

# CMUcam4 Guide

## Overview:



The CMUcam4 is a fully programmable embedded computer vision sensor. The main processor is the Parallax P8X32A (Propeller Chip) connected to an OmniVision 9665 CMOS camera sensor module. For more information please see the [wiki](#).

## Features

- Fully open source and re-programmable using the Propeller Tool
- Arduino Shield Compatible
  - w/ Supporting Interface Libraries and Demo Applications for the Arduino and BASIC Stamp
- VGA resolution (640x480) RGB565/YUV655 color sensor
  - Image processing rate of 30 frames per second
  - Raw image dumps over serial or to flash card
    - (640:320:160:80)x(480:240:120:60) image resolution
    - RGB565/YUV655 color space
- Onboard Image Processing (QQVGA 160x120)
  - Track user defined color blobs in the RGB/YUV color space
  - Mean, median, mode and standard deviation data collection – sampled from a 40x120 resolution
  - Segmented (thresholded) image capture for tracking visualization (over serial or to flash card)
    - 80x60 image resolution
    - Monochrome color space
  - Histogram generation (up to 128 Bins) – sampled from a 40x120 resolution
  - Arbitrary image clipping (windowing)

- $\mu$ SD/ $\mu$ SDHC flash card slot with FAT16/32 full file system driver support
  - w/ Directory and File manipulation
- I/O Interfaces
  - Two-port servo controller (pan and tilt w/ 1us resolution at a 50 Hz refresh rate)
    - Pan and/or Tilt servo channels can be configured as GPIOs
  - Indicator user controllable LED (red) and power LED (green)
  - TTL UART (up to 250,000 baud – 19,200 baud by default)
- **Monochrome baseband analog video output (NTSC/PAL) of 160x120 resolution for tracking visualization (segmented (thresholded) image w/ color centroid and bounding box overlay at 30 FPS)**
- CMUcam4 GUI for viewing images on the PC

## Typical Uses

---

The CMUcam4 can be used to track colors or collect basic image statistics. 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 moving beacons. Using color statistics, it is possible for the CMUcam4 to monitor a scene, detect a specific color, or do primitive motion detection. If the CMUcam4 detects a drastic color change, then chances are something in the scene changed. Using “line mode”, the CMUcam4 can generate low resolution binary images of colorful objects. This can be used to do more sophisticated image processing that includes line following with branch detection, or even simple shape recognition. These more advanced operations require custom algorithms to post process the binary images sent from the CMUcam4. As is the case with a normal digital camera, this type of processing might require a computer or at least a fast microcontroller.

## Typical Configuration

---

The most common configuration for the CMUcam4 is to have it communicate to a master processor via a standard TTL serial port. This “master processor” could be a computer (through USB or RS232), Arduino, Basic Stamp, PIC, or similar microcontroller. The CMUcam4 is small enough to add simple vision to embedded systems that can not afford the size or power of a standard computer based vision system. Its communication protocol is designed to accommodate even the slowest of processors. The CMUcam4 supports various baud rates to accommodate slower processors. For even slower processors, the CMUcam4 can operate in “poll mode”. In this mode, the host processor can ask the CMUcam4 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 communication delays, both poll mode and delay mode will lower the total number of frames that can be processed in one second.

## Quick Start

### Getting Started

---

Please follow the steps below:

1. You will need the following two items to test the CMUcam4
  - A **4V to 9V DC** external power supply capable of delivering at least **250 mA**
    - This can be from an **Arduino, FTDI Breakout Board, FTDI Cable, or Wall Wart**
  - An **NTSC TV** (or compatible television monitor) and an **RCA Cable**
2. Setup to test the CMUcam4
  - Connect the **NTSC TV** to the CMUcam4's **RCA Coaxial Jack** using the **RCA Cable**
  - Connect the external power supply to the CMUcam4's **DC Barrel Jack** (or other power ports)

The **green power LED** should illuminate once you connect the external power supply to the CMUcam4. After about **2 seconds**, the **red auxiliary LED** should illuminate to indicate the CMUcam4 is ready for action. If the **green power LED** does not illuminate please double check the external power supply connection. If the **red auxiliary LED** does not illuminate please go to the CMUcam4 forums for help.

### Testing Procedure

---

Please follow the steps below:

1. Press and hold the **reset button** on the CMUcam4
2. Press and hold the **user button** on the CMUcam4
3. Release the **reset button** (do not release the **user button**)
4. Wait until the **red auxiliary LED** turns on (**2 seconds**)
5. Wait until the **red auxiliary LED** starts blinking at **10 Hz** and then release the **user button**
  - The TV should turn on (you should see a splash screen displayed on the TV)
6. The CMUcam4 will now adjust to the lighting conditions for the next **5 seconds**

- Do not place the object you want to track in front of the CMUcam4 for the next **5 seconds**
- 7. Wait until the **red auxiliary LED** stops blinking at **10 Hz**
  - The CMUcam4 is now done adjusting to lighting conditions
  - The pan and tilt servo pins should output **1500 µs** pulses at **50 Hz**
- 8. Place the object you want to track in front of the CMUcam4 and press the **user button**
  - If the **red auxiliary LED** begins blinking at **10 Hz** examine the **OV9665** camera module connection
    - The **OV9665** camera module may be damaged and most likely needs to be replaced
- 9. You should now see the tracked object (or similar) displayed on the TV – otherwise ask for help
  - The pan and tilt servos, if connected, will also try to drive the camera towards the tracked object
- 10. Please try this procedure with different objects in different environments to see what works the best

Download the testing guide in PDF form [here](#).

