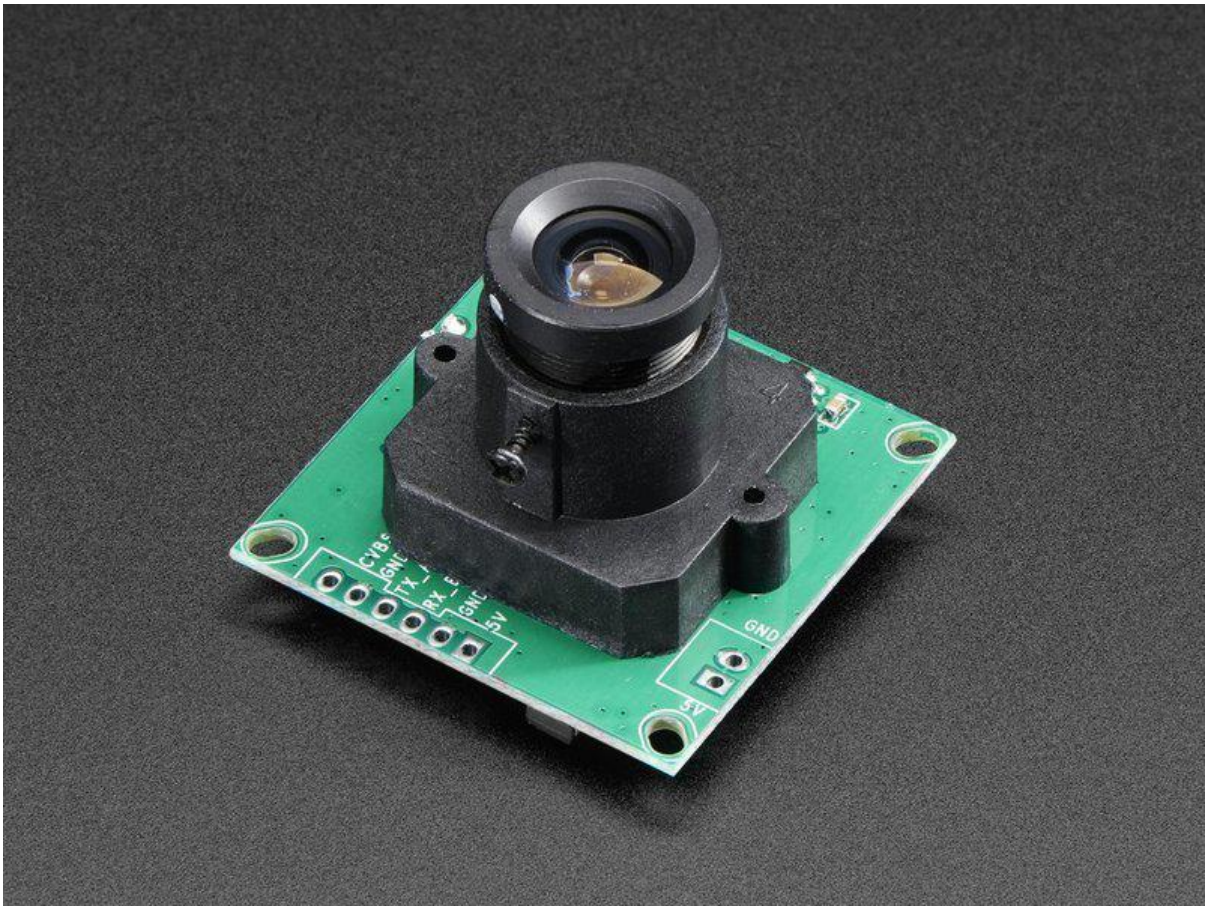
TTL Serial Camera

Created by lady ada



https://learn.adafruit.com/ttl-serial-camera

Last updated on 2023-08-29 02:07:26 PM EDT

Table of Contents

Overview 3

• Sample Images

Wiring the Camera 5

Testing the Camera 7

Using CommTool 8

• Despite the software letting you change the baud rate this is a very flaky setting and even if it works, when you power up the camera again it will reset. Some experimenters have accidentally disabled their cameras by trying to change the baud rate. We do not suggest you mess with the baud rate settings. If you do, you may permanently disable your camera and we will not replace it!

