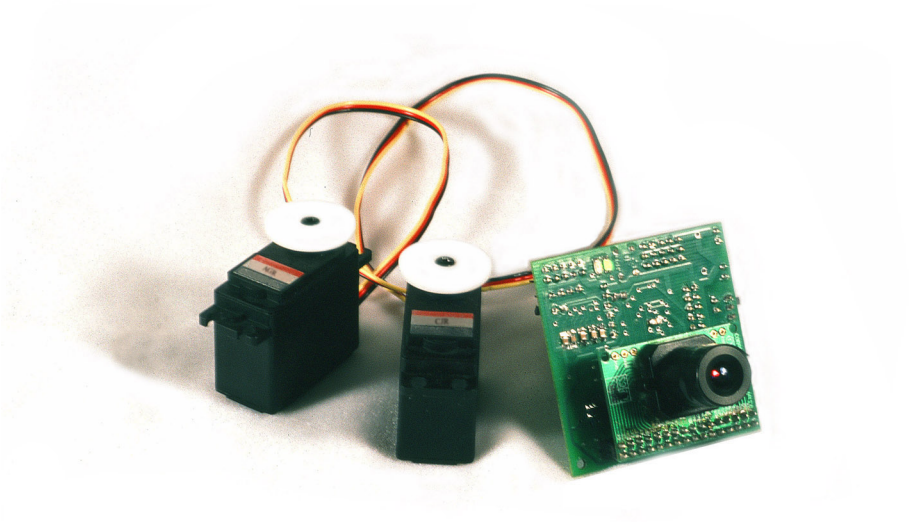
CMUcam2 Vision Sensor



User Guide

* Servos not included.

Contents

This Icon will link you to pages where more detailed general information can be found.

This icon will warn you about common mistakes.

This icon will point you to pages where commands used in the text are described.

This icon will suggest a generic tip from your friend the yellow dart.

This is the CMUcam2 Manual v1.06 for the CMUcam2 v1.0 firmware.
For more information go to http://www.cs.cmu.edu/~cmucam or contact us at cmucam@cs.cmu.edu
Copyright 2003 Anthony Rowe and Carnegie Mellon University. All Rights Reserved.
Edited by Charles Rosenberg and Illah Nourbakhsh

Introduction

The CMUcam2 consists of a SX52 microcontroller (http://www.ubicom.com/products/sx/sx.html) interfaced with an OV6620 or OV7620 Omnivision CMOS camera (http://www.ovt.com) on a chip that allows simple high level data to be extracted from the camera's streaming video. The board communicates via a RS-232 or a TTL serial port and has the following functionality:

- Track user defined color blobs at up to 50 Frames Per Second*
- Track motion using frame differencing at 26 Frames Per Second
- Find the centroid of any tracking data
- Gather mean color and variance data
- Gather a 28 bin histogram of each color channel
- Manipulate Horizontally Pixel Differenced Images
- Transfer a real-time binary bitmap of the tracked pixels in an image
- Arbitrary image windowing
- Adjust the camera's image properties
- Dump a raw image (single or multiple channels)
- Up to 160 x 255 Resolution**
- Supports Multiple Baudrates: 115,200 57,600 38,400 19,200 9,600 4,800 2,400 1,200
- Control 5 servo outputs
- Slave parallel image processing mode off of a single camera bus
- Automatically use servos to do two axis color tracking
- B/W Analog video output (PAL or NTSC)**
- Flexible output packet customization
- Multiple pass image processing on a buffered image
- Works with the OV7620 or OV6620 module

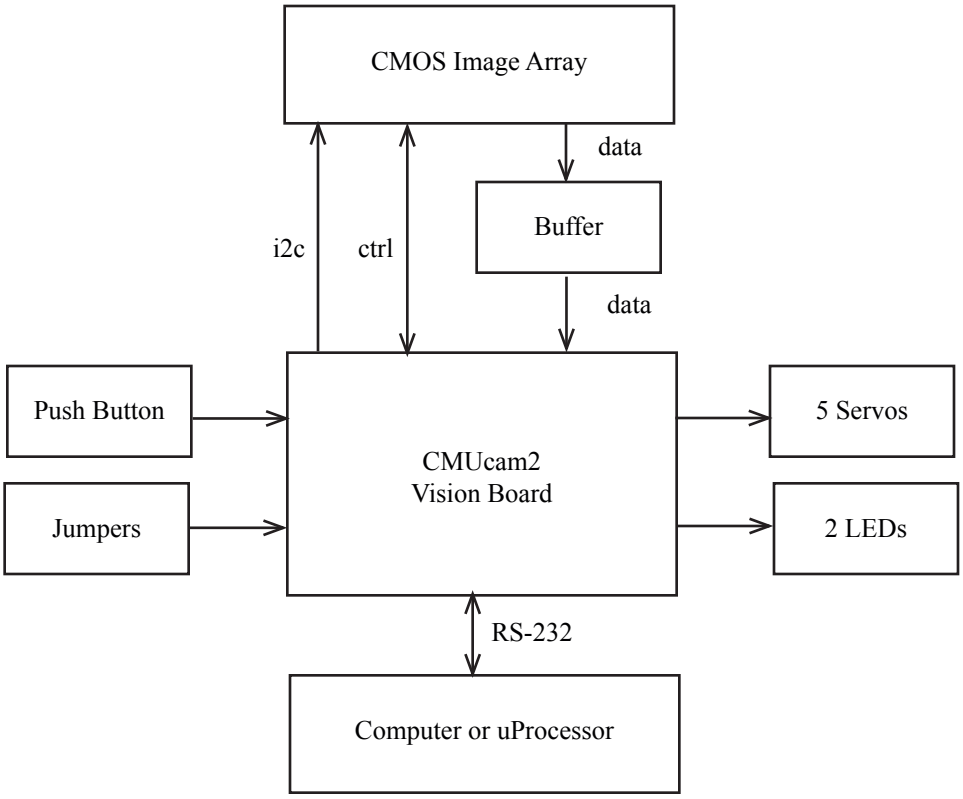
*Frame Rate Depends on Window Size
**Camera Properties Depend on Camera Module

Typical Configurations and Uses

Typical Uses

One of the primary uses of the CMUcam2 is to track or monitor color. The best performance can be achieved when there are highly contrasting and intense colors. For instance, it can easily track a red ball on a white background, but it would be hard to differentiate between different shades of brown in changing light. Tracking colorful objects can be used to localize landmarks, follow lines, or chase a moving beacon. Using color statistics, it is possible to monitor a scene, detect a specific color or do primitive motion detection. If the camera detects a drastic color change, then chances are something in the scene changed. Using "line mode," the CMUcam2 can act as an easy way to get low resolution binary images of colorful objects. This can be used to do more sophisticated line following that includes branch detection, or even simple shape recognition. These more advanced operations would require custom algorithms that would post process the binary images sent from the CMUcam2.

See line mode on page 35.



Typical Configuration

The most common configuration for the CMUcam2 is to have it communicate to a master processor via a standard RS232 serial port. This "master processor" could be a computer, PIC, Basic Stamp, Handy Board, Brainstem or similar microcontroller setup. The CMUcam2 is small enough to add simple vision to embedded systems that cannot afford the size or power of a standard computer based vision system. Its communication protocol is designed to accommodate even the slowest of processors. If your device does not have a fully level shifted serial port, you can also communicate to the CMUcam2 over the TTL serial port. This is the same as a normal serial port except that the data is transmitted using non-inverted 0 to 5 volt logic. The CMUcam2 supports various baud rates to accommodate slower processors. For even slower processors, the camera can operate in "poll mode". In this mode, the host processor can ask the CMUcam2 for just a single packet of data. This gives slower processors the ability to more easily stay synchronized with the data. It is also possible to add a delay between individual serial data characters using the "delay mode" command. Due to the communication delays, both poll mode and delay mode will lower the total frame rate that can be processed. Frame resolutions are not affected by delay mode or baud rate as they were in the original CMUcam.

See poll mode on page 46.

See delay mode on page 33.

Operational Explanation

How does an image get converted into a series of pixels?

The CMOS image sensor is the heart of what actually gathers the information. It is a silicon chip that contains a grid of boxes, each of which are sensitive to different colors of light. After light passes through the lens, it stimulates these boxes, generating a different voltage proportional to the amount of light. This voltage gets converted into a single numerical value for each channel. In the case of the CMUcam2, this value is in the range of 16 to 240. There is a red channel, a blue channel and two green channels, each of which are only sensitive to that particular color of light. The extra green channel helps fill in the grid so that each pixel can be evenly spaced across the sensor. The extra green information also more closely approximates the human eye which is more sensitive to the color green. For the purpose of simplification, the CMUcam2 ignores the second green value.

Camera Sensor Output Pixel Mapping

It is sometimes useful to understand more percisely how the data from the camera sensor is translated into pixels. Here we explain it for the OV6620 sensor, but the same basic layout applies to the OV7620 sensor.

The sensor has 356 columns and 292 rows of light sensitive cells arranged on a grid. Each location can detect a single color: red, green or blue. Here is the sensor layout of the first four rows:

Row 1: B(1,1) G(1,2) B(1,3) G(1,4) B(1,5) G(1,6) ...B(1,355) G(1,356)
Row 2: G(2,1) R(2,2) G(2,3) R(2,4) G(2,5) R(2,6) ...G(2,355) R(2,356)
Row 3: B(3,1) G(3,2) B(3,3) G(3,4) B(3,5) G(3,6) ...B(3,355) G(3,356)
Row 4: G(4,1) R(4,2) G(4,3) R(4,4) G(4,5) R(4,6) ...G(4,355) R(4,356)

The camera module takes the data from two sensor rows at a time to generate each line output from the camera module:

Row 1: B(1,1) G(2,1) R(2,2) G(1,2) B(1,3) G(2,3) R(2,4) G(1,4) ...
Row 2: B(3,1) G(2,1) R(2,2) G(3,2) B(3,3) G(2,3) R(2,4) G(3,4) ...

The CMUcam2 takes this data and outputs following pixel data:

Row 1: [R(2,2):G(1,2):B(1,1)] [R(2,4):G(1,4):B(1,3)] ...
Row 2: [R(2,2):G(3,2):B(3,1)] [R(2,4):G(3,4):B(3,3)] ...

What is tracking a color and how does the CMUcam2 do it?

Color tracking is the ability to take an image, isolate a particular color and extract information about the location of a region of that image that contains just that color. As an example, assume that you are given a photograph that contains a red ball sitting on a dirt road. If someone were to ask you to draw a box around anything that was the color red in the image, you would quite easily draw a rectangle around the ball. This is the basic idea behind color tracking. You did not need to know that the object was a ball. You only needed to have a concept of the color red in order to isolate the object in the picture. In this section we will briefly address how the CMUcam2 actually uses the information in a camera image to perform color tracking.

(Photograph Courtesy of Jim Reed)

In order to specify color, you need to define a minimum and maximum allowable value for each of those three color channels. Every unique color is represented by a red, green, and blue value that indicates how much of each channel is mixed into that final color. The tricky part about specifying a color is that you need to define a range of allowable values for all three color channels. Since light is not perfectly uniform and the color of an object is not perfectly uniform, you need to accommodate for these variations. However, you don't want to relax these bounds too much, or many unwanted colors will be accepted. Since, in the case of the CMUcam2, each color channel is converted into a number between 16 and 240, you can bound each channel with two numbers, an upper and lower limit. If you have two limits for each of the three channels, this means that six values can be used to constrain the entire color space that you wish to track. If you imagine the colors being represented by a cube where each side is a different color channel (red, green and blue) then the six values used to select your color would draw a three dimensional box inside that cube that defines your desired set of colors.

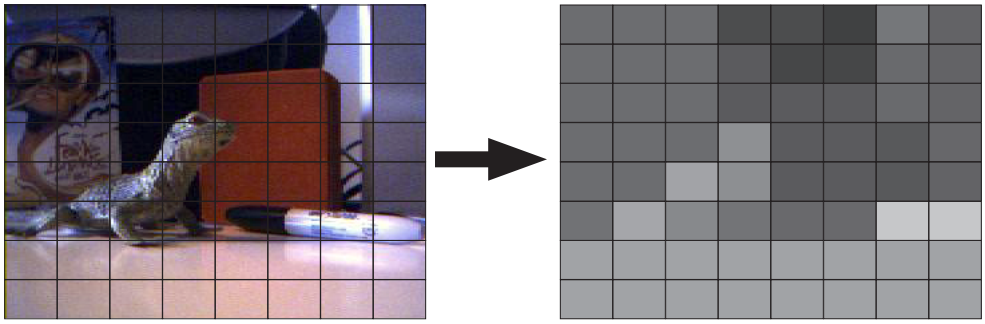
Once you have a bound for the color you wish to track, the CMUcam2 takes these bounds and processes the image. There are many ways to track colors in an image that can be quite complex. The CMUcam2 uses a simple one pass algorithm that processes each new image frame from the camera independently. It starts at the top left of the image and sequentially examines every pixel row by row. If the pixel it is inspecting falls inside the range of colors that the user specified, it marks that pixel as being tracked. It also examines the position of the current tracked pixel to see if it is the top most, bottom most, left most or right most position of all the tracked pixel found thus far in the image. If it finds that the pixel is outside of the current bounding box of the tracked region, it grows the bounding box to contain this new pixel. Because the location of even a single tracked pixel can change the bounding box, the bounding box can sometimes fluctuate quite a bit from frame to frame. Noise filtering (see next paragraph) can be used to reduce some of that fluctuation. The only other major piece of information that is stored is a sum of the horizontal and vertical coordinates of the tracked pixels. At the end the image you can take the horizontal sum and the vertical sum of the tracked pixels and divide each by the total number of tracked pixels, you get a value that shows where the middle of the tracked object is located. Because each tracked pixel only contributes a small part to the final horizontal and vertical sums the middle (often called the centroid) of the tracked pixels is typically a much more stable measurement than the bounding box. Once all of the pixels in the image have been checked, the total number of tracked pixels can also be used in conjunction with the area of the bounding box to calculate the confidence of the tracked object.