## Communication Tools

---

You will need one of the following (or similar) **USB to Serial Converters** to communicate with the CMUcam4:

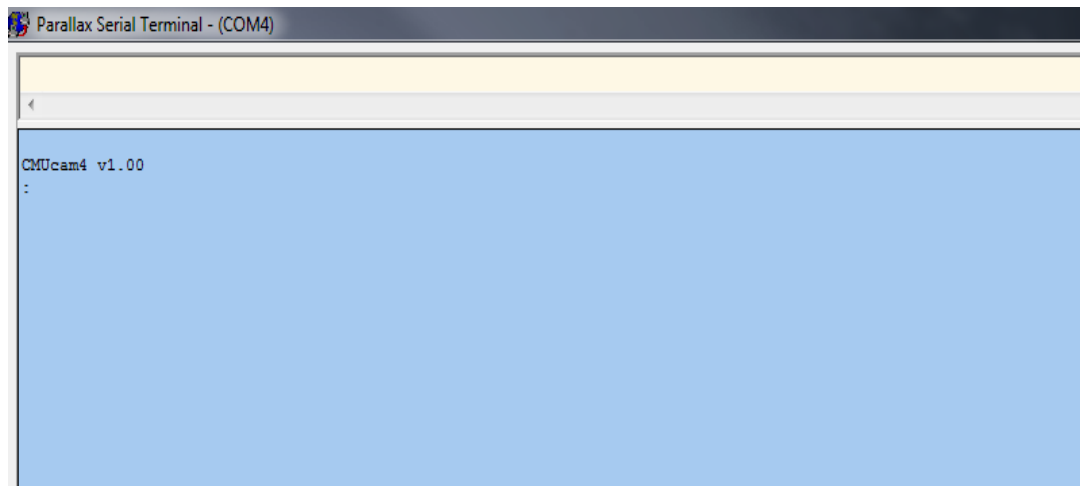
- An [FTDI 5V Breakout Board](#)
  - You will also need an external power supply capable of powering the CMUcam4
  - Please connect the **FTDI 5V Breakout Board** to the **6-pin** connector on the CMUcam4
- An [FTDI 3.3V Breakout Board](#)
  - You will also need an external power supply capable of powering the CMUcam4
  - Please connect the **FTDI 3.3V Breakout Board** to the **6-pin** connector on the CMUcam4
- A [Prop Clip](#)
  - You will also need an external power supply capable of powering the CMUcam4
  - Please connect the **Prop Clip** to the **4-pin** connector on the CMUcam4
- A [Prop Plug](#)
  - You will also need an external power supply capable of powering the CMUcam4

- Please connect the **Prop Plug** to the **4-pin** connector on the CMUcam4
- An [FTDI 5V Cable w/ 5V I/O](#)
  - **Recommended for 5V tolerant systems**
  - Please connect the **FTDI 5V Cable w/ 5V I/O** to the **6-pin** connector on the CMUcam4
- An [FTDI 5V Cable w/ 3.3V I/O](#)
  - **Recommended for 3.3V tolerant systems**
  - Please connect the **FTDI 5V Cable w/ 3.3V I/O** to the **6-pin** connector on the CMUcam4

## Recommended Serial Terminal Programs

---

The Parallax Serial Terminal is a handy tool for communication with serial-based microcontrollers such as the Parallax Propeller chip. It is the recommended serial terminal to use to communicate with the camera board. The Parallax Serial Terminal (PST) is a stand-alone application less than 1 MB in size and does not require installation to use. PST is available for download from Parallax Inc. [here](#). Follow the below steps to setup PST:



1. Run **PST**
2. Go to **Echo On** and make sure it is checked
3. Go to **Com Port** and select the COM port the **CMUcam4** is connected to from the drop-down list
4. Go to **Baud Rate** and select **19200** from the drop-down list
  - Click **Enable** if necessary

For non-Windows users Brad's SPIN Tool (BST) is recommended. BST can be downloaded [here](#). BST is a graphical user interface (GUI) integrated development environment (IDE) stand-alone application less than 10 MB in size designed for the

Parallax Propeller Chip and does not require installation to use. BST includes a built-in easy-to-use serial terminal. Follow the below steps to setup BST's built-in easy-to-use serial terminal:

1. Run **BST**
2. Go to **View** and select **Serial Terminal** from the drop-down list
  - The **bst Terminal** should pop-up – please click on it
3. Go to **Baud** and select **19200** from the drop-down list
4. Go to **Format** and select **8 Bits** and **Parity None** from the drop-down list
5. Go to **Port** and select the COM port the **CMUcam4** is connected to from the drop-down list
  - Look for ports named `/dev/tty/USB###`
6. Go to **Communicate** and select **Connect, Terminal Echo, and Reset Propeller** from the drop-down menu
  - Go to **Communicate** and select **Display ASCII** for ASCII output or **Display Hex** for hex output

**NOTE: You cannot use the FTDI 5V Cables with BST!**

*This is because **BST** pulls the **green RTS** wire low on the **FTDI 5V Cable** when you click **Connect**. This halts the **CMUcam4** indefinitely. You can remove the **green RTS** wire from the connector to fix this problem.*

## How to use the CMUcam4 properly

The CMUcam4 is an embedded computer vision system designed to track colors and to be used as a co-processor for the Arduino or equivalent microcontroller. If you've ever wanted to add computer vision to your Arduino or equivalent microcontroller powered robot, then the CMUcam4 is for you! However, if you want a general purpose computer vision system that can do more than just track colors and perform basic image statistics, then the CMUcam4 is not for you.

