the serial port before you can use it. This may require you to contact a system administrator if you do not have root access to your machine.

B.3.1 Serial Port Names

The device names for serial ports vary from system to system. Here are some typical serial port names, but realize that names sometimes vary even between different machines of the same architecture, so the following are suggestions for names to try, rather than the final answers.

- Linux: '`/dev/ttyS0`', '`/dev/ttyS1`', ... or '`/dev/cua0`', '`/dev/cua1`', ...
- SPARC running SunOS: '`/dev/ttya`', '`/dev/ttyb`'
- RS/6000: '`/dev/tty0`'
- NeXTStation: '`/dev/ttya`', '`/dev/ttyb`'
- DECstation: '`/dev/tty00`', '`/dev/tty01`'

The serial ports should show up in the file system, so you can do an `ls` in the '`/dev`' directory and see if you see any of the names listed above appear.

B.3.2 Serial Port Acess Privileges

Once you have determined a likely candidate for the name of the serial port, check that you have both read and write permission to that port.

```
quasar% ls -l /dev/ttyS*
crw-rw-rw-   1 root     tty        4,  64 Jul  5  1995 /dev/ttyS0
crw-rw-rw-   1 root     tty        4,  65 Feb  7  1995 /dev/ttyS1
```

The set of letters and dashes at the left side shows the access privileges for each port. If there are dashes instead of `rw` in any of the three positions above, you may not have enough access to use the serial port.

If you have root access, you can do the following to set the privileges properly. If you do not have root access, contact a system administrator:

`quasar:/usr/local/arc%` *chmod a+rw portname*

Now if you repeat the `ls -l` command, it should look like the example above.

B.3.3 Determining which Serial Port to Use

On some machines, the serial ports are labeled in an intuitive manner. For example, a Sparc station generally has two serial ports named '`/dev/ttya`' and '`/dev/ttyb`', and on the side of the computer are two serial ports—one labeled with an **A**, and the other labeled with a **B**.

On Linux machines, sometimes the ports are labeled as **COM1**, **COM2**, etc. If this is the case, then generally `/dev/ttyS0` or `/dev/cua0` corresponds to **COM1**, `/dev/ttyS1` or `/dev/cua1` corresponds to **COM2**, etc.

If which port to use seems unclear, it should still be possible to figure out experimentally which one to use by using loopback cable. The basic idea is that you connect the transmit and receive lines together on the serial port you plan to use. Then you send data out to a likely port name, and listen to see if you can hear yourself. If you do hear yourself with the loopback in place, but do not hear yourself when the loopback is removed, then it is likely that you have the right port.

B.3.3.1 Installing a Serial Loopback

Follow the instructions corresponding to the type of connector you are using. The loopback can be installed either directly at the back of the host computer, or at the end of a serial cable. Generally, you should start by installing the loopback as close to the computer as you can. Once that works, you can move the loopback further out towards the end of the cable. If it still works at the end of the cable which will plug into the vision board or robot, then you know that your cable is alright (with the possible exception that it may still need a null modem—this procedure does not test that).

DB-25 or DB-9

These are common connectors on Sparc, DEC, and IBM PC compatible computers. Both are shaped like a trapazoid with one more pin on the top row than the bottom row.

To loop back these connectors, connect pins 2 and 3 together. Usually there are small numbers printed next to each pin. If this is not the case, look at the connector with the larger side up. Pin 1 starts at the top left for male, and top right for female connectors. Pin 2 is beside pin 1 on the same row. Numbering restarts on the bottom row starting from 13 for DB-25 or 6 for DB-9 below pin 1.

If the connector is female, use a piece of solid core wire. Be careful about the gauge of the wire—too thick may damage the connector, and too thin may not make a reliable contact. If the connector is male, you can use a jumper lead or have a friend hold a wire in place.

8-pin Mini DIN

These connectors are round, about 1/4" in diameter, and have 8 tiny pins. They are common for serial ports on Macintosh and NeXT computers, as well as recent Sparc models. Do not try to deal with these connectors directly. Instead buy a DB-25 adapter cable and follow the DB-25 directions. These cables generally cost around five dollars and are sold as Macintosh modem cables. DO NOT buy a Macintosh *printer* cable—transmit and receive (pins 2 and 3) are swapped on these cables. If you only have a printer cable, you need to buy a null modem. These are available from just about any computer store.