Noise filtering allows us to make the color tracking ranges larger so we can accommodate larger variations in the image pixel values without causing other random variations in the image to be tracked. The idea behind noise filtering is that we only want to consider a pixel to be of the tracked color if it is part of a group of pixels that are within the color tracking bounds. Again in the CMUcam2 we implement this in a way that only requires a single pass over the image. While processing the pixels in an image the CMUcam2 maintains a counter which keeps of track of how many sequential pixels in the current row, before the current pixel were within the tracked color bounds. If that value is above the noise filter value then the current pixel is marked as a tracked pixel.

How does the CMUcam2 do Frame Differencing?

Frame differencing is a method of identifying changes in a series of images. Given multiple images at different times from the same or similar view points, it is possible to compare them in order to isolate objects that may have moved. Using the CMUcam2's frame differencing functionality is a good way to detect and track such motion in a scene. Instead of storing an entire image, the CMUcam2 stores an abstraction of the image. Using a similar process to color tracking, the CMUcam2 will generate or compare the image on a line by line basis as it receives the data.

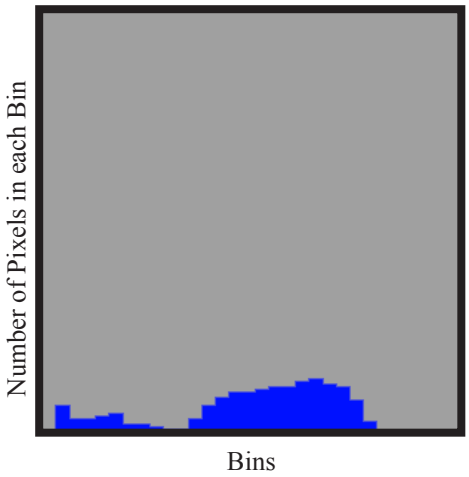
The CMUcam2 internally represents a reference image as an array of 8 by 8 bytes. Each element of this array stores the average of a corresponding region on the main camera image. The default setting uses the green or intensity channel, but this can be changed for situations where one channel clearly shows more variation than the others. When a new image is read in, it is also converted into an array of 8x8 bytes. To look for a change, each block in the 8x8 grid is subtracted from the corresponding reference image block. If there is more than a specified threshold, a change is flagged. The rest of the data, such as middle mass, is calculated in an almost identical manner to the way it is in color tracking.



What is a histogram and what is it good for?

A histogram is a type of chart that displays the frequency and distribution of data. In the case of the CMUcam2, the histograms show the frequency and distribution of color values found in an image. Each bar represents a range of color values for a specific channel. The CMUcam2 can divide the possible color values from 16 to 240 into up to 28 different bins. Each bin contains the number of pixels found in the image that fall within those color bounds. So a large value in one particular bin, means that many of those colors where found in the image. Each histogram only represents one select channel of color. Using buffer mode it is possible to quickly grab three histograms, one for each channel.

Histograms are a way of abstracting the contents of an image. They have many uses such as primitive object recognition, thresholding or color balancing. They are particularly useful for distinguishing between different textures. Try pointing the CMUcam2 with auto-gain turned off at two different textured surfaces and notice the difference in their color distributions. This effect could be used to distinguish floor surfaces or detect obstacles. When used in conjunction with pixel differencing a histogram can tell you about the strength of the edges visible to the camera.

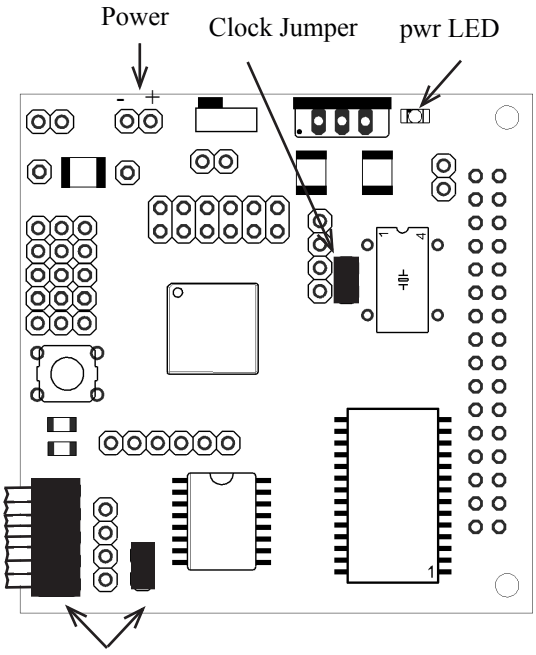


Getting Started

Setting Up the Hardware

In order to initially test your CMUcam2, you will need a serial cable, a power adapter and a computer. The CMUcam2 can use a power supply which produces anywhere from 6 to 15 volts of DC power capable of supplying at least 200mA of current. This can be provided by either an AC adapter (possibly included) or a battery supply. These should be available at any local electronics store. The serial cable should have been provided with your CMUcam2. Make sure that you have the CMOS sensor board connected to the CMUcam2 board so that it is in the same orientation as the picture shows on the cover of this manual.

See page 59 for startup troubleshooting.



First, connect the power. Make sure that the positive side of your power plug is facing away from the main components on the board. If the camera came with an AC adapter, make sure that the connector locks into the socket correctly.

Now that the camera has power, connect the serial link between the camera and your computer. This link is required initially so that you can test and focus your camera. The serial cable should be connected so that the ribbon part of the cable faces away from the board. You must also connect the serial pass through jumper.

Check to make sure that the clock jumper is connected. This allows the clock to actively drive the processor.

Once everything is wired up, try turning the board on. The power LED should illuminate green and only one LED should remain on. Both LEDs turn on upon startup, and one turns off after the camera has been sucessfully configured.

Make Sure Clock and Serial Jumper are in place.

Testing the Firmware

Once you have set the board up and downloaded the firmware, a good way to test the system is to connect it to the serial port of a computer.

Step 1: If one does not already exist, build a serial and/or power cable

Step 2: Plug both of them in.

Step 3: Open the terminal emulator of your choice.

See page 62, for more detailed terminal software information.

Step 4: Inside the terminal emulator set the communication protocol to 115,200 Baud, 8 Data bits, 1 Stop bit, no parity, local echo on, no flow control and if possible turn on "add line feed" (add \n to a received \r). These setting should usually appear under "serial port" or some other similar menu option.

Step 5: Turn on the CMUcam2 board; the Power LED should light up and only one of the two status LEDs should remain on.

Step 6: You should see the following on your terminal emulator:

```
CMUcam2 v1.0 c6
:
```

If you have seen this, the board was able to successfully configure the camera and start the firmware.

Step 7: Type gv followed by the enter key. You should see the following:

```
:gv
ACK
CMUcam2 v1.0 c6
:
```

See get version on page 37.

This shows the current version of the firmware. If this is successful, your computer's serial port is also configured correctly and both transmit and receive are working.

Focusing with the CMUcam2 Graphical User Interface (GUI)

When you first run your CMUcam2, the lens will most likely not be in focus. In order to focus the camera you need to look at some dumped images. The easiest way to do this is using a graphical user interface that can display the CMUcam2 frame dump packets. One option is to use the CMUcam2GUI, a Java program that can be found on the CMUcam2 website.

Step 1: Testing if you have java installed

The CMUcam2GUI needs java version 1.4.0 or higher.

The first step is to determine if your computer already has java installed. The easiest way to do this is go to the "start menu" in windows and select "run". Inside the run dialog, type "command" to get a dos prompt. (In unix or later versions of the Mac OS, open up a shell.) Now try typing "Java -version" into your command line. If a message that says "Java version "1.x.xx" appears then java is installed. If instead you get "command not found" or some similar message, then you need to go to java.sun.com and download a copy of Java (J2SE, JDK, JRE are all valid things to install). Sun should have platform specific instructions on how to install java. Also be sure that your version of Java is 1.4.0 or newer. If it is not, then you will need to download a new copy of Java.

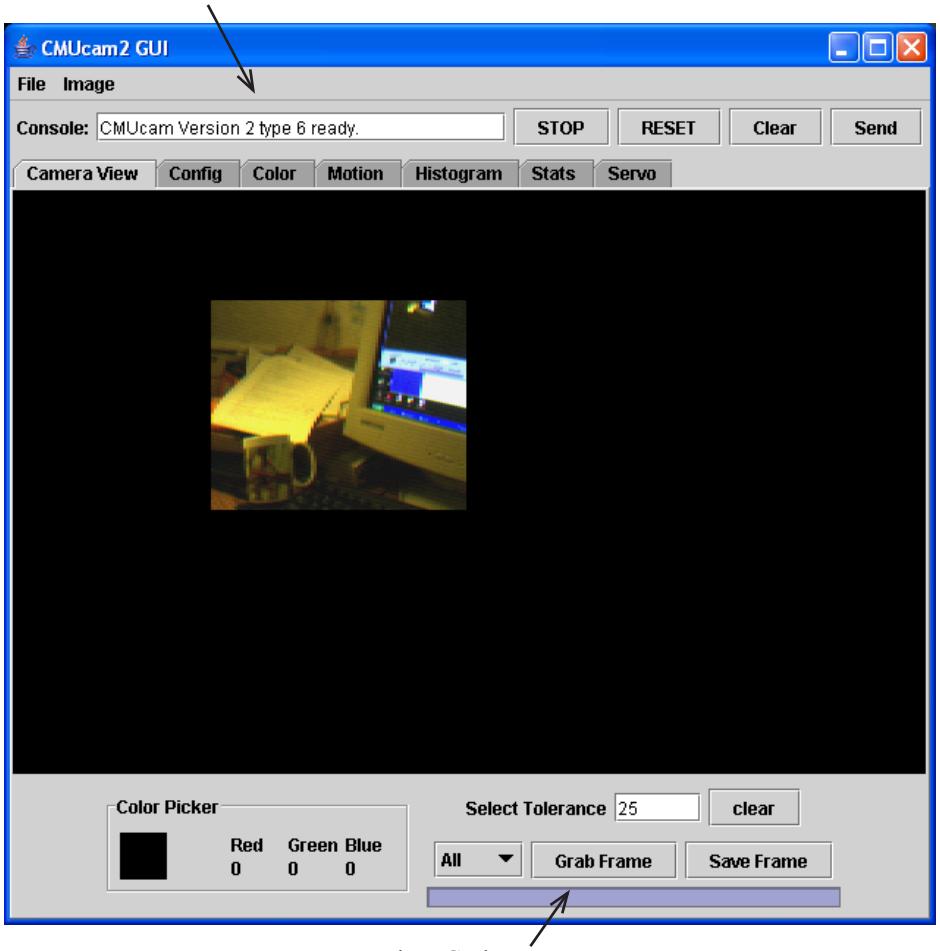
Step 2: Running the CMUcam2GUI

Once you have Java installed, download a copy of the latest CMUcam2GUI Java program. Unzip the CMUcam2GUI.zip file. Open up the stand_alone folder. In Windows double click on the CMUcam2GUI jar file. In unix, navigate to the CMUcam2GUI directory and type "java -jar CMUcam2GUI" to execute the GUI.

Step 3: Grabbing a Frame

You should now see a dialog box that asks you to select the correct serial COM port. In windows, type in the number of the COM port that the CMUcam is connected to and press the "Ok" button. In unix, make sure that the path to your com port is correct and then press "Ok". The CMUcamGUI should now open and display the message "CMUcam version 2 type X ready." in the "Console" box. That means that the CMUcam2GUI found and was able to communicate with the camera. Once this works, select "Send Frame". After a few seconds you should see an image appear in the window.

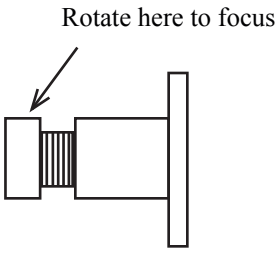
This means that the camera was found.



Push to Grab a Frame from the Camera

Step 4: Focusing

Once you have the ability to grab frames from the camera, you should be able to rotate the front part of the CMUcam lens and see the image change. Try to get the picture to be as sharp as possible by dumping frames and changing the position of the lens a small amount each time. Usually the camera is in focus when the lens is a few rotations away from the base. (Once you have focused the lens you may find it useful to use some electrical tape to keep it in place)



CMUcam Lens Mount

Step 5: Other things to try once the camera is focused

Now take a quick look at the Config tab. When you change Color Space, White Balance, etc. (except for Noise Filter), it will automatically get sent and configured to the CMUcam.

Now go to the Color tab. This has the TrackWindow button. Place a uniform, highly color-saturated object in front of the camera and click this button to track. To stop it use the "STOP" button top right. Try it with line mode by setting Config line mode on.