Computer vision systems like the Kinect can do much more than the CMUcam4. But, you can't connect the Kinect to your Arduino. If you just want to track colors and control motors with your Arduino then using the CMUcam4 with your Arduino will be far easier. Otherwise, you would have to program an application for your PC to use the Kinect and to talk to your Arduino, and an application for your Arduino to communicate to the PC and control motors. Using the CMUcam4 will save you half the work!

The CMUcam4 is also a low powered embedded computer vision system. It draws about 100 mA on average while running. This means you can connect it to your Arduino and USB port (assuming the USB port supplies up to 500 mA) without issues. The Kinect on the other hand draws over 1 A while operating. Not to mention that you'll need a PC to use it which may also draw over 1 A while operating. If power will not be an issue for you, then this paragraph is a moot point. But, for many robots power consumption is important.

Like everything else in life, the devil is in the details. The CMUcam4 was designed to interface with the Arduino from the ground up - other computer vision systems are not. So, if you want to use your Arduino with other more powerful computer vision systems then you will need to figure out the details.

### **How to Track Colors**

---

The CMUcam4 features a frame dump command called "DF" (Dump Frame). The "DF" command saves a picture of what the CMUcam4 sees to the microSD card. By using the "DF" command you can take pictures of all the colored objects you want to track with the CMUcam4. In general, you'll want to use the "DF" command to take 160x120 pictures because 160x120 is the same resolution the CMUcam4 uses to track colors. Once you have all the pictures of the colored objects you want to track you then need to use the "UM" (Unmount Disk) command to unmount the microSD card from the CMUcam4 before removing the microSD card from the CMUcam4. Otherwise, when you plug your microSD card into your computer your operating system will complain about the microSD card not being removed properly.

Anyway, once you connect your microSD card to your PC you can then look at the CMUcam4 frame dumps. The CMUcam4 saves pictures in the BMP file format so that any program will be able to open them. However, first you'll need to rotate the image by 90 degrees before being able to view it properly. After rotating the image you can then examine the pixel color values for the object you want to track using the eye dropper tool like in Microsoft Paint.

You'll want to take a few samples of the red, green, and blue color channels of the object you want to track. Try to figure out the area of color that the object spans. Eventually, you should end up with a minimum and maximum value for the red, green, and blue color channels. You'll then want to feed these minimum and maximum values back to the CMUcam4 to track the colored object. You'll probably want to increase the range of the min and max values you find to compensate for changes in lighting. Otherwise, if you have the CMUcam4 hooked up to a TV, you'll be able notice the object moving in and out of being tracked as the lighting changes on the object.

If you have a program that can display the histogram of the image like Photoshop, then you'll want to crop the image around the color of the object you want to track. By using the image histogram feature of the program you'll be able to see the distribution of the pixels in the red, blue, and green color channels. This makes figuring out what color of the object you want to track much easier. You'll also want to increase the range of colors you are tracking to compensate for changes in lighting like in the above paragraph and similarly feed the minimum and maximum red, green, and blue values of the color you want to track back to the CMUcam4 so that it tracks the object.

This calibration process requires a bit of iteration, but, the CMUcam4 is just like any other sensor, you can't expect it to magically work without effort. However, the CMUcam4's video output feature lets you see what the CMUcam4 sees in real time so that you can more quickly finish the calibration process and verify that everything is working for your application.

## Color-tracking Explanation

### **What is tracking a color and how does the CMUcam4 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. Below, we will briefly address how the CMUcam4 actually uses the information in a camera image to perform color tracking.

In order to specify color, you need to define a minimum and maximum allowable value for 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 the unique 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 CMUcam4, each color channel is converted into a number between 0 and 255, 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 CMUcam4 takes these bounds and processes the image. There are many ways to track colors in an image that can be quite complex. The CMUcam4 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 the CMUcam4 takes the horizontal sum and the vertical sum of the tracked pixels and divides each by the total number of tracked pixels and gets 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 and the number of pixels in 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. In the CMUcam4 we implement this in a way that only requires a single pass over the image. While processing the pixels in an image the CMUcam4 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.

### **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 CMUcam4, the histogram shows the frequency and distribution of

color values found in an image. Each bar represents a range of color values for a specific channel. The CMUcam4 can divide the possible color values from 0 to 255 into 1, 2, 4, 8, 16, 32, and 64 different bins. Each bin contains the number of pixels found in the image that fall within some color bounds. So a large value in one particular bin, means that many of those colors were found in the image. Each histogram only represents one select channel of color.

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 CMUcam4 with auto-gain turned off at two different textured surfaces and notice the difference in their color distributions. This effect can be used to distinguish floor surfaces or detect obstacles.