3-pin Molex Connector

These are the connectors used for the serial ports on the Vesta 68332 board (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). If you are using a stand alone vision system, you should have received a cable with one of these connectors at one end. Connect the outer two pins together for loopback.

RJ-11 (Phone Cable) Connector

If you are using RJ-11 with adapters for serial and it is not convenient to use an adapter for loopback using the above instructions, you can loopback the RJ-11 connector directly. Have a friend hold a fine wire against the outside two of the 4 metal contacts. (If you are using a similar cable which has 6 or 8 metal contacts, use the outer two of the center 4 contacts.)

B.3.3.2 Finding the Right Serial Port

After installing the loopback on the physical port you plan to use, you are ready to send data to various port names until one echoes what you send. Any serial interaction program which allows you to specify the port name will work for this task.

Using ARC

Start ARC in the following manner:

```
arc -port portname -gdb_inhibit
```

The `-gdb_inhibit` flag is to prevent it from querying the status of the board.

If you do not use the `-gdb_inhibit` flag, it will automatically send a packet to itself every second or so. If you try this on the port which has the loopback installed, it will complain about the format of the response.

```
got bad gdb packet: >$H2,0:wsb<
got bad gdb packet: >$H2,0:wsb<
...
```

This is a way to have ARC automatically talk to itself, but it is probably more satisfying to type yourself.

When ARC starts up, it enters prompt mode. In this mode, ARC waits until you time a whole line followed by RETURN, then decides what to do with what you have typed. In order to send and receive characters directly to and from the serial port, type '`^x`' (control-x). This will put ARC in terminal mode, which allows you to type directly to the serial port.

When you are done, type '`^x`' again to go back to the `arc>` prompt. Then type `quit`.

Using Kermit

Kermit is a standard serial communications program installed by default on most UNIX systems. By performing this test using kermit, you can verify that any serial problems you may have are not ARC's fault.

Invoke kermit in the following manner:

```
kermit -l portname -b 38400 -c
```

Note: Do not be concerned if you see everything fine except that the cursor does not move to a new line when you press RETURN. This is normal behavior for kermit.

When you are done, type the escape character (typically '`^\`') followed by `c` to quit.

If you did not see anything echoed, try a different value of *portname* and try again.

If you did see what you typed echoed, you may want to remove the loopback and verify that the port no longer echoes. This is to make sure that you are not just talking to another device which happens to echo.

Write down the value of *portname* you used, and use the `-port` *portname* flag when running ARC in the future using that port. Even better yet, write the name on a sticker and put it right next to the port on the computer so no one will have to do this again.

B.3.3.3 Setting the Default Serial Port for ARC

If you are the sole user of ARC at your site, or if you know that everyone will want to use the same port, you can set this to be the default serial port for when no `-port` flag is used.

This procedure requires that you have write access in the binary directory (ie. '`sparcbin`', '`linuxbin`', etc), and that you use a machine of the appropriate architecture. For example, it is not possible to change the default serial port for the Sparc version while running on a Linux machine—you would have to be on a Sparc. If you do not have write permission in the directory, you may need to contact someone who does before continuing.

Go into the binary directory containg the ARC binary and type `arc -config`. Backspace over the old value for the default serial name, and type the name you just determined was appropriate.

```
quasar% which arc
/usr/local/arc/linuxbin/arc
quasar% cd /usr/local/arc/linuxbin
quasar% arc -config

ARC version 1.4.1  (18-May-1996)
Copyright 1995, 1996 Newton Research Labs
   by Randy Sargent, Anne Wright, and Carl Witty
   For information or help, please see http://www.newtonlabs.com/arc
   Email: arc@newtonlabs.com

Backing up arc to arc.bak
Done
Configuring 'arc'.
For each option, you can press 'return' to leave item same,
or you can edit the item (by either backspacing the old item and typing in
the new, or using emacs-style control keys).

Default serial device name (in the form /dev/?):  /dev/ttyS1
Person, Company ARC is licensed to:  xxxxxxxxxx [RET to leave the same]
License key:  xxxx-xxxxx-xxxxx-x [RET to leave the same]
```

B.3.4 Serial Port Usage Conflicts

A common problem encountered with using serial ports is that it does not work for more than one program to read a serial port at the same time. Unfortunately, it is not always clear what programs are using a given port. This section provides strategies for dealing with this problem.