Go to Motion tab. Position the camera so it is looking at something static (non-moving) and hit Load Frame. Then immediately hit Frame Diff and continuous frame differencing to the loaded frame begins. Move a small object like a pencil across the camera FOV(field of view) to test. When done hit Stop.

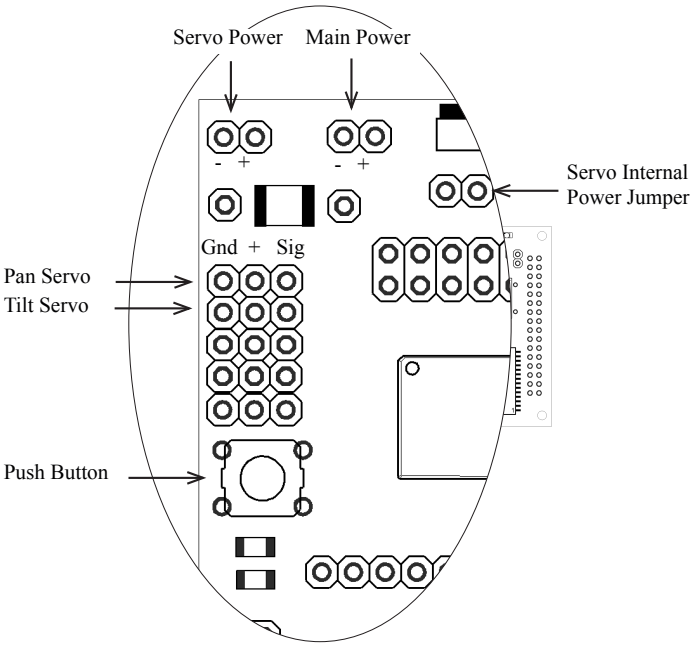
Now go to Histogram tab. In here you can do 1D histograms of each color channel individually. Left to right, the histogram shows amount of '0' at the left extreme (no intensity in that color) and '255' at the right extreme (high intensity in that color). It's continuous once you hit Get Histogram. Try something black, homogenously colored, something with varied color. Again, use STOP to finish.

On the Stats page, once you hit GetMean, there is a very nice continuous update of the mean color seen across the camera's window.

Demo Mode

Demo mode causes the camera to call track window and then drive two standard hobby servos towards the object being tracked. This can be initiated autonomously at startup. First you need to plug a pan and/or tilt servo into servo ports 0 and 1. Servo port 0 is for the pan, while 1 is for the tilt. Next, make sure that the servos are being powered by either the internal servo power jumper or by an external power source. While holding down the push button, turn the camera on. The tracking LED should begin rapidly blinking. Immediately release the push button and wait for the LED to stop blinking. Next, point the camera at a colored object and press the push button again. This should grab the color of the object and begin automatically servoing towards it. If the servos appear to be driving in the reverse direction, add the appropriate servo direction jumper. During the period when the LED is blinking, the camera is adjusting to the light conditions in the room. Try not to hold the object in front of the camera while this is occurring. Experiment with different colors and lighting. You will notice that some work much better than others.

See page 21, for pan and tilt servo reverse jumpers.

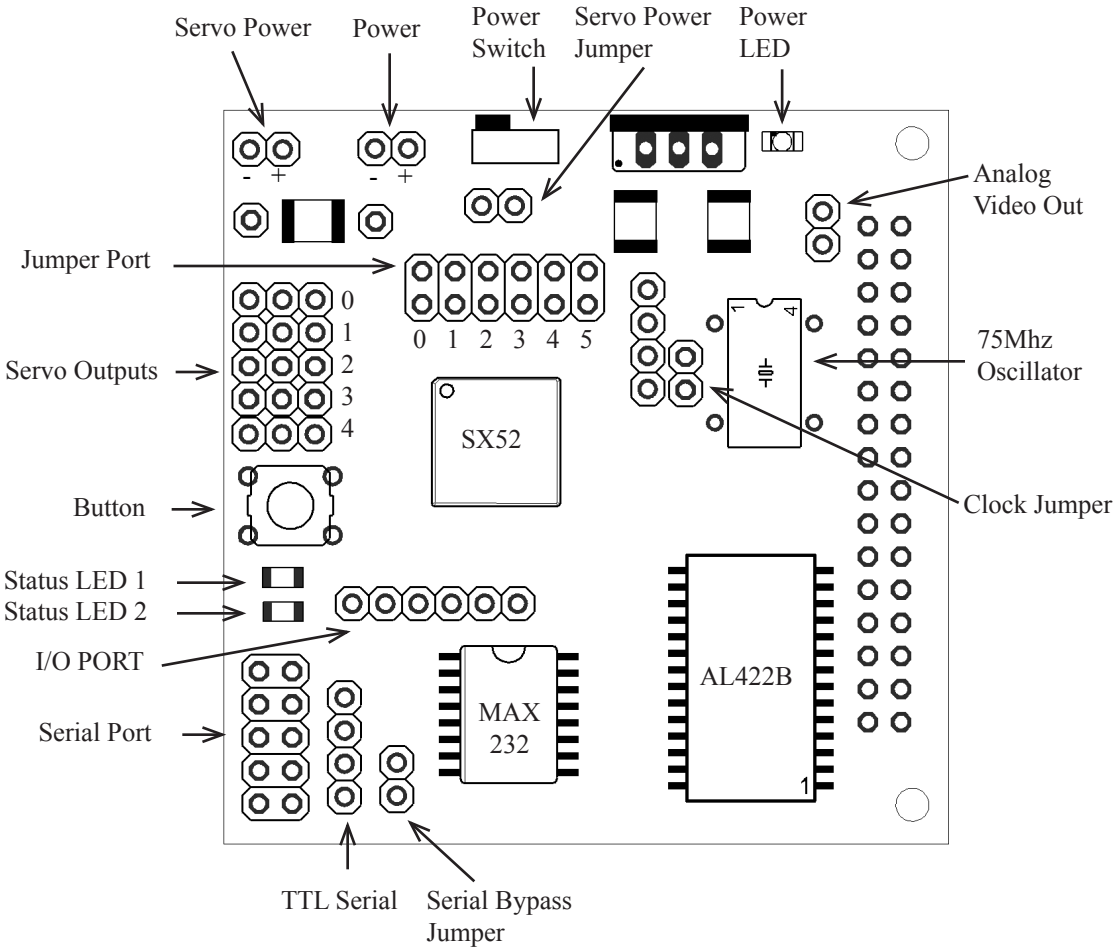


The following steps are performed during power up in demo mode:

1. RS is sent to the camera
2. Pause 5 seconds while blinking the LED to allow the camera to stabilize
3. The Camera register string “CR 18 32 19 32” is sent.
4. Auto Servo Mode is enabled.
5. TW is called.

See RS on page 49.
See CR on page 31.
See SM on page 51.
See TW on page 54.

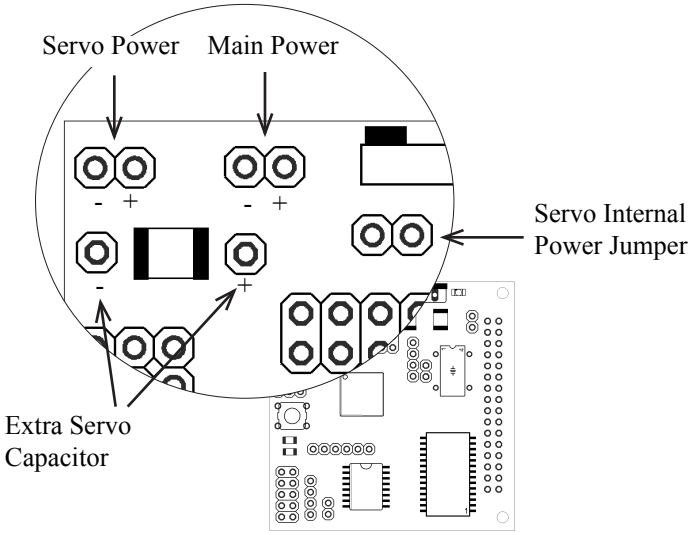
Board Layout



Ports

Power

The input power to the board goes through a 5 volt regulator. It is ideal to supply the board with between 6 and 15 volts of DC power that is capable of supplying at least 200 milliamperes of current.



Do not connect external servo power while the servo jumper is in place

If the servos are jittering or the camera does not properly power up, try soldering a 100uF external capacitor to the extra servo cap pads.

The servos can either be powered by internal power, or by the external servo power connector. To run them off of internal power, connect a jumper across the “servo internal power jumper”. To run them off of external power, leave the jumper open, and connect another 5volt supply to the servo power connector. Do not connect an external servo supply while the servo power jumper is in place. If the servos are drawing too much power or seem to be noisy, try soldering a large valued capacitor across the “Extra Servo Capacitor” connections. The external servo power is not switched by the main power switch.

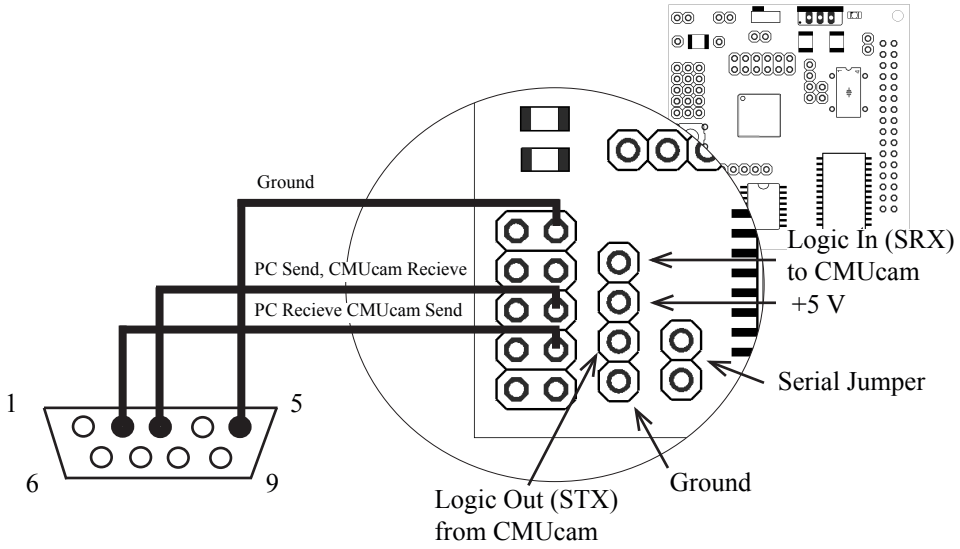
Serial Port

The CMUcam2 has a standard level shifted serial port to talk to a computer as well as a TTL serial port for talking to a microcontroller.

If the standard serial port does not work, try plugging in the serial connector the opposite way.

The level shifted serial port only uses 3 of the 10 pins. It is in a 2x5 pin configuration that fits a standard 9 pin ribbon cable clip-on serial sockets and 10 pin female clip on serial headers that can both attach to a 10 wire ribbon cable. If this initially does not work, try flipping the direction that the ribbon cable connects to the CMUcam2 board. Make sure the serial jumper is in place when you use this mode.

The TTL connector can be used to talk to a micrcontroller without the use of a level shifting chip. It operates between 0 and 5 volts. Remove the Serial Jumper when you use this mode.

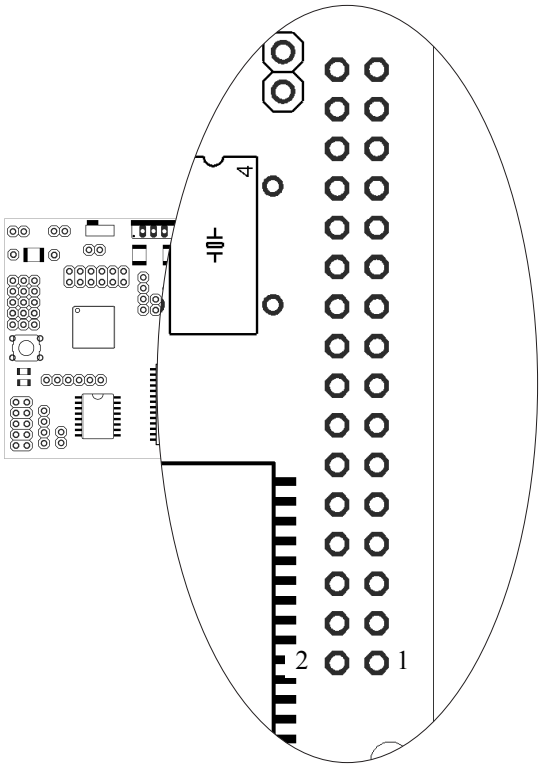


The Trapezoidal serial connector shown is what the serial connector on your computer should look like if drawn in an annoying line art drawing program.

Camera Bus

This bus interfaces with the CMOS camera chip. The CMOS camera board is mounted parallel to the processing part of the board and connects starting at pin 1. The female camera header should be soldered on the back of the board.

See page 25 for more information on the CMOS camera chips.