## Tips and Tricks

### Demo Mode

Demo mode allows you to demo the CMUcam4 without a master processor. In demo mode, the CMUcam4 executes the "TW" (Track Window) command and then drives two standard hobby servos towards the object being tracked while at the same time displaying the tracked object on a standard TV. Once the CMUcam4 enters demo mode it will not exit demo mode until it is reset. Follow the steps below to enter demo mode:

1. Press and hold the **reset button** on the CMUcam4
2. Press and hold the **user button** on the CMUcam4
3. Release the **reset button** (do not release the **user button**)
4. Wait until the red auxiliary LED turns on (**2 seconds**)
5. Wait until the red auxiliary LED starts blinking at **10 Hz** and then release the **user button**
  - The TV should turn on (you should see a splash screen displayed on the TV) if the CMUcam4 is connected to a TV
6. The CMUcam4 will now adjust to the lighting conditions for the next **5 seconds**
  - Do not place the object you want to track in front of the CMUcam4 for the next **5 seconds**
7. Wait until the red auxiliary LED stops blinking at **10 Hz**
  - The CMUcam4 is now done adjusting to lighting conditions
  - The pan and tilt servo pins should output **1500 µs** pulses at **50 Hz**
8. Place the object you want to track in front of the CMUcam4 and press the **user button**

- If the red auxiliary LED begins blinking at **10 Hz** examine the **OV9665** camera module connection
  - The **OV9665** camera module may be damaged and most likely needs to be replaced
- 9. You should now see the tracked object (or similar) displayed on the TV if the CMUcam4 is connected to a TV
  - The pan and tilt servos, if connected, will also try to drive the camera towards the tracked object
- 10. Please try this procedure with different objects in different environments to see what works the best
- 11. The CMUcam4 is now running in demo mode

Press the **reset button** to exit demo mode.

For non-reversed operation of the pan servo, pulse lengths lower than 1500  $\mu\text{s}$  must move the camera module's X position to the right (from the camera module's point-of-view) and pulse lengths higher than 1500  $\mu\text{s}$  must move the camera module's X position to the left (from the camera module's point-of-view).

For non-reversed operation of the tilt servo, pulse lengths lower than 1500  $\mu\text{s}$  must move the camera module's Y position down (from the camera module's point-of-view) and pulse lengths higher than 1500  $\mu\text{s}$  must move the camera module's Y position up (from the camera module's point-of-view).

## Halt Mode

Halt mode allows you to halt the CMUcam4 while still connected to an Arduino. In halt mode, the CMUcam4 draws very little power and does not prevent an Arduino from being programmed by blocking the Arduino's serial port. Halt mode is only necessary if the CMUcam4 interferes with the Arduino programming process. If it does not then halt mode is unnecessary - this is usually the case. Once the CMUcam4 enters halt mode it will not exit halt mode until it is reset. Follow the steps below to enter halt mode:

1. Press and hold the **reset button** on the CMUcam4
2. Press and hold the **user button** on the CMUcam4
3. Release the **reset button** (do not release the **user button**)
4. Wait until the red auxiliary LED turns on (**2 seconds**)
5. Release the **user button**
6. The CMUcam4 is now halted indefinitely

Press the **reset button** to exit halt mode.

## Notes on Better Tracking

### Better Tracking with 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.

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, shadows, or etc. If auto-gain and white balance were not disabled and the camera changed its settings for the RGB or YCbCr values, then the new measured values may fall outside the originally selected color tracking thresholds.

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

### YUV (YCbCr) Color Space

---

YCbCr is a different color space definition from the more commonly known RGB space. In YCbCr the illumination data is stored in a separate channel. Because of this property, in YCbCr mode the camera may be more resistant to changes in illumination. Because it is a different color space, images in YCbCr do not look like standard RGB images when directly mapped by a frame dump program. The RGB channels map to CrYCb. So in YCbCr 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

YCbCr. It should then be possible to find the Cb and Cr bounds while giving a very relaxed Y bound showing that illumination is not very important. When using YCbCr, make sure you take into account that in terms of all CMUcam4 I/O, Red maps to Cr, Green to Y, and Blue to Cb.

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

## About the Camera Module

From power up, the camera can take up to 5 seconds to automatically adjust to the light. 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 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 and light statistics, and track colors can be interwoven by the host processor to create higher level functionality.

## How to use the interface library

### Downloading and installing the interface library

You need to install the Arduino Interface Library first before being able to use it with your Arduino Environment. To do so, go to the *files* web page by clicking [here](#) and download the *CMUcam4-Arduino-Interface-Library-100* file and unzip it. You should see a single folder inside called *CMUcam4*.

Next, find and open your Arduino sketchbook folder. If there is already a folder inside of your sketchbook folder called *libraries*, then place the *CMUcam4* folder inside of the *libraries* folder. Otherwise, create a folder called *libraries* inside of your sketchbook folder and place the *CMUcam4* folder inside of the *libraries* folder.

Finally, if you currently have the Arduino IDE open, restart it. Then, if you go to the **Sketch > Import Library** menu you should see *CMUcam4* listed there. Click on the *CMUcam4* menu item to include the interface library at the top of your sketch.

We recommend that you check out the example code included with the interface library before you begin programming. You can check out our example code by going to **File > Examples > CMUcam4** and clicking on the examples listed there.

Please refer to the [CMUcam4 Manual](#) for more information about CMUcam4 commands, data packets, and error codes. This documentation assumes that you have read through the command list users manual. Additionally, please refer to the [Electrical and Component Characteristics](#) documentation for more information about the CMUcam4's power consumption and the CMUcam4's specifications.