B.3.4.1 Symptoms of Serial Port Usage Conflicts

There are two categories of symptoms which are caused by another program using the same serial port that you are trying to use:

Locking Conflicts

Some systems perform locking on the serial ports. This means that the first program which tries to open a given port succeeds, but any other programs are denied access. When a serial port is locked and you try to run ARC, you will get a message like this: `Could not open port` *portname* `for input`. ARC will then print usage information and exit.

Locking is in general a good thing, and it is not as confusing as contention without locking, but it causes some problems:

- The symptoms are difficult to distinguish from lack of read/write access to the port. You can tell determine conclusively that it is or is not an access problem by doing `ls -l` *portname* (see Section B.3.2 [Serial Port Access Privileges], page 87).
- Sometimes there really is not another program which has the lock, but the system will not release the lock. When this happens, it generally requires a reboot to fix.

  If you are using Linux, it may work to use a `cua` port instead of a `ttyS` port, as the locking mechanisms are somewhat different between the two. If you were using `/dev/ttySN` (where `N` is a number from 0 up) and then experienced locking problems, you can try `/dev/cuaN`. This is generally an alias for the same port with different locking mechanisms.

Non-locking Conflicts

When a system does not perform locking on serial ports properly, it will allow more than one program to connect to the serial port at the same time. Generally this will work reasonably on output: all output sent by any of the programs will be sent (it is, of course, probably confusing to the device at the other end if they talk at the same time). However, on input it will give the serial data received to whoever asks first.

A good test for this condition is to reset the board and look at the boot message. If it seems to be missing chunks of letters, then there is probably another program getting those letters. See Section 6.1.1 [Startup Behavior], page 43, for information on what the boot message should look like.

If you see these symptoms, you have the following options:

If you know which program conflicts and can exit

If you control the conflicting program, then the solution is easy—quit it. This is the most common case.

If you know which program conflicts and cannot exit

Sometimes you started the conflicting program, but cannot quit normally, either because it was started from a different machine or beacuse it crashed. In this case, you should use the `ps` command (see below) to determine the process ID, *pid*, of the program, then kill it using `kill` *pid*. Use `ps` again to make sure it is dead. If it is not dead, you can kill it more insistently by using `kill -9` *pid*.

If someone else started the program, you probably do not have permission to kill the process. In this case, you should contact the person who started the program, or a system administrator, and ask them to quit or kill the program.

If you do not know which program conflicts

Sometimes you cannot figure out what program is already using the port. In this case, use the methods listed later in the section. Once you have determined what program it is, follow the instructions above.

If you cannot determine which program conflicts

This is the worst case. This may be solved by contacting someone else who has more experience with the particular machine involved, such as a system administrator, who can find what is conflicting. Sometimes, however, there is simply no program which is conflicting and the locking is in a bad state. In this case, the machine may need to be rebooted. *Do not try this with a UNIX machine unless you know what you are doing—contact someone who does.*

B.3.4.2 Determining what Programs are using a Serial Port

If you are experiencing the symptoms of serial port usage conflicts (see Section B.3.4.1 [Symptoms of Serial Port Usage Conflicts], page 92), the first thing to try is to look at the process table for other programs which are likely to be using the serial port.

Determining Serial Port Usage with ps

The most general method for finding what programs may be using a serial port is to use the `ps` command. It lists what processes are running, as well as various statistics about it. This method has the advantage that it is available on all UNIX machines, and it does not require you to have root access, but it has the disadvantage that it is not very specific and therefore is hard to use accurately.

Note that the flags to `ps` vary somewhat from machine to machine (in particular, System V and BSD derived systems differ in this respect), so you may want to do `man ps` to determine what flags to use if the example below fails. In the following example (which will work for Linux and SunOS), the flags tell it to list the processes for all users ('`a`'), list which user owns the process ('`u`'), list all processes ('`x`'), and use multiple lines if necessary to list all the arguments ('`w`').

```
quasar% ps auxw
rsargent 17024  1.7  8.2  6276  5172  pf S 07:11 1:47 emacs
rsargent 17451  0.0  0.5  2072   328  pa S 07:33 0:00 arc -port /dev/cua1
rsargent 17866  0.0  0.6   864   400  pf R 08:52 0:00 ps auw
...
```

Typically this list will continue for a couple of pages, so it may be difficult to look through all of it. Therefore it is a good idea to pipe the result of the `ps` command into `grep`, which will filter the output according to a pattern and only print the lines which match.

```
quasar% ps auxw | grep arc
rsargent 17451  0.0  0.5  2072   328  pa S 07:33 0:00 arc -port /dev/cua1
rsargent 17889  0.0  0.4   968   260  pf S 08:56 0:00 grep arc
```

In this case, the argument to `grep` made it look for all processes containing the sequence of charaters `arc`. It found that user `rsargent` was running `arc` on port `/dev/cua1`. (The `grep` process usually appears too—don't let that worry you.)