1-8	Y0-Y7	Digital Output Y Bus
9	PWDN	Power Down Mode
10	RST	Reset
11	SDA	I2C Serial Data
12	FODD	Odd Field Flag
13	SCL	I2C Serial Clock
14	HREF	Horizontal Ref
15	AGND	Analog Ground
16	VSYNC	Vertical Sync
17	AGND	Analog Ground
18	PCLK	Pixel Clock
19	EXCLK	External Clock
20	VCC	+5 VDC
21	AGND	Analog Ground
22	VCC	+5 VDC
23-30	UV0-UV7	Digital Output UV BUS
31	GND	Common Ground
32	VTO	Video Out (75Ohm)

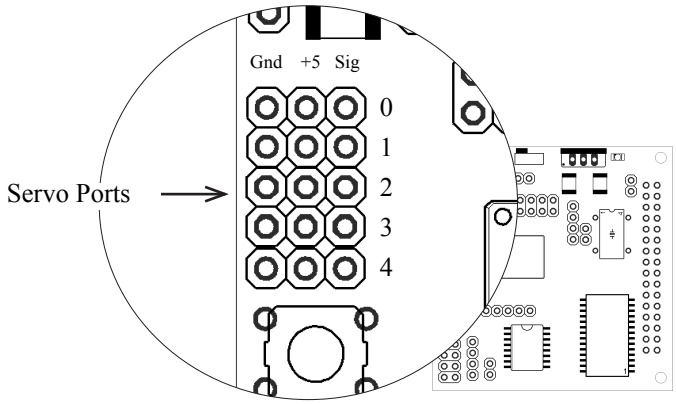
See the picture on the cover of the manual to make sure that you have the CMOS sensor connected correctly.

Servo Port

The CMUcam2 has the ability to control 5 servos. This can be useful if you do not wish to use a separate servo controller. The servo port can also be used as a general purpose digital outputs.

See SV servo command on page 53.

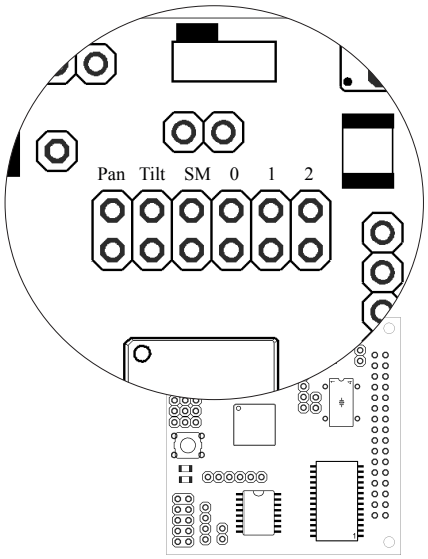
See SO servo command on page 51.



See page 17 for more information on servo power.

Configuration Jumpers

The jumpers can be used to set the camera's baudrates or configure various modes of operation.



Jumpers 0 1 and 2 set the camera into the following baudrates:

Baud Rate	Pin 0 1 2
115,200 Baud	_ _ _
57,600 Baud	_ _ X
38,400 Baud	_ X _
19,200 Baud	_ X X
9,600 Baud	X _ _
4,800 Baud	X _ X
2,400 Baud	X X _
1,200 Baud	X X X

X - jumper closed
_ - jumper open

Pan and Tilt Reverse Jumpers

During Auto Servo Mode, or demo mode it may be necessary to reverse the direction of the pan or tilt servo. Connecting the pan and/or tilt jumper will cause auto servo mode to send the opposite commands to each servo. Note, this only works for auto servo mode, and not for normal servo operations.

Slave Mode Jumper

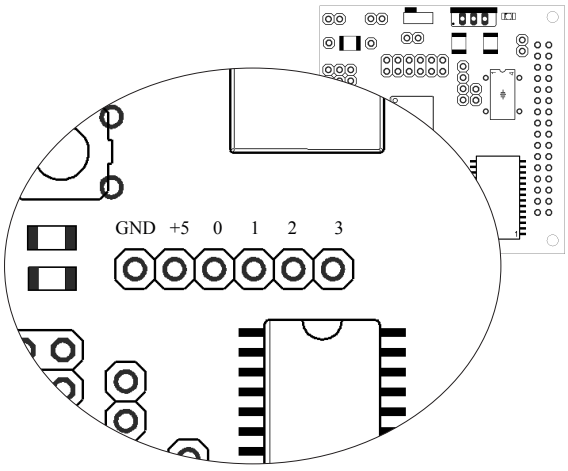
See CT command on page 32.

The CMUcam2 supports a mode of operation that allows multiple boards to process data from the same camera. If a PC104 style pass-through header is used instead of the standard double row female header, it is possible to rack multiple boards along the same camera bus. Upon startup, if the "SM" jumper is set, the camera becomes a slave. Slave mode stops the camera board from being able to configure or interfere with the CMOS camera's settings. Instead it processes the format setup by the master vision board. When linking the buses together you must only have one master; all other boards should be setup to be in slave mode. In this current version of the system there is no message passing between boards other than the image data from the camera bus. This means you have to communicate to each slave board via a separate serial link. This communication to the board should be identical to using a single CMUcam2. For example, you could have the master board tracking some color while the slave board could be told to get mean color data. Each board runs independently of one another and only the master can control camera registers.

Axuliary I/O

See GI command on page 35.
See SD command on page 49.

The CMUcam has 4 auxiliary Input Output ports that can be used for reading data from external devices. Note, that pin 3 is used for the sleep deeply command.



Analog Video Port

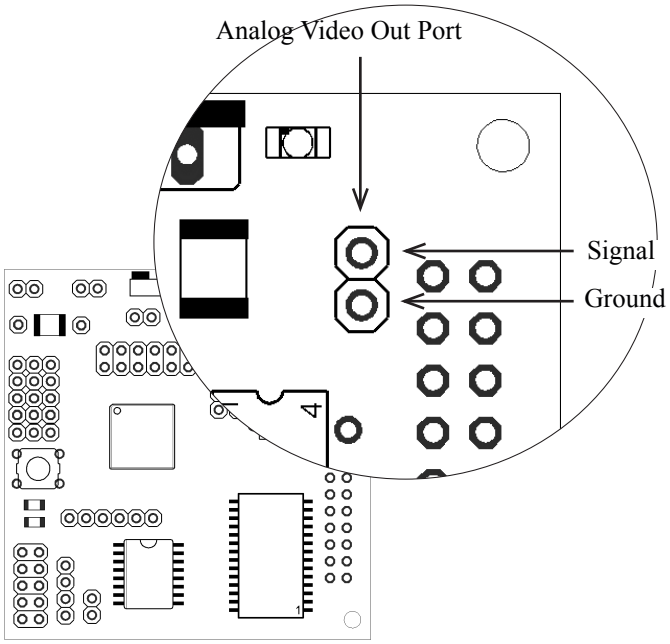
Using the OV6620 camera module, you will be able to get a PAL video signal from the analog port of the CMUcam2. This would sync up with any PAL monitor, but will not work with a standard NTSC monitor.

The OV7620 camera module will output a standard black and white NTSC video signal.

Make Sure Camera is operating at full frame rate and in YCrCb mode.

To use this output, it is necessary to keep the camera at its maximium frame rate (the default) and switch it into YCrCb mode in order to see the image on a monitor.

See CR command on page 31 for info on how to switch to YCrCb mode.



Notes on Better Tracking

Auto-gain and White Balance

Auto-gain is an internal control that adjusts the brightness level of the image to best suit the environment. It attempts to normalize the lights and darks in the image so that they approximate the overall brightness of a hand adjusted image. This process iterates over many frames as the camera automatically adjusts its brightness levels. If for example a light is turned on and the environment gets brighter, the camera will try and adjust the brightness to dim the overall image.

The camera module requires Auto-gain to be enabled to utilize white balance.

White balance on the other hand attempts to correct the camera's color gains. The ambient light in your image may not be pure white. In this case, the camera will see colors differently. The camera begins with an initial guess of how much gain to give each color channel. If active, white balance will adjust these gains on a frame-by-frame basis so that the average color in the image approaches a gray color. Empirically, this "gray world" method has been found to work relatively well. The problem with gray world white balance is that if a solid color fills the camera's view, the white balance will slowly set the gains so that the color appears to be gray and not its true color. Then when the solid color is removed, the image will have undesirable color gains until it re-establishes its gray average.

When tracking colors, like in demo mode, you may wish to allow auto-gain and white balance to run for a short period and then shut them off. While on for a period of about 5 seconds, the camera can set its brightness gain and color gains to what it sees as fit. Then turning them off will stop the camera from unnecessarily changing its settings due to an object being held close to the lens or shadows etc. If auto-gain and white balance where not disabled and the camera changed its settings for the RGB values, then the new measured values may fall outside the originally selected color tracking thresholds.

YCrCb Color Space

YCrCb is a different color space definition from the more commonly known RGB space. In YCrCb the pixel illumination data is stored in the Y channel. Because of this property, in YCrCb mode the camera may be more resistant to changes in illumination. Because it is a different color space, images in YCrCb do not look like standard RGB images when directly mapped by a frame dump program. The RGB channels map to CrYCb. So in YCrCb mode, the value returned as the red parameter is actually Cr, the green parameter is Y and the blue parameter is Cb. So if you wish to track a red object, you need to look at a dumped frame to see what that object's colors map to in YCrCb. It should then be possible to find the Cr and Cb bounds while giving a very relaxed Y bound showing that illumination is not very important. Below are the transformations used by the camera to convert RGB into YCrCb:

Notice that the RGB channels map to give you CrYCb, not YCrCb.

RGB -> CrYCb
Y=0.59G + 0.31R + 0.11B
Cr=0.713x (R-Y)
Cb=0.564x (B-Y)

When using YCrCb, make sure you take into account that in terms of all CMUcam I/O, Red maps to Cr, Green to Y and Blue to Cb.

About the CMOS Camera Modules

From power up, the camera can take up to 5 seconds to automatically adjust to the lighting conditions. Drastic changes in the environment, such as lights being turned on and off, can induce a similar readjustment time. When using the camera outside, due to the sun's powerful IR emissions, even on relatively cloudy days, it will probably be necessary to use either an IR filter or a neutral density camera filter to decrease the ambient light level. The field of view depends on the lens attached to the camera. It is possible to special order the camera with wider or narrower lenses. Individual lenses can be purchased separately.

The functions provided by the camera board are meant to give the user a toolbox of color vision functions. Actual applications may greatly vary and are left up to the imagination of the user. The ability to change the viewable window, grab color / light statistics and track colors can be interwoven by the host processor to create higher level functionality.

One notable property of the CMOS sensor array is that it returns values between 16 and 240 for each pixel. This effect is noticeable when the camera is tracking colors, getting mean color data or dumping a frame. This limited range on the data does not depend on the mode of the camera and still applies in YCrCb mode.

Serial Commands

The serial communication parameters are as follows:

- 1,200 to 115,200 Baud
- 8 Data bits
- 1 Stop bit
- No Parity
- No Flow Control (Not Xon/Xoff or Hardware)

All commands are sent using visible ASCII characters (123 is 3 bytes “123”). Upon a successful transmission of a command, the ACK string should be returned by the system. If there was a problem in the syntax of the transmission, or if a detectable transfer error occurred, a NCK string is returned. After either an ACK or a NCK, a \r is returned. When a prompt (’\r’ followed by a ‘:’) is returned, it means that the camera is waiting for another command in the idle state. White spaces do matter and are used to separate argument parameters. The \r (ASCII 13 carriage return) is used to end each line and activate each-command. If visible character transmission causes too much overhead, it is possible to use varying degrees of raw data transfer.

See Raw Mode on page 48 for information on configuring ascii vs raw text packets.

Functionally Grouped Command Listing

Buffer Commands

Camera Module Commands

Data Rate Commands

Servo Commands

Image Windowing Commands

Auxiliary I/O Commands

Color Tracking Commands

Histogram Commands

Frame Differencing Commands

Color Statistics Commands

System Level Commands

Alphabetical Command Listing

\r

This command is used to set the camera board into an idle state. Like all other commands, you should receive the acknowledgment string “ACK” or the not acknowledge string “NCK” on failure. After acknowledging the idle command the camera board waits for further commands, which is shown by the ‘:’ prompt. While in this idle state a \r by itself will return an “ACK” followed by \r and : character prompt. This is how you stop the camera while in streaming mode.

Example of how to check if the camera is alive while in the idle state:

```
:
ACK
:
```

BM active \r

This command sets the mode of the CMUcam’s frame buffer. A value of 0 (default) means that new frames are constantly being pushed into the frame buffer. A value of 1, means that only a single frame remains in the frame buffer. This allows multiple processing calls to be applied to the same frame. Instead of grabbing a new frame, all commands are applied to the current frame in memory. So you could get a histogram on all three channels of the same image and then track a color or call get mean and have these process a single buffered frame. Calling **RF** will then read a new frame into the buffer from the camera. When **BM** is off, **RF** is not required to get new frames.

See RF on page 47 to read a new frame when buffer mode is enabled.

Example of how to track multiple colors using buffer mode:

Processing on an already buffered image is much faster than processing a new image.

```
:BM 1
ACK
:PM 1
ACK
:TC 200 240 0 30 0 30
ACK
T 20 40 10 30 30 50 20 30
:RF
ACK
:TC 0 30 200 240 0 30
ACK
T 30 50 20 40 40 60 22 31
```

CR [reg1 value1 [reg2 value2 ... reg16 value16]]\r

This command sets the **C**amera's internal **R**egister values directly. The register locations and possible settings can be found in the Omnivision CMOS camera documentation. All the data sent to this command should be in decimal visible character form unless the camera has previously been set into raw mode. It is possible to send up to 16 register-value combinations. Previous register settings are not reset between CR calls; however, you may overwrite previous settings. Calling this command with no arguments resets the camera and restores the camera registers to their default state. This command can be used to hard code gain values or manipulate other low level image properties.