## Connecting the CMUcam4 to your Arduino

To use the CMUcam4 with your Arduino you need to connect the CMUcam4 to one of the Arduino's serial ports. You can connect the CMUcam4 to the Arduino from either the **4-Pin Prop Clip/Plug Port**, the **6-Pin Arduino Adapter Port**, or the **2-Pin Arduino Shield Port** located on the CMUcam4. Please see the [Board Layout and Ports](#) documentation for more information about these serial ports.

For serial communication to work you need to connect one and only one **RXI** pin from the CMUcam4 to a **TXO** pin on your Arduino, one and only one **TXO** pin from the CMUcam4 to a **RXI** pin on your Arduino (the **RXI** and **TXO** pins must be from the same serial port on your Arduino), and the CMUcam4 and your Arduino must share a common ground. Additionally, you need to power the CMUcam4 with a power supply capable of delivering at least **250 mA** at between **4V to 9V DC**.

Because the CMUcam4 connects to the Arduino's serial port it may interfere with programming your Arduino. Some Arduino boards put current limiting resistors between the CMUcam4's serial port and the Arduino's serial port - like the Arduino Pro. This configuration allows the Arduino Pro to be reprogrammed when a serial programmer is connected and communicate with the CMUcam4 when a serial programmer is not connected. However, the Arduino Pro cannot communicate with the CMUcam4 while a serial programmer is connected. Other types of Arduino boards, like the Arduino Uno, put current limiting resistors between the serial programmer and the Arduino, but not between the CMUcam4 and the Arduino. This configuration allows the Arduino to be able to always communicate with the CMUcam4. However, if a serial programmer is connected to the Arduino while the CMUcam4 is connected to the Arduino then the serial programmer will not be able to communicate with the Arduino.

If you cannot reprogram your Arduino when the CMUcam4 is connected to your Arduino you can either disconnect the CMUcam4 from your Arduino or you can put the CMUcam4 into halt mode.

Halt mode allows you to halt the CMUcam4 while still connected to an Arduino. In halt mode, the CMUcam4 draws very little power and does not prevent an Arduino from being programmed by blocking the Arduino's serial port. Halt mode is only necessary if the CMUcam4 interferes with the Arduino programming process. If it does not then halt mode is unnecessary - this is usually the case. Once the CMUcam4 enters halt mode it will not exit halt mode until it is reset. Follow the steps below to enter halt mode:

1. Press and hold the **reset button** on the CMUcam4
2. Press and hold the **user button** on the CMUcam4
3. Release the **reset button** (do not release the **user button**)
4. Wait until the red auxiliary LED turns on (**2 seconds**)
5. Release the **user button**
6. The CMUcam4 is now halted indefinitely

Press the **reset button** to exit halt mode.

## Initializing the interface library and the CMUcam4

To use the CMUcam4 interface library you must first include the [CMUcam4.h](#) file at the top of your code. Next, you need to instantiate a [CMUcam4](#) interface library object in your code. You can pass either nothing when instantiating the [CMUcam4](#) interface library object or you can pass a serial port number for the serial port the [CMUcam4](#) interface library object should use. The serial port number can be either CMUCOM4\_SERIAL, CMUCOM4\_SERIAL1, CMUCOM4\_SERIAL2, or CMUCOM4\_SERIAL3 for serial ports Serial, Serial1, Serial2, and Serial3 on the Arduino and Arduino Mega (only the Arduino Mega supports serial ports Serial1, Serial2, and Serial3). If you do not pass a serial port number, or an invalid serial port number, then the interface library will use the serial port Serial by default. DO NOT use the serial port that you pass the CMUcam4 interface library after calling [CMUcam4::begin\(\)](#) and before calling [CMUcam4::end\(\)](#). Sending or receiving data on the serial port while the CMUcam4 interface library is using the serial port will cause the interface library and the CMUcam4 to become confused.

Calling the [CMUcam4::begin\(\)](#) function initializes the interface library and the CMUcam4. It changes the communication baud rate from 19,200 BPS to 115,200 BPS for CMUcam4s with firmware version 1.01 or less and changes the communication baud rate from 19,200 BPS to 250,000 BPS for CMUcam4s with firmware version 1.02 or greater. Additionally, the function changes the number of stop bits to one. If the [CMUcam4::begin\(\)](#) function is not called all CMUcam4 wrapper functions in the interface library will return the CMUCAM4\_NOT\_ACTIVATED error number (CMUcam4 wrapper functions are the functions in the CMUcam4 interface library that have the same name as the CMUcam4 commands).

Call the [CMUcam4::end\(\)](#) function to finalize the CMUcam4 interface library and release the serial port the interface library was using for use. Before using the serial port the interface library was using you should physically disconnect the CMUcam4 from the serial port the interface library was using.

Once the CMUcam4 interface library has been initialized by calling the [CMUcam4::begin\(\)](#) function you can call other functions inside the CMUcam4 interface library. For example, if you want to put the CMUcam4 to sleep to save power you can call either the [CMUcam4::sleepLightly\(\)](#) or [CMUcam4::sleepDeeply\(\)](#) functions. These functions are wrappers for the CMUcam4 "SD" (sleep deeply) and "SL" (sleep lightly) commands. The interface library has a wrapper function for every CMUcam4 command and helper functions for working with CMUcam4 data structures.