Determining Serial Port Usage with fuser

The `fuser` command is a more specific and accurate method of determining serial port usage. Unfortunately, it is not available on all systems, and it requires root access for accurate results. If you have the `fuser` command on your system and have enough privilege to use it, it is the best way to go.

To use `fuser`, first become root, and then call `fuser` with the port you are interested in as an argument. It will tell you what process or processes are using that port. Then you can use `ps` to find out more about the relevant processes.

```
quasar# fuser /dev/cua0
/dev/cua0:         13702 13705
quasar# ps auxw 13702 13705
USER     PID %CPU %MEM  SIZE RSS  TTY STAT START TIME COMMAND
anarch 13702  0.0  0.5  2072 328  p9 S    14:55 0:00 arc -port /dev/cua0
```

```
anarch 13705  0.0  0.2  340  168  q6 S    14:55 0:00 simple_telnet
```

In this example, fuser determined that processes 13702 and 13705 were connected to '/dev/cua0', the serial port I want to use. By running ps, I determined that user anarch is running arc on that port already (simple_telnet is a helper program started by arc).

B.3.4.3 Disabling getty if it is Running on a Serial Port

getty is a program which runs on a serial port to allow users to log in remotely over the serial port. This interferes with using the serial port for anything else. getty is usually set up to run automatically on startup. Therefore it requires special methods to find and disable it. It is somewhat common for a getty to be running even years after it was set up, long after everyone who knew about it has forgotten or left.

If you do not have root access on your system, or if your system does not conform to the directions below, you should contact your system administrator for help.

Under System V derivatives (Linux, HP-UX, etc.)

Look in '/etc/inittab' for lines like this:

```
s2:45:respawn:/sbin/agetty 9600 ttyS0 vt100
```

In this example, a getty may be running on '/dev/ttyS0'.

To disable the getty, do the following:

- Comment out the relevant line or lines out by inserting a '#' at the beginning of each line. Make sure the lines you comment out contain serial ports you recognize; otherwise you might comment out a getty on a console login, which could be bad.
- To make init re-read '/etc/inittab' and disable getty, become root and type kill -1 1.

Under Ultrix, SunOS, and some other BSD unixes

Look in '/etc/ttys' for lines like this:

```
tty00   "/etc/getty std.9600" vt100     on nomodem
```

In this example, a getty may be running on '/dev/tty00'.

To disable the getty, do the following:

- Disable the relevant line or lines out by inserting a '#' at the beginning of each line, or in the case there is a field that says "on", you can change it to "off". Make sure the lines you change contain serial ports you recognize; otherwise you might comment out a getty on a console login, which could be bad.
- To make init re-read '/etc/ttys' disable getty, become root and type kill -1 1.

B.3.5 Serial Port Pinouts

DB-25 These are common connectors on Sparc, DEC, and IBM PC compatible computers. They are shaped like a trapazoid with 13 pins on the top row and 12 on the bottom row. Usually there are small numbers printed next to each pin. If this is not the case, look at the connector with the larger side up. Pin 1 starts at the top left for male, and top right for female connectors. Pin 2 is beside pin 1 on the same row. Numbering restarts on the bottom row starting from 13 below pin 1.

Computers that use DB-25 connectors will wire their ground on pin 7, and their transmit and receive lines on 2 and 3. Which is on 2, and which is on 3, depends on the computer.

IBM PC's, DECstation 5000's, and Macintoshes with modem cables will wire the computer's transmit to 2, and the computer's receive to 3. (You should, of course, connect the computer's transmit to the board's receive, and vice versa). In general, when your cable doesn't work, there's a good chance you've just swapped pins 2 and 3. Either try rewiring the cable, or insert a null modem in the cable (this swaps 2 and 3, as well as some of the handshaking lines, which the vision system doesn't care about).