See page 25 for more information on YCrCb color space.

Register		Value	Effect
5	Contrast	0-255	
6	Brightness	0-255	
18	Color Mode		
		36	YCrCb Auto White Balance On
		32	YCrCb Auto White Balance Off
		44	RGB Auto White Balance On
		40	*RGB Auto White Balance Off
17	Clock Speed		
		0	*50 fps
		1	26 fps
		2	17 fps
		3	13 fps
		4	11 fps
		5	9 fps
		6	8 fps
		7	7 fps
		8	6 fps
		10	5 fps
19	Auto Exposure		
		32	Auto gain off
		33	*Auto gain on

* indicates the default state

Example of switching into YCrCb mode with White Balance off

```
:CR 18 32
ACK
:
```

CP boolean \r

This command toggles the **C**amera module's **P**ower. A value of 0, puts the camera module into a power down mode. A value of 1 turns the camera back on while maintaining the current camera register values. This should be used in situations where battery life needs to be extended, while the camera is not actively processing image data. Images in the frame buffer may become corrupt when the camera is powered down.

See SL and SD on pages 49 and 50 to decrease camera power consumption even more.

CT boolean \r

This command toggles the **C**amera **T**ype while the camera is in slave mode. Since the CMUcam2 can not determine the type of the camera without communicating with the module, it is not possible for it to auto-detect the camera type in slave mode. A value of 0, sets the CMUcam2 into ov6620 mode. A value of 1 sets it into ov7620 mode. The default slave mode startup value assumes the ov6620.

See slave mode on page 22.

DC value \r

This command sets the **C**hannel that is used for frame **D**ifferencing commands. A value of 0, sets the frame differencing commands LF and FD to use the red (Cr) channel. A value of 1 (default) sets them to use the green (Y) channel, and 2 sets them to use the blue (Cb) channel.

See LF and FD on page 39 and page 34.

DM value \r

This command sets the **D**elay **M**ode which controls the delay between characters that are transmitted over the serial port. This can give slower processors the time they need to handle serial data. The value should be set between 0 and 255. A value of 0 (default) has no delay and 255 sets the maximum delay. Each delay unit is equal to the transfer time of one bit at the current baud rate.

DS x_factor y_factor \r

This command allows **D**own **S**ampling of the image being processed. An x_factor of 1 (default) means that there is no change in horizontal resolution. An x_factor of 2, means that the horizontal resolution is effectively halved. So all commands, like send frame and track color, will operate at this lower down sampled resolution. This gives you some speed increase and reduces the amount of data sent in the send frame and bitmap linemodes without clipping the image like virtual windowing would. Similarly, the y_factor independently controls the vertical resolution. (Increasing the y_factor downsampling gives more of a speed increase than changing the x_factor.) The virtual window is reset to the full size whenever this command is called.

Example of down sampling the resolution by a factor of 2 on both the horizontal and vertical dimension.

```
:DS 2 2
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2
```

FD threshold \r

See LF on page 39 to load a new baseline frame to difference off of.

This command calls **F**rame **D**ifferencing against the last loaded frame using the **LF** command. It returns a type T packet containing the middle mass, bounding box, pixel count and confidence of any change since the previously loaded frame. It does this by calculating the average color intensity of an 8x8 grid of 64 regions on the image and comparing those plus or minus the user assigned *threshold*. So the larger the threshold, the less sensitive the camera will be towards differences in the image. Usually values between 5 and 20 yield good results. (In high resolution mode a 16x16 grid is used with 256 regions.)

See MD on page 44 to see how to reduce motion noise.

FS boolean \r

This command sets the **F**rame **S**treaming mode of the camera. A value of 1, enables frame streaming, while a 0 (default) disables it. When frame streaming is active, a send frame command will continuously send frames back to back out the serial connection.

See SF on page 50.

GB \r

See Demo Mode on page 15.

This command **G**ets a **B**utton press if one has been detected. This command returns either a 1 or a 0. If a 1 is returned, this means that the button was pressed sometime since the last call to Get Button. If a 0 is returned, then no button press was detected.

GH <channel> \r

See HC and HT commands on pages 38 and 39.

This command **G**ets a **H**istogram of the *channel* specified by the user. The histogram contains 28 bins each holding the number of pixels that occurred within that bin's range of color values. So bin 0 on channel 0 would contain the number of red pixels that were between 16 and 23 in value. If no arguments are given, get histogram uses the last channel passed to get histogram. If get histogram is first called with no arguments, the green channel is used. The value returned in each bin is the number of pixels in that bin divided by the total number of pixels times 256 and capped at 255.

GI \r

This command **G**ets the auxiliary I/O **I**nput values. When get inputs is called, a byte is returned containing the values of the auxiliary IO pins. This can be used to read digital inputs connected to the auxiliary I/O port. The aux I/O pins are internally lightly pulled high. See page 22 for pin numbering. Note that the pins are pulled up internally by the processor.

Example of how to read the auxiliary I/O pins. (in this case, pins 0 and 1 are high, while pins 2 and 3 are low).

```
:GI
3
ACK
:
```

GM \r

This command will **G**et the **M**ean color value in the current image. If, optionally, a subregion of the image is selected via virtual windowing, this function will only operate on the selected region. The mean values will be between 16 and 240 due to the limits of each color channel on the CMOS camera. It will also return a measure of the average absolute deviation of color found in that region. The mean together with the deviation can be a useful tool for automated tracking or detecting change in a scene. In YCrCb mode RGB maps to CrYCb.

See page 45 to see how the OM command can create a custom S Packet.

This command returns a Type S data packet that by default has the following parameters:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation\r

Example of how to grab the mean color of the entire window:

```
:SW 1 1 40 143
ACK
:GM
ACK
S 89 90 67 5 6 3
S 89 91 67 5 6 2
```

GS servo \r

This command will **G**et the last position that was sent to the **S**ervos.

See SV command on page 53.

Example of how to use get servo:

```
:GS 1
ACK
128
:
```

GT \r

This command **G**ets the current **T**rack color values. This is a useful way to see what color values track window is using.

This example shows how to get the current tracking values:

```
:TW
 ACK
 T 12 34 ....
:GT
ACK
200 16 16 240 20 20
:
```

GV \r

This command **G**ets the current **V**ersion of the firmware and camera module version from the camera. It returns an ACK followed by the firmware version string. c6 means that it detects an OV6620, while c7 means that it detected an OV7620.

Example of how to ask for the firmware version and camera type:

```
:GV
ACK
CMUcam2 v1.00 c6
```

GW \r

This command **G**ets the current virtual **W**indowing values. This command allows you to confirm your current window configuration. It returns the x1, y1, x2 and y2 values that bound the current window.

HC #_of_bins scale \r

See GH on page 35 to see how to get histograms.

This command lets you **C**onfigure the **H**istogram settings. The first parameter takes one of three possible values. A value of 0 (default) will cause GH to output 28 bins. A value of 1 will generate 14 bins and a value of 2 will generate 7 bins. The scale parameter (default 0) allows you to better examine bins with smaller counts. Bin values are scaled by 2^{scale} where scale is the second parameter of the command.

#_of_bins

Input	Bins
0	28
1	14
2	7

HD boolean \r

See LF and FD on pages 39 and 34 to see how to use the more basic frame differencing commands.

This command enables or disables **H**iRes frame **D**ifferencing. A value of 0 (default) disables the high resolution frame differencing mode, while a value of 1 enables it. When enabled, frame differencing will operate at 16x16 instead of 8x8. The captured image is still stored internally at 8x8. The extra resolution is achieved by doing 4 smaller comparisons against each internally stored pixel. This will only yield good results when the background image is relatively smooth, or has a uniform color.

HR state \r

This sets the camera into **HiR**es mode. This is only available using the OV6620 camera module. A *state* value of 0 (default) gives you the standard 88x143, while 1 gives you 176x287. HiRes mode truncates the image to 176x255 for tracking so that the value does not overflow 8 bits.

HT boolean \r

See GH on page 35 to see how to get a histogram.

This command enables or disables **H**istogram **T**racking. When histogram tracking is enabled, only values that are within the color tracking bounds will be displayed in the histograms. This allows you to select exact color ranges giving you more detail, and ignoring any other background influences. A value of 0 (default) will disable histogram tracking, while a value of 1 will enable it. Note that the tracking noise filter applies just like it does with the TC and TW commands.

L0 boolean \r
L1 boolean \r

These commands enable and disable the two tracking **L**EDs. A value of 0 will turn the LED off, while a value of 1 will turn it on. A value of 2 (default) will leave the LED in automatic mode. In this mode, LED 1 turns on when the camera confidently detects an object while tracking and provides feedback during a send frame. In automatic mode, LED 0 does nothing, so it can be manually set.

LF \r

See FD on page 34.

This command **L**oads a new **F**rame into the processor's memory to be differenced from. This does not have anything to do with the camera's frame buffer. It simply loads a baseline image for motion differencing and motion tracking.

LM type mode \r

This command enables **L**ine **M**ode which transmits more detailed data about the image. It adds prefix data onto either **T** or **S** packets. This mode is intended for users who wish to do more complex image processing on less reduced data. Due to the higher rate of data, this may not be suitable for many slower microcontrollers. These are the different types and modes that line mode applies to different processing functions:

Type	Mode	Effected Command	Description
0	0	TC TW	Default where line mode is disabled
0	1	TC TW	Sends a binary image of the pixels being tracked
0	2	TC TW	Sends the Mean, Min, Max, confidence and count for every horizontal line of the tracked image.
1	0	GM	Default where line mode is disabled
1	1	GM	Sends the mean values for every line in the image
1	2	GM	Sends the mean values and the deviations for every line being tracked in the image
2	0	FD	Default where line mode is disabled
2	1	FD	Returns a bitmap of tracked pixels much like type 0 mode 0 of track color
2	2	FD	Sends the difference between the current image pixel value and the stored image. This gives you delta frame differenced images.
2	3	LF FD	This gives you the actual averaged value for each element in a differenced frame. It also returns these values when you load in a new frame. This can be used to give a very high speed gray scale low resolution stream of images.

Note, that the “mode” of each “type” of linemode can be controlled independently.

Line Mode Type 0: Track Color

Mode 1: Bitmap of tracked region

When the linemode type is 0 and the mode is set to 1, TC or TW will send a binary bitmap of the image as it is being processed. It will start this bitmap with an 0xAA flag value (hex value AA not in human readable form) followed by the Xsize and Ysize of the binary image. The value 0xAA will not occur in the data stream. This is followed by bytes each of which contains the binary value of 8 pixels being streamed from the top-left to the bottom-right of the image. The bits for each row are padded with zeros to fill an integral number of bytes. The binary bitmap is terminated by two 0xAA's. This is then followed by the normally expected standard **T** data packet.

See TC on page 53.

Example of TC with line mode on:

```
:LM 0 1
ACK
:TC
ACK
(raw data: AA Xsize Ysize XX XX …. XX XX XX AA AA) T 55 90 45 72 65 106 18 51
(raw data: AA Xsize Ysize XX XX …. XX XX XX AA AA) T 55 90 46 72 65 106 18 52
```

Mode 2: Per row statistics in the tracked region

When the linemode type is 0 and the mode is set to 2, TC or TW will send various statistics about each row that is being tracked. It sends the minimium x value, the maximium x value, the average x value, the count of tracked pixels on that line and the confidence. This can be especially useful for line following applications since you can essentially get a trace of the middle of the line. Like other linemode options, this new data is sent as a prefixed packet. The packet starts with an 0xFE, followed by the number of rows (the y-size) that it will send. The packet will then contain, the xLineMean, xLineMin, xLineMax, line pixel count, and line confidence for each row. These will all be sent as raw values. The packet terminates with a 0xFD followed by a normal T packet.

See OM on page 45 to see how to mask these line mode data packets.

0xFE y-size xLineMean xLineMin xLineMax LineCount Conf … 0xFD Tpacket

Line Mode Type 1: Get Mean

Mode 1: Per line statistics

See GM on page 36.

When the linemode type is 1 and the mode is set to 1, GM will send a raw (not human readable) mean value of every line being processed. These packets start with an 0xFE. The data is sent in the following raw format rLineMean, gLineMean, bLineMean, and terminate with an 0xFD.

0xFE Rmean Gmean Bmean ... 0xFD Mpacket

Example of GM with line mode on

```
:LM 1 1
ACK
:GM
ACK
(raw data: FE XX XX XX … XX XX XX FD) M 45 56 34 10 15 8
(raw data: FE XX XX XX … XX XX XX FD) M 45 56 34 10 15 8
```

See OM on page 45 to see how to mask these line mode data packets.

Mode 2: More per line statistics

When the linemode type is 1 and the mode is set to 2, GM will send a raw (not human readable) mean value and deviation for every line being processed. These packets are started with an 0xFE. The data is sent in the following raw format rLineMean, gLineMean, bLineMean, rDeviation, gDeviation, bDeviation and terminate with an 0xFD.

0xFE Rmean Gmean Bmean Rdev Gdev Bdev ... 0xFD Mpacket

Line Mode Type 2: Frame Differencing

Mode 1: Bitmap for tracked pixels

See FD on page 34.

When the linemode type is 2 and the mode is set to 1, FD will send a binary bitmap of the image as it is being processed. It will start this bitmap with an 0xAA flag value (hex value AA not in human readable form) followed by the Xsize and Ysize of the binary image. The value 0xAA will not occur in the data stream. This is followed by bytes each of which contains the binary value of 8 (or 16) pixels being streamed from the top-left to the bottom-right of the image. The binary bitmap is terminated by two 0xAA's. This is then followed by the normally expected standard **T** data packet.