## Color tracking with the interface library

The [CMUcam4::trackColor\(\)](#), [CMUcam4::trackWindow\(\)](#), [CMUcam4::getHistogram\(\)](#), and [CMUcam4::getMean\(\)](#) functions are used to track colors with the CMUcam4 and get image statistics with the CMUcam4. When you call any one of the above functions the CMUcam4 enters stream mode and begins sending type F, H, S, and/or T data packets depending on what function you called and what mode the CMUcam4 was in. To get the data packets the CMUcam4 is sending you may call the [CMUcam4::getTypeFDataPacket\(\)](#), [CMUcam4::getTypeHDataPacket\(\)](#), [CMUcam4::getTypeSDataPacket\(\)](#), and [CMUcam4::getTypeTDataPacket\(\)](#) to get type F, H, S, and T packets the CMUcam4 is sending respectively. If you call any other function than a `getType*DataPacket()` function then the CMUcam4 will exit stream mode and you may no longer call any `getType*DataPacket()` function until you tell the CMUcam4 to enter stream mode again by calling [CMUcam4::trackColor\(\)](#), [CMUcam4::trackWindow\(\)](#), [CMUcam4::getHistogram\(\)](#), or [CMUcam4::getMean\(\)](#). All other functions in the CMUcam4 interface library do not cause the CMUcam4 to enter stream mode and may be called in any order.

When the CMUcam4 enters stream mode it begins to constantly send data packets to the Arduino. You need to call the `getType*DataPacket()` functions to get the data packets and keep the Arduino's serial buffer from overflowing while the CMUcam4 is in stream mode. Always process data packets sent by the CMUcam4 after receiving all the data packets sent by the CMUcam4 during the current image frame. For example, if line mode is enabled and the CMUcam4 is sending type T data packets followed by type F data packets then call [CMUcam4::getTypeTDataPacket\(\)](#) and then call [CMUcam4::getTypeFDataPacket\(\)](#) and then process the type T data packet and the type F data packet. DO NOT process the type T data packet before getting the type F data packet or the Arduino's serial buffer may overflow.



If you want the CMUcam4 to exit stream mode but do not want it to execute another command call the [CMUcam4::idleCamera\(\)](#) function.

All non-file system related functions have a timeout of 1 second. This means that if communication is lost with the CMUcam4 all non-file system related functions will return in a second with the CMUCAM4\_SERIAL\_TIMEOUT error. However, all file system related functions have a timeout of 1 hour because file system related commands can take an arbitrary amount of time. Be suspicious of electrical communication and/or power problems if your Arduino program appears to hang on a CMUcam4 interface library file system call. But, understand that file system related functions

like [CMUcam4::formatDisk\(\)](#) and [CMUcam4::diskSpace\(\)](#) can take several minutes or more to run.

## Arduino and Arduino Mega memory usage

The interface library uses a non-trivial amount of RAM on the Arduino and Arduino Mega. For example, Type F Data Packets require six hundred bytes of RAM. Because of this, you need to keep track of your memory usage. To learn how to do this please click [here](#) for a tutorial by Jee Labs on memory usage.

## Porting the interface library

The interface library is composed of a [CMUcam4](#) object and a [CMUcom4](#) object. The [CMUcam4](#) object is the interface library and the [CMUcom4](#) object is a platform specific functionality wrapper. Only the [CMUcom4.cpp](#) file and [CMUcom4.h](#) file need to be edited to port the interface library. The [CMUcam4](#) object calls functions inside of the [CMUcom4](#) object for serial and timer functionality.

Porting the [CMUcom4.cpp](#) and [CMUcom4.h](#) file is straight forward. Just supply each Arduino serial and timer function wrapper with the appropriate function call in your microcontroller or operating system. Additionally, you will need to edit the maximum baud rate and minimum stop bits constants in the [CMUcom4.h](#) file. Finally, you will also need to edit the command and response serial buffer sizes in the [CMUcom4.h](#) file - the command and response serial buffers do not need to be and should not be larger than 256 bytes.

The interface library assumes that chars are at least 8 bits, ints and size\_ts are at least 16 bits, and longs are at least 32 bits. These assumptions are valid for all platforms implementing standard [C data types](#). Additionally, the interface library assumes your microcontroller or operating system is little little-endian. If your operating system or microntroller does not implement standard C data types and/or

is not little endian then you will need to also edit the [CMUcam4.cpp](#) file and [CMUcam4.h](#) file to support your particular microcontroller or operating system.

## Troubleshooting

### **In Demo Mode the power light, auxiliary light, and TV turn on for a second and then everything stops**

---

When both the CMUcam4 and pan and/or tilt servos 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 or glows dimly**

---

The board either has a fault, or your power supply is not generating enough power. The power supply should be capable of delivering at least 250 mA at between 4 volts to 9 volts DC. Check the power supply and look over all of the wire connections. Try unplugging all of the wires except for the power wires and turn the CMUcam4 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.

### **I get wavy lines in my image or a distorted black and white image when I call "DF" (Dump Frame)**

---

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 adapter) is a good way to test if this is the problem. The camera module could also possibly be damaged.