DB-9 These are common connectors on IBM PC compatible computers. They are shaped like a trapazoid with 5 pins on the top row and 4 on the bottom row. Usually there are small numbers printed next to each pin. If this is not the case, look at the connector with the larger side up. Pin 1 starts at the top left for male, and the top right for female connectors. Pin 2 is beside pin 1 on the same row. Numbering restarts on the bottom row starting from 6 below pin 1.

The IBM DB-9 standard is to put ground on pin 5, computer transmit on pin 3, and computer receive on pin 2.

8-pin Mini DIN
These connectors are round, about 1/4" in diameter, and have 8 tiny pins. They are common for serial ports on Macintosh and NeXT computers, as well as recent Sparc models. Do not try to deal with these connectors directly. Instead buy a DB-25 adapter cable and follow the DB-25 directions. These cables generally cost around five dollars and are sold as Macintosh modem cables. DO NOT buy a Macintosh *printer* cable—transmit and receive (pins 2 and 3) are swapped on these cables. If you only have a printer cable, you need to buy a null modem. These are available from just about any computer store.

3-pin Molex Connector
These are the connectors used for the serial ports on the processor board. See Section 4.1 [Cognachrome Vision System Board Diagram], page 16 for pinouts.

If you have a voltmeter, you can distinguish between the transmit and receive lines in the following manner:

- Set the meter to measure DC Volts.
- Touch the black probe to the ground pin and the red probe to the pin you expect to be transmit.
- If you measure between -5V and -10V, then that pin is indeed transmit.
- If the reading is very small (between 1V and -1V), then that either is not the transmit line, the port is not really connected, or the port is not powered up (this is common in the case of portable computers).
- Touch the red probe to the pin you expect to be receive. If the reading is very small, then it may be receive. If the reading is between -5V and -10V, then this is transmit (assuming the transmitter is idle). In this case, you need a null modem.

If you want to check if this transmit pin corresponds to a specific port, you can check the voltage when you send characters out that port. It should pulse briefly from the negative idle voltage to a positive voltage (between +5V and +10V) when data is transmitted. If you have an oscilloscope, you can see this happen. However, it happens too quickly for a voltmeter in DC mode to detect.

If you only have a voltmeter, you can detect serial data transmission somewhat by measuring AC Volts. Continuously transmit data over the serial port, either by holding down a key (and setting your auto-repeat rate fast enough), or by pasting a large buffer of text into the serial interaction program. You should see the AC voltage reading change from around 0V when data is not being transmitted, to something higher when data is being transmitted (in my test, I read up to about .4V when data was being transmitted).

B.3.6 Pioneer-Specific Serial Troubleshooting

If you are using a vision system installed in a Pioneer robot and you are having trouble getting serial interaction to the robot to work even though the loopback tests above work, follow these instructions. If you still cannot get the interaction to work after trying this procedure a couple of times, you may want to seek help. See Appendix D [Reaching Newton Research Labs], page 105.

- Make sure that Saphira is not running, and that no other software is trying to use the serial port you've connected the Pioneer to.
- Run ARC with the `-saphira` flag: `arc -saphira -port` *portname*.
- Reset the vision system by pressing the red *Boot Button* on the vision interaction panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7).

- See that you get a boot message in the main ARC window (see Section 6.1.1 [Startup Behavior], page 43).
  - Ideally you get something like this:
    ```
    Control-c to inhibit reset run...10...9...8...7...6...5...4...3...2...1...
    Initializing library vislib-26_0
    Cognachrome Vision System
    Software version 26.0
    Channel 0: Restored 1164 bytes
    Channel 1: Restored 1035 bytes
    Channel 2: Restored 6 bytes
    Restored a total of 2205 bytes
    Restored protocol string: "$%04xF:%04xR:%04xC#" from EEPROM
    Running program pioneer-26_0 as process 99
    vestaboot.986>
    ```
  - If you get something just like that, but it does not say `Running program pioneer-26_0` ..., then you need to download the Pioneer software (see Section 6.1.4 [Downloading Programs to the Vision System], page 47).
  - If you get something like this, then you need to download both the vision library and the Pioneer software (see Section 6.1.4 [Downloading Programs to the Vision System], page 47):
    ```
    Checking for reset run programs
    vestaboot.986>
    ```
  - If you get something, but it looks like garbage, the board is probably running at 38400 baud. (It is also likely that the Pioneer software is not loaded, as that software would set the baud rate to 9600 after the first reboot.)