Example of TC with line mode on:

```
:LM 2 1
ACK
:FD 10
ACK
(raw data: AA XX XX XX … XX XX XX AA AA) T 5 10 4 9 8 100
(raw data: AA XX XX XX … XX XX XX AA AA) T 5 10 4 9 8 100
```

Mode 2: Deltas between reference frame

When the linemode type is 2 and the mode is set to 2, FD will send the values of the differences between the current image and the original saved frame. The packet starts with 0xFC followed by the xSize and ySize of the image buffer that is to be sent. A single value for each pixel is transmitted and the packet ends with an 0xFD. The delta values are capped at +/- 112 with 128 added to the delta, so 128 means zero difference. This forces the values to remain in the 16-240 range.

```
:LM 2 2
ACK
:FD 10
ACK
(raw data: FC xSize ySize  XX XX XX … XX XX XX FD)
(raw data: FC xSize ySize  XX XX XX … XX XX XX FD)
```

Mode 3: Deltas between reference frame

When the linemode type is 2 and the mode is set to 3, LF and FD will send a binary bitmap of the internally stored image that they are operating on. This image is stored in the same format as mode 2 of frame differencing.

MD threshold \r

See FD on page 34.

This command is almost identical to FD except that it **M**asks the first frame it **D**ifferences on. Any motion detected on the first frame is masked out, so that areas with high amounts of noise are ignored. Basically, if you call frame differencing and there is always an area of the frame that is moving, then MD will mask out that portion of the image so subsequent calls to FD will ignore that portion of the image. Calling the LF command will clear any masked pixels.

NF threshold \r

This command controls the **N**oise **F**ilter setting. It accepts a value that determines how many consecutive active pixels before the current pixel are required before the pixel should be detected (default 2). The range is between 0 and 255.

Example of how to turn off noise filtering:

```
:NF 0
ACK
:
```

OM packet mask \r

This command sets the **O**utput **M**ask for various packets. The first argument sets the type of packet:

#	**Tracking Type**	**Packet**
0	Track Color	T
1	Get Mean	S
2	Frame Difference	T
3	Non-tracked packets*	T
4	Additional Count Information**	T, H
5	Track Color Line Mode 2	T
6	Get Mean Line Modes 1 and 2	S

The *mask* should be a single byte that represents the bitwise mask of the tracking packet. So a value 255 would allow all the parameters to be printed, while a value of 3 would only allow the first two parameters to be printed. Each mask for each paket type is stored separately and remains set until the camera is reset.

*Non-Tracked packets are packets that are printed when the object being tracked is not detected. If this is set to 0, then no packet is printed when the object is not found. If this is set to 1, then just a "T 0" is sent when no object is found. If this is set to 2 (default) then the packet is identical to a tracked packet of that type.

**The additional count information flag lets you get access to the full 16 bit count values for color tracking or histogramming. A value of 0 (default) disables the 16 bit values. A value of 1, adds the 16 bit count of tracked pixels in 2 separate bytes, the first for the LSB and the second for the MSB. A value of 2 will add a 16 bit count of all pixels used to generate a histogram as the first two bytes following the H in the histogram packet. A value of 3, enables both modes simultaneously.

Example of how to only show Mx and My in a T packet:

```
:OM 0 3
ACK
:TC 200 230 0 30 0 30
T 23 45
```

PD boolean \r

See TI on page 53 to find out how to use inverse tracking for better edge following.

This command enables the **P**ixel **D**ifference mode. By default, the mode is off. A value of 1 causes the difference between the current pixel and the previous pixel to be used by all processing commands instead of the original pixel value. This essentially does a horizontal edge detecting convolution on the image. So the intensity of the remaining lines in each channel is proportional to the sharpness of an edge found in that channel. The best way to understand this command is to try enabling pixel differencing and try sending a frame. Notice what types of lines appear stronger than others. You can then track these edges based on their intensity using track color etc. The difference values are capped at +/- 112 with 128 added to each delta so a value of 128 indicates a 0 difference. This forces the values to remain between 16 and 240. This command applies to all commands.

PF boolean \r

This command enables the **P**acket **F**iltering mode. By default, the mode is off. A value of 1 makes it so that only the first empty packet when a tracked object disappears from the screen is displayed. No packets will be transmitted until the object returns into view. This command can help in situations where empty packets may unnecessarily tax the host processor.

PM mode \r

This command puts the board into **P**oll **M**ode. Setting the mode parameter to 1 engages poll mode while 0 (default) turns it off. When poll mode is set to 0, a continous stream of packets is returned from a processing function. When poll mode is set to a value of 1, only one packet is returned when an image processing function is called. If mode is set to a value of 2, then poll mode will wait until an object is tracked and then return. This could be useful if you would like to rapidly change parameters or if you have a slow processor that can't keep up with a given frame rate.

Example of how to get one packet at a time:

```
:PM 1
ACK
:TC 50 20 90 130 70 255
ACK
C 38 82 53 128 35 98
:
```

PS number \r

This command controls if **P**ackets should be **S**kipped or not. The default value is 0, which means that all packets will be transmitted. A value of 1 means that every other packet will be skipped. A value of 2 means that only every second packet will be displayed etc. This is useful if you need to slow down the data rate so that your processor can keep up with the data stream when poll mode is enabled.

RF \r

See BM on page 30.

This command **R**eads a new **F**rame into the buffer. This should only be used to get new data when using buffer mode (**BM**). The frame buffer is what allows multiple pass image processing on a single frame. While in buffer mode, you are constantly reprocessing the same frame until read frame is called. Under normal non-buffer mode operation, a new frame is loaded right before a processing function is called.

RM bit_flags \r

This command is used to engage the **R**aw serial transfer **M**ode. It reads the bit values of the first 3 (lsb) bits to configure settings. All bits cleared sets the default visible ASCII mode. If bit 0 is set, then all output from the camera is in raw byte packets. The format of the data packets will be changed so as not to include spaces or be formatted as readable ASCII text. Instead you will receive a 255 valued byte at the beginning of each packet, the packet identifying character (i.e. C for a color packet) and finally the packet data.There is no \r sent after each packet, so you must use the 255 to synchronize the incoming data. Any 255 valued bytes that may be sent as part of the packet are set to 254 to avoid confusion. If bit 1 is set, the "ACK\r" and "NCK\r" confirmations are not sent. If bit 2 is set, input will be read as raw byte values, too. In this mode, after the two command byte values are sent, send 1 byte telling how many arguments are to follow. (i.e. DF followed by the raw byte value 0 for no arguments) No \r character is required.

bit_flags = B2 B1 B0

B0	Output from the camera is in raw bytes
B1	"ACK\r" and "NCK\r" confirmations are suppressed
B2	Input to the camera is in raw bytes

Example of the new packet for Track Color with Raw Mode output only:

```
:RM 1
ACK
:TC 50 100 30 90 0 30
ACK
T>#$%KFDSAG@#$
```

RS \r

This command **R**e**S**ets the vision board. Note, on reset the first character is a \r. Also keep in mind that all register values are reset to their default state.

Example of how to reset the camera:

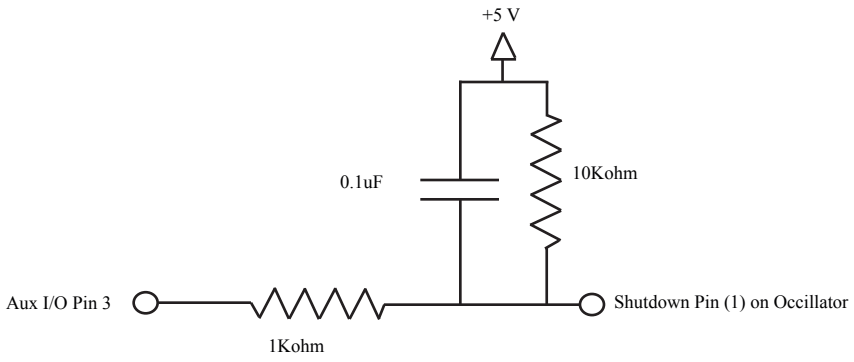
```
:RS
ACK

CMUcam2 v1.0 c6
:
```

SD \r

This command **S**leeps the camera **D**eeply to save power. This command puts the processor to sleep just as the SL command does and aditionally uses one of the auxilary I/O pins to sleep the oscillator. Wakeup from this mode is achieved by sending any character to the module, typically '\r'. The oscillator needs to shut off slightly later than the processor to ensure that that processor powers down correctly. To achieve this delay, it is neccessary to add a pullup resistor on the enable line of the oscillator and then have a resistor and capacitor in series with each other before being connected to auxiliary I/O pin 3. You will need to connect 1K series resistor between the oscillator's enable pin and the aux IO pin 3. You then need to connect a 10K resistor in parallel with a 0.1uF capacitor between the enable pin on the oscillator and +5 volts. See diagram below.

See SL on page 50, for a faster, more basic sleep command.



SF [channel] \r

This command will **S**end a **F**rame out the serial port to a computer. This is the only command that will by default only return a non-visible ASCII character packet. It dumps a type F packet that consists of the raw video data row by row with a frame synchronize byte and a column synchronize byte. (This data can be read and displayed by the CMUcam2GUI java application.) To get the correct aspect ratio, double each column of pixels. Since the image is being read from a buffer, the image resolution is not dependent on baud rate. The baud rate just controls how fast the image will be transmitted. Optionally, a channel (0-2) can be added to the command which causes send frame to only send that channel. This will effectively transmit one third of the data.

See FS on page 34, to find out how to stream frames.
See DS on page 33, to find out how to reduce data sent by send frame.

Type F data packet format flags:
1 - new frame followed by X size and Y size
2- new col
3 - end of frame
RGB (CrYCb) ranges from 16-240

1 xSize ySize 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b 3

SL active \r

For greater power saving see the SD and CP commands on pages 49 and 32.

This command enables **SL**eep mode by putting the processor to sleep. Sleep mode can be used when the camera is not needed in order to save power. Sending any character wakes the camera back up after a delay of up to 10ms. It is best to use '\r' to wake the camera up since this will ensure that no unforeseen command gets executed. Sleeping will disable the servo outputs.

SM bit_flags \r

This command sets the **S**ervo **M**ask on the CMUcam. The servo mask controls which automatic servo axes are active and which ones should report their values at the end of tracking packets. Pan and Tilt enable / disable turn off the respective automatic servo function while tracking. The servo reporting is added after all of the normal outputs in the Tracking packet, but before the final "\r". Note that automatic control only operates with 'T' packets returned by TC and TW commands.

bit_flags = B3 B2 B1 B0

B0	Pan Control Enable
B1	Tilt Control Enable
B2	Pan Report Enable
B3	Tilt Report Enable

Example of how to enable both pan and tilt automatic servoing and both pan and tilt reporting. In this case, since it doesn't see the object, the servos stay at position 128:

```
:SM 15
ACK
:TC
ACK
T 0 0 0 0 0 0 0 0 128 128
```

SO servo_number level \r

This command sets a **S**ervo **O**utput on the CMUcam to be either a constant high or low value. This essentially converts the servo outputs to be standard TTL digital outputs. The servo number (0-4) selects which servo you want to control, and a level value of either 1 or 0 switches between 5 and 0 volts. If a servo is connected and the output is set to 0, the servo is effectively turned off.

SP [pan_range_far pan_range_near pan_step
tilt_range_far tilt_range_near tilt_step] \r

This command sets the **S**ervo **P**arameters that are used by the automatic tracking control law. Changing these values can help you tune your tracking for a particular servo setup. The automatic servoing uses a two stage "bang-bang" control law. When the pixel value is greater than the "far" range, the related servo will move by the step amount. When the pixel value is between the near and far range, the servo will move by half of the step amount. Any value smaller than the near value is part of the dead zone and will not trigger any servo motion.

See page 21 for pan tilt direction jumpers.

See the SM command on page 51.

Variable	Description	Default
pan_range_far	Pixel distance needed to do a large pan step	16
pan_range_near	Pixel distance needed to do a small pan step	8
pan_step	Servo position change of a long pan step	5
tilt_range_far	Pixel distance needed to do a large tilt step	30
tilt_range_near	Pixel distance needed to do a small tilt step	15
tilt_step	Servo position change of a long tilt step	5

ST Rmin Rmax Gmin Gmax Bmin Bmax \r

This command allows you to **S**et **T**racking parameters without actually calling track color. These values can then be stored until you might call TC with no arguments later.

Example of how to use ST:

```
:ST 200 0 0 250 20 20
ACK
:TC
ACK
T 6 55 2 40 12 60 10 70
T 6 55 2 41 12 61 11 70
```

SV servo position \r

See SO on page 51 to learn how to disable the servos.

This command lets you set the position of one of five **S**er**V**os. The servos have an active region of between 46 and 210. A value of 128 is the center and generates a 1500 us pulse. The pulse increments by 8.68us and covers a range from 400 us to 1820 us.

Example to set servo 1 to position 200:

```
:SV 1 200
ACK
:
```

TC [Rmin Rmax Gmin Gmax Bmin Bmax] \r

See page 45 to see how the OM command can create a custom S Packet.

Using VW on page 55 to decrease the vertical resolution will allow 50 fps tracking.