### **My processor can not keep up with the serial data stream**

---

Try running the camera in poll mode using the "PM" (Poll Mode) command and setting a delay mode value using the "DM" (Delay Mode) command.

### **I don't seem to get any serial data**

---

Make sure that the serial cable is attached to the CMUcam4 correctly. If in doubt, try reversing it.

### **I see the CMUcam4 startup message, but, then nothing happens**

---
















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

### My microSD card doesn't work

MicroSD cards differ based on brand. It is possible that your brand doesn't work with the CMUcam4. It is also possible that your card has a corrupt file system. In this case, try reformatting your card and check to make sure the contacts on your board and card look clean.

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

 <u>CMUcam4</u>	The CMUcam4 class implements a generic C++ interface library for the CMUcam4
 <u>CMUcam4_directory_entry_t</u>	File or directory entry data structure
 <u>CMUcam4_disk_information_t</u>	Disk information data structure
 <u>CMUcam4_disk_space_t</u>	Disk space data structure
 <u>CMUcam4_entry_attributes_t</u>	File or directory attributes data structure
 <u>CMUcam4_histogram_data_16_t</u>	CMUcam4 16-bin histogram structure
 <u>CMUcam4_histogram_data_1_t</u>	CMUcam4 1-bin histogram structure
 <u>CMUcam4_histogram_data_2_t</u>	CMUcam4 2-bin histogram structure
 <u>CMUcam4_histogram_data_32_t</u>	CMUcam4 32-bin histogram structure
 <u>CMUcam4_histogram_data_4_t</u>	CMUcam4 4-bin histogram structure
 <u>CMUcam4_histogram_data_64_t</u>	CMUcam4 64-bin histogram structure
 <u>CMUcam4_histogram_data_8_t</u>	CMUcam4 8-bin histogram structure
 <u>CMUcam4_image_data_t</u>	CMUcam4 binary bitmap structure
 <u>CMUcam4_statistics_data_t</u>	CMUcam4 statistics data structure
 <u>CMUcam4_tracking_data_t</u>	CMUcam4 tracking data structure

<b>C</b> <a href="#"><u>CMUcam4_tracking_parameters_t</u></a>	CMUcam4 tracking parameters structure
<b>C</b> <a href="#"><u>CMUcam4_tracking_window_t</u></a>	CMUcam4 tracking window structure
<b>C</b> <a href="#"><u>CMUcom4</u></a>	This is a hardware abstraction layer for the CMUcam4 class

Here is a list of all documented class members with links to the class documentation for each member:

- a -

- [andPixels\(\)](#) : [CMUcam4](#)
- [archive](#) : [CMUcam4\\_entry\\_attributes\\_t](#)
- [attributes](#) : [CMUcam4\\_directory\\_entry\\_t](#)
- [autoGainControl\(\)](#) : [CMUcam4](#)
- [automaticPan\(\)](#) : [CMUcam4](#)
- [automaticTilt\(\)](#) : [CMUcam4](#)
- [autoPanParameters\(\)](#) : [CMUcam4](#)
- [autoTiltParameters\(\)](#) : [CMUcam4](#)
- [autoWhiteBalance\(\)](#) : [CMUcam4](#)
- [available\(\)](#) : [CMUcom4](#)

- b -

- [begin\(\)](#) : [CMUcam4](#) , [CMUcom4](#)
- [bins](#) : [CMUcam4\\_histogram\\_data\\_2\\_t](#) , [CMUcam4\\_histogram\\_data\\_32\\_t](#) , [CMUcam4\\_histogram\\_data\\_64\\_t](#) , [CMUcam4\\_histogram\\_data\\_a\\_4\\_t](#) , [CMUcam4\\_histogram\\_data\\_1\\_t](#) , [CMUcam4\\_histogram\\_data\\_a\\_8\\_t](#) , [CMUcam4\\_histogram\\_data\\_16\\_t](#)
- [blackAndWhiteMode\(\)](#) : [CMUcam4](#)
- [blueMax](#) : [CMUcam4\\_tracking\\_parameters\\_t](#)
- [blueMin](#) : [CMUcam4\\_tracking\\_parameters\\_t](#)
- [BMean](#) : [CMUcam4\\_statistics\\_data\\_t](#)
- [BMedian](#) : [CMUcam4\\_statistics\\_data\\_t](#)
- [BMode](#) : [CMUcam4\\_statistics\\_data\\_t](#)
- [bottomRightX](#) : [CMUcam4\\_tracking\\_window\\_t](#)
- [bottomRightY](#) : [CMUcam4\\_tracking\\_window\\_t](#)
- [BStDev](#) : [CMUcam4\\_statistics\\_data\\_t](#)
- [bytesPerSector](#) : [CMUcam4\\_disk\\_information\\_t](#)

- c -

- [cameraBrightness\(\)](#) : [CMUcam4](#)

- cameraContrast() : CMUcam4
- cameraRegisterRead() : CMUcam4
- cameraRegisterWrite() : CMUcam4
- changeAttributes() : CMUcam4
- changeDirectory() : CMUcam4
- CMUcam4() : CMUcam4
- CMUcom4() : CMUcom4
- colorTracking() : CMUcam4
- confidence : CMUcam4\_tracking\_data\_t
- countOfClusters : CMUcam4\_disk\_information\_t
- countOfDataSectors : CMUcam4\_disk\_information\_t