    Restart ARC at 38400 baud—do not use a `-baud`, `-saphira`, or `-pioneer` flag:

    `arc -port` *portname*

    Now download the Pioneer software to the board (see Section 6.1.4 [Downloading Programs to the Vision System], page 47), and go back to the start of these instructions.
  - If you get nothing at all, you probably have a problem with getting power to the board or you have a problem with your serial cabling. See Appendix C [Pioneer Vision System Hardware Installation], page 101, for information about hooking up the power and serial cables.
- Repeat these steps until it works.

B.4 Problems and Possible Solutions

This section goes into problems you might run into that you can't figure out simply by looking at the LED's. Of course, if you have problems that aren't covered here and you need help, please contact us (see Appendix D [Reaching Newton Research Labs], page 105).

Symptom: The board acts as if it receives spurious commands during operation.

Your board stops tracking or executes unrequested commands at unexpected times.

If the jumpers on the processor board are misconfigured, the input to serial port B may be floating. Make sure that the Serial B jumper is installed (see Section B.2 [Processor Board Jumper Configuration], page 86).

If you are using a system with an IR receiver, you might be receiving noise through the IR receiver which the board interprets as commands. IR receivers will normally see a small amount of low-frequency noise during normal operation, but this small noise is extremely unlikely to ever be recognized as a valid Sony IR command. Still: *You might try disconnecting your IR sensor.*

In direct sunlight, Sharp IR sensors become extremely noisy, and may cause spurious commands on the order of one every few hours. Furthermore, it may be too noisy to interpret your IR remote control. Therefore: *Disconnect your IR sensor when operating in direct sunlight.*

B.5 Poor Color Recognition

Symptom: The debugging monitor shows that the color recognition is bad, but I can't seem to make it better by retraining.

Often this problem is due to a poor input video signal. There are lots of things that can cause a bad signal: some are camera-related, some are wiring related, and some are related to environmental conditions such as lighting. See Section 4.3.1 [Video Input], page 19, for some highly recommended reading to help you recognize and fix this family of problems.

Another possible problem is that your target is hard to distinguish from the background. If your target is too small, or has a surface that is "glossy" rather than "matte", you may have problems. Try to use targets with high color saturation—fluorescent or "neon" colors work well. See Section 5.1 [Choosing Tracking Targets], page 25, for recommendations.

Check whether signal filtering is on. When filtering is on, a minimum horizontal size must be reached in order for the system to recognize an object. If you need to track small objects, reduce filtering. If you see a lot of noise in the background, increase filtering. See Section 6.3.4.2 [Training and Configuring Channels Detail], page 52.

Double-check that your power supplies, especially those supplying analog voltages, are well-filtered. See Section 4.4 [Power Connections], page 22.

Finally, you may be having hardware trouble with our board. If you suspect hardware trouble, or can't otherwise track down the source of your problems, please contact us (see Appendix D [Reaching Newton Research Labs], page 105).

Appendix C Pioneer Vision System Hardware Installation

Hopefully, your Pioneer robot arrived with a vision system already installed and hooked up. If this is the case, you should not need these instructions and should just follow the Quick Start instructions (see Section 3.2 [Setting Up the Pioneer Vision Hardware], page 7). However, if you are installing the vision system into a Pioneer yourself, or if your system has been taken apart, these instructions should help you get it together and working.

Identify the boards

Your vision system consists of two boards (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). Notice now which board is the vision board, and which is the processor board. The vision board has two video connectors and many LED's at each end. The processor board has four large rectangular chips (the memory), and one large square chip (the processor). Stack the boards together with the vision board on top, and the processor board on the bottom. Verify the orientation is correct by seeing that the mounting holes on the four corners of each board line up. Press the boards together firmly.

Connect power to the vision system

Next, you'll want to connect power to your board. The power connector is the male 6-pin molex connector at the same end of the board as the video connectors (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). There should be a compatible female 6-pin molex connector coming from the Pioneer robot's power circuitry. Connect these two connectors together to provide power to the vision system.

While the robot is still open, turn on the main robot power. If you look from the right side of the robot so that the LCD display is at the top, this is the slide switch below and to the right of the charging plug. LED's on the top of the robot should light. Now turn on the vision system power. Again looking from the side with the LCD at the top, this is the switch above and to the left of the charging plug. Be sure that all the red LED's on the vision system light. If none of the vision system LED's turn on, check that both power switches are on.