This command begins to **T**rack a **C**olor . It takes in the minimum and maximum RGB (CrYCb) values and outputs a type T packet. This packet by default returns the middle mass x and y coordinates, the bounding box, the number of pixels tracked, and a confidence value. The packet can be masked using the **OM** output mask function. Remember that the color values from the CMOS camera will range from between 16 and 240. If TC is called with no arguments it will track with the precious set of tracking parameters.

Default Type T packet
T mx my x1 y1 x2 y2 pixels confidence\r

Example of how to Track a Color with the default mode parameters:

```
:TC 130 255 0 0 30 30
ACK
T 50 80 38 82 53 128 35 98
T 52 81 38 82 53 128 35 98
```

TI boolean \r

This command activates **T**rack **I**nverted mode. When track inverted mode is enabled, the camera will track colors that are outside of the user defined color range instead of inside. This is good for either tracking edges, or tracking any object that shows up against a homogenous background.

TW \r

This command will **T**rack the color found in the central region of the current **W**indow. After the color in the current window is grabbed, the track color function is called with those parameters and on the full image window. This can be useful for locking onto and tracking an object held in front of the camera. Since it actually calls track color, it returns the same type T track packet. Note, the current virtual window setting will only be used for grabbing the color to track and then the window will return to its maximum size.

The following internal steps are performed when "TW" is called:

1. Shrink the window to 1/2 the size (in each dimension) of the current window centered on the current window. (sw 30 54 50 90)

2. Call the get mean command but do not display the output. (gm)

3. Restore the window to the full image size. (sw 1 1 88 143)

4. Set the min and max value for each color channel to be the mean for that channel +/- 30.

Example of how to use Track Window:

```
:TW
ACK
T 6 55 2 40 12 60 10 70
T 6 55 2 41 12 61 11 70
```

UD <64 raw bytes> \r

This command allows you to **U**pload a **D**ifference frame buffer. The command waits for 64 raw byte values that fill up the 8 by 8 internal frame difference buffer. A '\r' cancels the transfer. A value of 0 indicates that the region should be masked and not detect motion. With this command in combination with line mode type 2, it is possible to download and upload different reference frames for frame differencing.

See LM on page 40 for instructions on downloading a difference buffer.

VW [x y x2 y2] \r

Do not try VW 0 0 88 144, this is outside of the 1 1 88 143 bounds.

This command sets the **V**irtual **W**indow size of the camera. It accepts the x and y Cartesian coordinates of the upper left corner (1,1) followed by the lower right of the window you wish to set. The origin is located at the upper left of the field of view. **VW** can be called before an image processing command to constrain the field of view. Without arguments it returns to the default full window size of for the current combination of camera type, downsampling and resolution mode. Note that reducing the vertical window size can be used to speed up processing time to achieve higher frame rates with the track color command. 50 fps can be achieved with a vertical dimension of 65 or less.

See GW on page 37 to find out how to check your window configuration.

Example of setting the camera to select the mid portion of the view:

```
:VW 35 65 45 75
ACK
```

Data Packet Description

When raw mode is disabled all output data packets are in ASCII viewable format except for the F frame and prefix packets.

ACK

This is the standard acknowledge string that indicates that the command was received and fits a known format.

NCK

This is the failure string that is sent when an error occurred. The only time this should be sent when an error has not occurred is during binary data packets.

Type **F** data packet format:

This packet does NOT begin with an "F" and it only contains raw data.

1 Xsize Ysize 2 r g b r g b ... r g b r g b 2 r g b r g b ... r g b r g b 3

1 - new frame 2 - new row 3 - end of frame
RGB (CrYCb) ranges from 16 - 240
RGB (CrYCb) represents two pixels color values. Each pixel shares the red and blue.
176 cols of R G B (Cr Y Cb) packets (forms 352 pixels)
144 rows
To display the correct aspect ratio, double each column so that your final image is 352x144

Type **H** packet:

H bin0 bin1 bin2 bin3 ... bin26 bin27 \r

This is the return packet from calling get histogram (**GH)**. Each bin is an 8 bit value that represents the number of pixels that fell within a set range of values on a user selected channel of the image.

Bin0 – number of pixels between 16 and 23
Bin1 – number of pixels between 24 and 31
.
.
.
Bin27 – number of pixels between 232 and 240

Type **T** packet:

T mx my x1 y1 x2 y2 pixels confidence\r

This is the return packet from a color tracking or frame differencing command.

mx - The middle of mass x value
m*y* - The middle of mass y value
x1 - The left most corner's x value
y1 - The left most corner's y value
x2 - The right most corner's x value
y2 -The right most corner's y value
pixels –Number of Pixels in the tracked region, scaled and capped at 255: (pixels+4)/8
confidence -The (# of pixels / area)*256 of the bounded rectangle and capped at 255

Type **S** data packet format:

S Rmean Gmean Bmean Rdeviation Gdeviation Bdeviation \r

This is a statistic packet that gives information about the camera's view

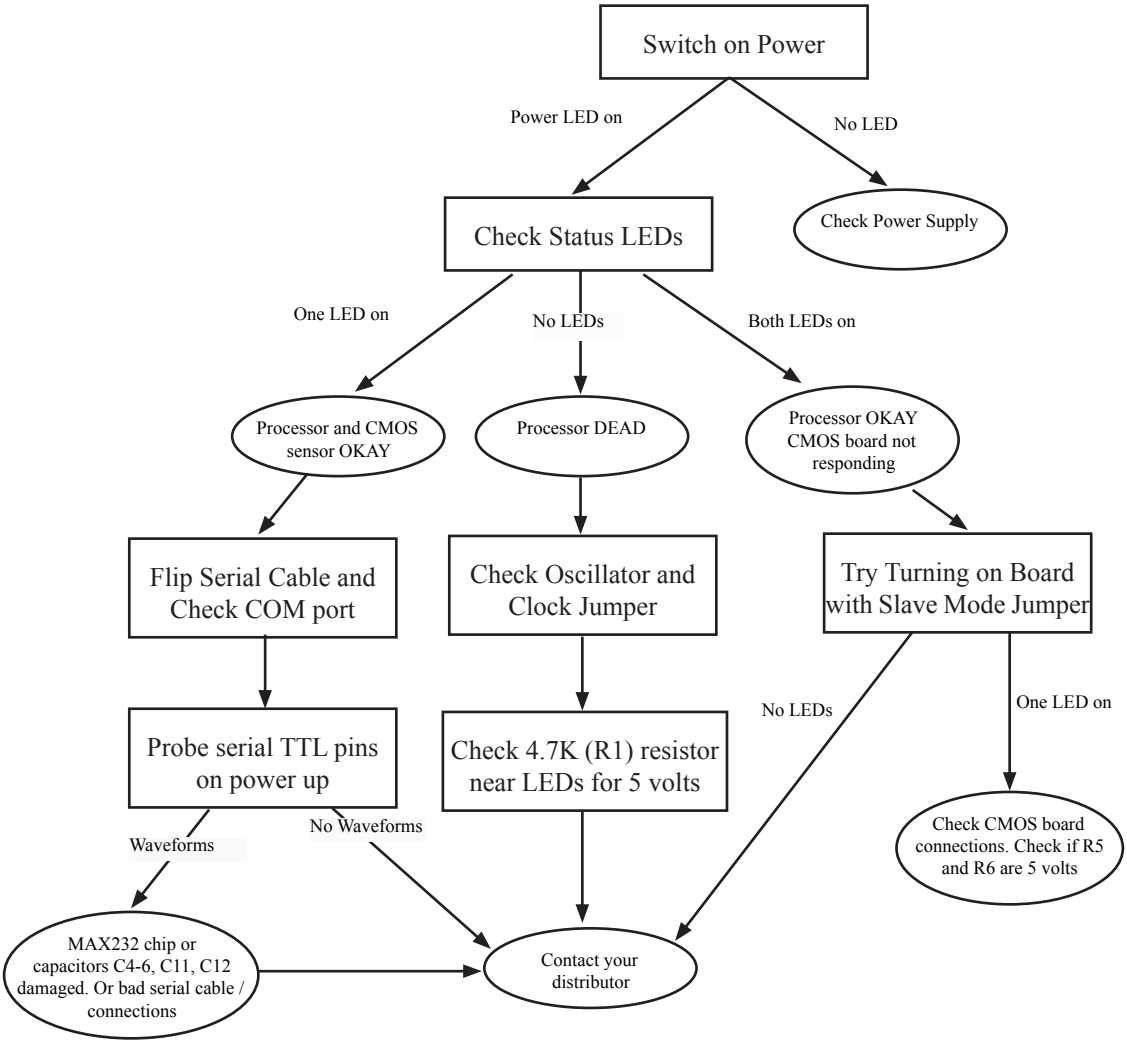
Rmean - the mean Red or Cr (approximates r-g) value in the current window
Gmean - the mean Green or Y (approximates intensity) value found in the current window
Bmean - the mean Blue or Cb (approximates b-g) found in the current window
R*deviation* - the *deviation of red or Cr found in the current window
G*deviation*- the *deviation of green or Y found in the current window
B*deviation*- the *deviation of blue or Cb found in the current window

*deviation: The mean of the absolute difference between the pixels and the region mean.

Troubleshooting

Diagnostic Fault Tree

The diagram below shows a few quick steps to help you diagnose a hardware problem with the CMUcam2.



General

In Demo Mode, the light turns on for a second and then everything stops:

When both the camera and servo are active, the power required is greater. Try using a battery or voltage source rated at a higher current.

The power LED does not glow:

The board either has a fault, or your power supply is not generating enough power. Check the power supply and look over all of the soldier connections. Try unplugging all of the cables except power and turn it on again.

I get garbage output from the camera:

Try turning the camera off and unplugging it for 10 seconds. Then plug it back in and try again. Also, make sure that the baud rate is set correctly.

I get wavy lines or a distorted black and white image when I call dumpframe:

This is most likely due to power. Make sure that you have a high enough voltage and that you are getting a clean signal. Running the camera off of fresh batteries (not an AC adaptor) is a good way to test if this is the problem.

See page 46 for poll mode and page 33 for delay mode.

My processor can not keep up with the serial data stream:

Try running the camera in poll mode and setting a delay mode value.

I don't seem to get any serial data:

Make sure that the serial cable is connected on the CMUcam side correctly. If in doubt, try reversing it.

Why does VW keep giving me a NCK?

Make sure you are within the VW 1 1 88 143 bounds.

I see the CMUcam startup message, but then nothing happens:

Check to make sure the transmit line on your serial cable is connected correctly.

CMUcam2 GUI

When I run java I get: Exception in thread "main" java.lang.NoClassDefFoundError CMUcam2GUI:

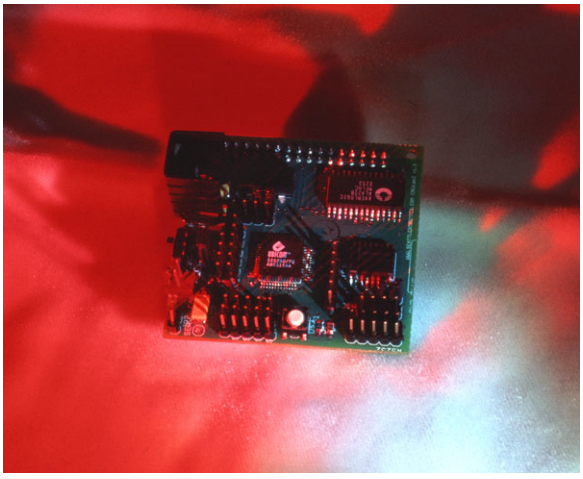
Chances are you are not in the CMUcam2GUI directory. Type "dir" at the command line prompt and make sure that you see the CMUcam2GUI.class file. Also check to make sure an old version of Quicktime did not set your CLASSPATH variable (there should be no CLASSPATH variable in new versions of java).

I see CMUcam2GUI.java but I don't see the CMUcam2GUI.class file:

You should download a new copy of the GUI, because the .class files should be included. If you really need to recompile them, type "javac *.java" .

I get: 'java' is not recognized as an internal or external command, operable program or batch file:

This means that java is not correctly installed in your path. Try re-installing java and reading Sun's installation documentation.



3rd Party Software Information

Java

The CMUcam2 GUI should not need to be recompiled!

The CMUcam2 GUI requires java's JRE version 1.4.0 or newer. The latest version of java can be downloaded for free at http://java.sun.com. JRE stands for java runtime environment and contains all that is needed to run a java program. JDK stants for java development kit and does everything that the JRE does, plus it allows you to compile java programs into byte code. Since the CMUcam2 GUI is given to you already compiled, you should be able to run it on any type of computer without having to recompile.