- d -

- delayMilliseconds() : CMUcom4
- directory : CMUcam4\_entry\_attributes\_t
- diskInformation() : CMUcam4
- diskSignature : CMUcam4\_disk\_information\_t
- diskSpace() : CMUcam4
- dumpBitmap() : CMUcam4
- dumpFrame() : CMUcam4

- e -

- end() : CMUcam4 , CMUcom4

- f -

- filePrint() : CMUcam4
- fileType : CMUcam4\_disk\_information\_t
- flush() : CMUcom4
- formatDisk() : CMUcam4
- freeSectorCount : CMUcam4\_disk\_space\_t

- g -

- getButtonDuration() : CMUcam4
- getButtonPressed() : CMUcam4
- getButtonReleased() : CMUcam4
- getButtonState() : CMUcam4
- getHistogram() : CMUcam4
- getInputs() : CMUcam4
- getMean() : CMUcam4
- getPixel() : CMUcam4
- getServoPosition() : CMUcam4

- `getTrackingParameters()` : CMUcam4
- `getTrackingWindow()` : CMUcam4
- `getTypeFDataPacket()` : CMUcam4
- `getTypeHDataPacket()` : CMUcam4
- `getTypeSDataPacket()` : CMUcam4
- `getTypeTDataPacket()` : CMUcam4
- `getVersion()` : CMUcam4
- `GMean` : CMUcam4\_statistics\_data\_t
- `GMedian` : CMUcam4\_statistics\_data\_t
- `GMode` : CMUcam4\_statistics\_data\_t
- `greenMax` : CMUcam4\_tracking\_parameters\_t
- `greenMin` : CMUcam4\_tracking\_parameters\_t
- `GStDev` : CMUcam4\_statistics\_data\_t

- h -

- `hidden` : CMUcam4\_entry\_attributes\_t
- `histogramTracking()` : CMUcam4
- `horizontalMirror()` : CMUcam4

- i -

- `idleCamera()` : CMUcam4
- `invertedFilter()` : CMUcam4
- `isArchive()` : CMUcam4
- `isDirectory()` : CMUcam4
- `isHidden()` : CMUcam4
- `isReadOnly()` : CMUcam4
- `isSystem()` : CMUcam4
- `isVolumeID()` : CMUcam4

- l -

- `LEDOff()` : CMUcam4
- `LEDOOn()` : CMUcam4
- `lineMode()` : CMUcam4
- `listDirectory()` : CMUcam4

- m -

- `makeDirectory()` : CMUcam4
- `milliseconds()` : CMUcom4
- `monitorFreeze()` : CMUcam4
- `monitorOff()` : CMUcam4
- `monitorOn()` : CMUcam4

- monitorSignal() : CMUcam4
- moveEntry() : CMUcam4
- mx : CMUcam4 tracking data t
- my : CMUcam4 tracking data t

- n -

- name : CMUcam4 directory entry t
- negativeMode() : CMUcam4
- noiseFilter() : CMUcam4
- notPixels() : CMUcam4
- nullTerminator : CMUcam4 entry attributes t

- o -

- orPixels() : CMUcam4

- p -

- panInput() : CMUcam4
- panOutput() : CMUcam4
- peek() : CMUcom4
- pixels : CMUcam4 tracking data t , CMUcam4 image data t
- pollMode() : CMUcam4
- printLine() : CMUcam4

- r -

- read() : CMUcom4
- readOnly : CMUcam4 entry attributes t
- redMax : CMUcam4 tracking parameters t
- redMin : CMUcam4 tracking parameters t
- removeEntry() : CMUcam4
- resetSystem() : CMUcam4
- RMean : CMUcam4 statistics data t
- RMedian : CMUcam4 statistics data t
- RMode : CMUcam4 statistics data t
- RStDev : CMUcam4 statistics data t

- s -

- sectorsPerCluster : CMUcam4 disk information t
- sendBitmap() : CMUcam4
- setFrame() : CMUcam4
- setOutputs() : CMUcam4

- setPixel() : CMUcam4
- setServoPosition() : CMUcam4
- setTrackingParameters() : CMUcam4
- setTrackingWindow() : CMUcam4
- size : CMUcam4\_directory\_entry\_t
- sleepDeeply() : CMUcam4
- sleepLightly() : CMUcam4
- switchingMode() : CMUcam4
- system : CMUcam4\_entry\_attributes\_t

- t -

- testMode() : CMUcam4
- tiltInput() : CMUcam4
- tiltOutput() : CMUcam4
- topLeftX : CMUcam4\_tracking\_window\_t
- topLeftY : CMUcam4\_tracking\_window\_t
- trackColor() : CMUcam4
- trackWindow() : CMUcam4

- u -

- unmountDisk() : CMUcam4
- usedSectorCount : CMUcam4\_disk\_space\_t

- v -

- verticalFlip() : CMUcam4
- volumeID : CMUcam4\_entry\_attributes\_t
- volumeIdentification : CMUcam4\_disk\_information\_t
- volumeLabel : CMUcam4\_disk\_information\_t

- w -

- write() : CMUcom4

- x -

- x1 : CMUcam4\_tracking\_data\_t
- x2 : CMUcam4\_tracking\_data\_t
- xorPixels() : CMUcam4

- y -

- y1 : CMUcam4\_tracking\_data\_t
- y2 : CMUcam4\_tracking\_data\_t