WARNING: The power wiring is different for stand alone vision boards and for boards bundled with the Pioneer. The power wiring should have already been modified for use with the Pioneer. However, if the the red +5V Digital Power LED does not light, but the other two red LED's do, you may be using an unmodified vision system. Contact Newton Labs for information (see Appendix D [Reaching Newton Research Labs], page 105)

See Section B.1 [Status Lights and Beeper], page 85, for more general troubleshooting information.

Connect the serial wiring

The serial connectors for the vision system are 3-pin molex connectors at one end of the vision CPU board (the bottom board). *WARNING: The vision board also has a 3-pin molex connector that is used for power in the stand alone mode. DO NOT CONNECT SERIAL TO THAT CONNECTOR!* See Section 4.1 [Cognachrome Vision System Board Diagram], page 16.

Find the 6 conductor ribbon cable with two 3-pin molex connectors at one end. Half of the conductors in this cable go to the 9-pin serial connector on the vision interface panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7). This is the *host serial connector*. The other half of the ribbon cable's conductors go into the 6811 board in the top of the Pioneer robot. This is the *robot serial connector*.

Connect the host serial connector (as described above) to serial port A on the vision system (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). The other end of this cable will be connected to a host computer running ARC or Saphira. This will be accomplished in one of two ways:

- By directly connecting a serial cable from the computer to the 9-pin serial port on the vision interface panel.
- Via a radio modem, which will connect to the 10 pin header connector on the inside of the vision interface panel right below the 9-pin *Serial Port* connector (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7). Refer to the Pioneer robot documentation for more information about using a radio modem.

Connect the 3-pin molex end to port B (the one on the left) on the vision system (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). This allows the vision system to communicate with the robot controller. This is used either to pass information through between the robot and Saphira, or to allow the vision system to control the robot directly.

Connect the video wiring

Video input to the vision system comes from the camera, into the *Camera in* RCA connector on the vision interface panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7), through a cable, and into the *Video In* connector on the vision system (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16). To set this up, find the cable which goes from the inside of the vision interface panel *Camera In* connector. Trace it to the RCA plug at its other end. Now plug this into the vision system *Video In* connector (the white one on the left).

Video output comes from the vision system (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16), through a cable, and into the *Video Out* connector on the vision interface panel (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7). To set this up, find the cable which goes from the inside of the vision interface panel *Video Out* connector. Trace it to the RCA plug at its other end. With any luck, the cable leads to the only free RCA plug. Connect this into the vision system *Video Out* connector (the red one on the right). This will allow you to connect a video monitor to the *Video Out* RCA jack on the vision interface panel.

Connect the reset button

The reset button is the red button on the vision interface panel labeled *Boot Button* (see Section 3.1 [Pioneer Vision Interface Panel Diagram], page 7). The reset button will be attached to a cable ending in a 2-pin molex connector. Connect this to the corresponding 2-pin connector on the vision board. This is at the opposite end of the board from the video and power connectors (see Section 4.1 [Cognachrome Vision System Board Diagram], page 16; it is labeled *Reset Lever* on the diagram, but for the Pioneer configuration, this is a two pin connector).

Make sure it works

Work through the Pioneer Quick Start instructions (see Chapter 3 [Pioneer Quick Start Guide], page 7) to make sure it works. So long as the power and serial connections are correct, it should work to interact with the robot using the Saphira host program. In order for the vision tracking to work, the video input connections must be correct. These two systems are largely independent, so it should be easy to track down problems in each set of connections separately.

If after you have double checked the connections it still does not work, you may wish to contact ActivMedia or Newton Labs (see Appendix D [Reaching Newton Research Labs], page 105.

Screw the robot together

Congratulations! All the internal wiring is done at this point, so you can put the robot back together. You should now be able to interface to the vision system using the vision interface panel without needing to open the robot again

Appendix D Reaching Newton Research Labs

When you have questions about your vision board and this manual doesn't help, please send e-mail to `vision@newtonlabs.com`.

Also, we greatly appreciate any feedback you might give us about the Cognachrome Vision System and this documentation.

Address:

Newton Research Labs
Evergreen Park
16504 NE 37th St. #X2073
Redmond, WA 98052
Voice: (206) 869-4988
Fax: (206) 882-2043
E-mail: vision@newtonlabs.com
http://www.newtonlabs.com

Appendix E Index

A

B

C

D

H

I

J

K

L

M

Q

R

S

T

U

Table of Contents