Terminal Emulation Programs

The following are free terminal emulation programs that can be found as shareware on the internet:

Windows:
- HyperTerm - built into windows, but tends to be confusing
- TeraTerm - Fast and easy to use

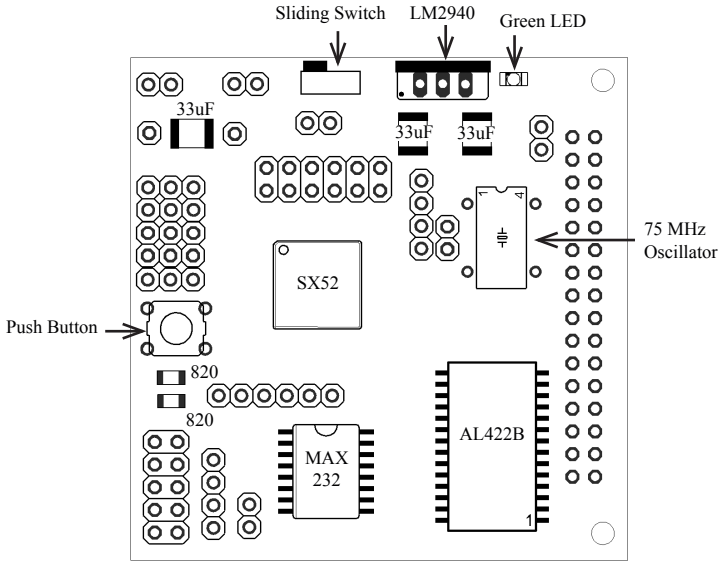
Macintosh OS 7,8,9:
- Zterm - fast, free, easy to use

Mac OS X
- Port Term

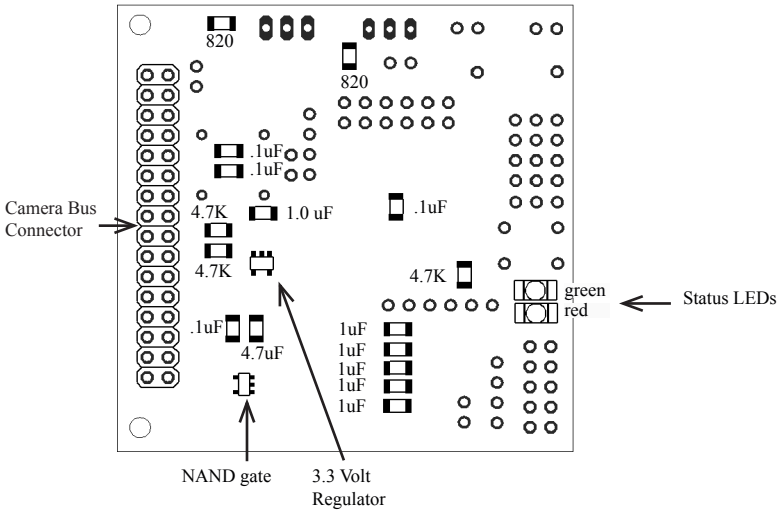
Unix / Linux:
- Minicom - standard easy to use com program
  - alt-a adds line feeds to \r
  - alt-e turns on local echo
  - alt-s lets you configure the serial port

Components and Schematic

Top Components



Bottom Components

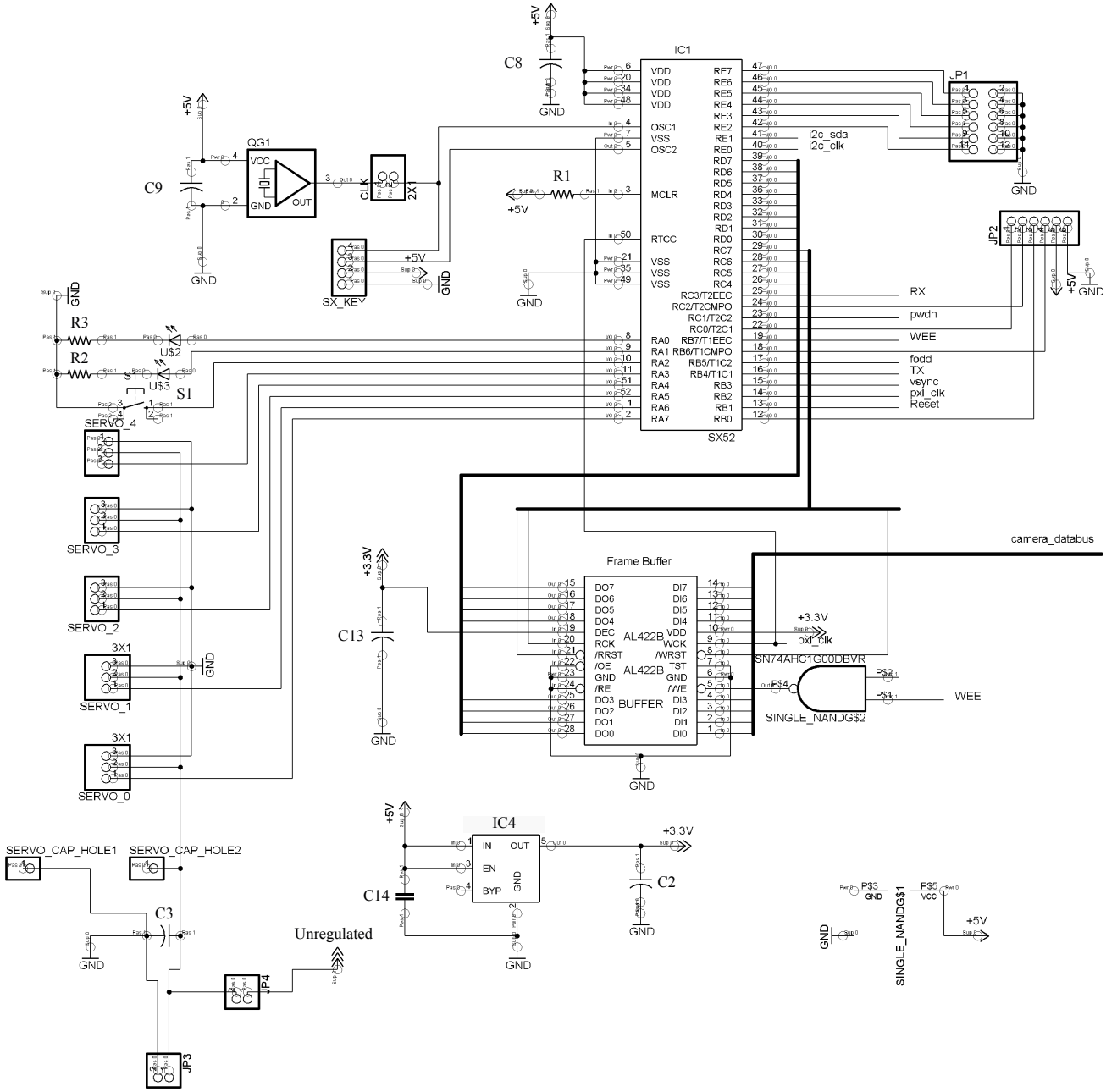


This image is a bottom view of the components. Note that the camera connector is on the left side.

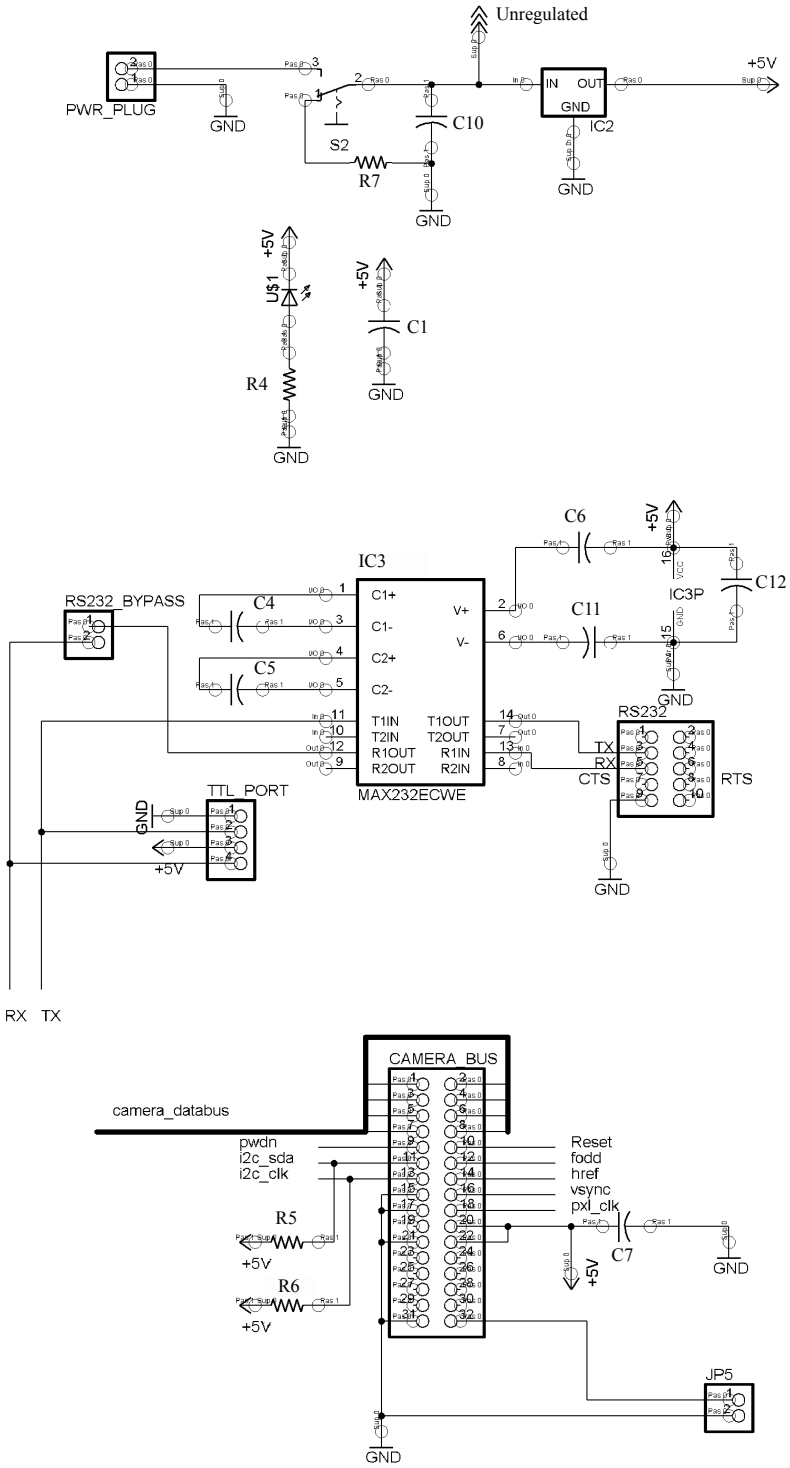
Parts List

Part	Digikey Part Number	Qty.	Schematic
High Brightness Red Led	160-1405-1-ND	1	U$2
High Brightness Green Led	160-1404-1-ND	2	U$3,U$1
820 Ohm Resistor	311-820ACT-ND	4	R2-4, R7
4.7 K resistor	311-4.7KATR-ND	3	R1,R5-6
NAND Gate	296-1087-1-ND	1	SINGLE_NANDG$2
Max 232 chip	296-13095-1-ND	1	IC3
	MAX232CWE-ND	alternate	
75 Mhz Oscillator	SG-8002DC-SHC-ND	1	QG1
5v Regulator	LM2940CT-5.0-ND	1	IC2
3.3v Regulator	LP2985IM5-3.3CT-ND	1	IC4
Averlogic AL422B		1	AL422B
Ubicom SX52		1	IC1
Slide Switch	EG1847-ND	1	S2
Push Switch	SW400-ND	1	S1
0.1uF Cap	311-1141-1-ND	4	C7-9,C13
33uF Cap	399-1634-1-ND	2	C1,C3
33uF Cap or 10uF 25V Cap	399-1634-1-ND 399-1599-1-ND	1	C10
1.0uF Cap	PCC2249CT-ND	5	C4-6,C11,C12
2.2uF Cap	PCC1923CT-ND	1	C2
Heatsink	HS333-ND	optional	
Double Female Header	929852-01-36-ND	1	CAMERA_BUS
Single Male Header	929647-09-36-ND	3	CLK, SX-KEY, RS232, PWR_PLUG, TTL_ PORT, RS232_BYPASS, JP1-2, JP4-5
Polarized 2 pin Terminal Housing	WM2700-ND	2	
Crimp Terminals	WM2200-ND	4	
Polarized 2 pin terminal header	WM2000-ND	2	PWR_PLUG, JP3
Female serial ribbon cable head	AFS09G-ND	1	
Serial Ribbon Cable Socket Connector	ASC10G-ND	1	

Schematic - Main Body



Schematic - Peripheral



Disclaimer

No warranties, either expressed or implied, are made regarding the operation, use or results of this hardware. This product is meant for educational purposes only. Any resemblance to real persons, living or dead is purely coincidental. CMUcam2 void where prohibited with some assembly required. Batteries and servos not included. Contents may settle during shipment so only use as directed. No other warranty expressed or implied. Do not use while operating a motor vehicle or heavy equipment. Apply CMUcam2 only to affected area. If condition persists, consult your physician. May be too intense for some viewers and for recreational use only. Do not disturb CMUcam2 during boot process. All models over eighteen years of age. No user-serviceable parts inside. Freshest if used before date on carton. Subject to change without notice. Many CMUcam2 times approximate and many pictures simulated. Breaking seal constitutes acceptance of agreement. This product is known to the state of California to cause birth defects. As seen on TV one size fits all. My man the yellow darts comments not actually generated by the yellow dart. Contains a substantial amount of non-tobacco ingredients. Colors may, in time, fade. Slippery when wet. If CMUcam2 acts up; keep cool; process promptly. Not responsible for direct, indirect, incidental or consequential damages resulting from any defect, error or failure to perform. Substantial penalty for early withdrawal. Keep away from fire or flame. Replace with same type. Some of the trademarks mentioned in this product appear for identification purposes only. No animals were hurt in the production of this device. This supersedes all previous notices.