•

Arduino Usage 15

• Taking a Snapshot
• Detecting Motion
• Adjusting the Manual Focus

CircuitPython & Python Usage 18

• CircuitPython Microcontroller Wiring
• Python Computer Wiring
• CircuitPython Installation of VC0706
• Python Installation of VC0706 Library
• Microcontroller CircuitPython Usage (not for Linux/SBC)
• Saving Images to CircuitPython Internal Filesystem
• Activate Internal storage on Microcontrollers
• Example Code for saving to internal file system (CircuitPython or Linux / SBC)
• Saving Images to Computer or Raspberry Pi / Linux
• USB to Serial Converter
• Raspberry Pi / Linux

Python Docs 31

F.A.Q. 31

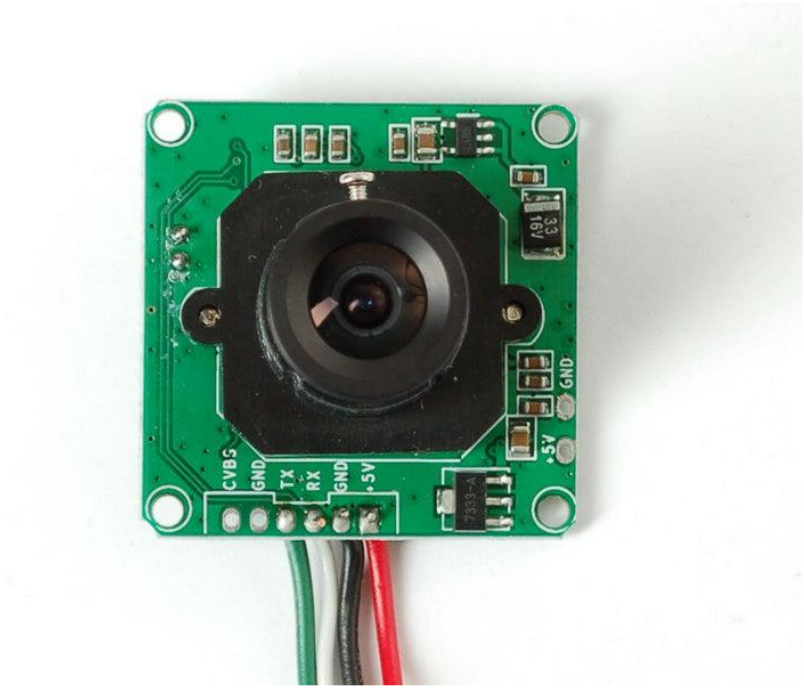
Buy a TTL Serial Camera 32

Downloads 32

• Unsupported Alternate libraries

Overview

This tutorial is for our new TTL serial camera module with NTSC video output. These modules are a nice addition to a microcontroller project when you want to take a photo or control a video stream. The modules have a few features built in, such as the ability to change the brightness/saturation/hue of images, auto-contrast and auto-brightness adjustment, and motion detection.



Since it is a little confusing how this is both a snapshot and video camera, we'd like to explain it in detail now. The module was initially designed for surveillance purposes. Its meant to constantly stream TV-resolution video out of the Video pin (this is NTSC monochrome format) and also take commands from the serial port. The serial port commands can request that the module freeze the video and then download a JPEG color image. So for example, normally its just displaying video to a security monitor. When motion is detected, it would take a photo and save it to a disk for later analysis.

The module is admittedly not extremely high resolution - the maximum image size it can take is 640x480 pixels. And it is sensitive to infrared light, which alters the color rendition somewhat. The reason for all this is that it's meant for surveillance, not for nature photography. However, as far as we can tell, this is the best module on the market.

- Module size: 32mm x 32mm
- Image sensor: CMOS 1/4 inch
- CMOS Pixels: 0.3M
- Pixel size: 5.6um*5.6um

- Output format: Standard JPEG/M-JPEG
- White balance: Automatic
- Exposure: Automatic
- Gain: Automatic
- Shutter: Electronic rolling shutter
- SNR: 45DB
- Dynamic Range: 60DB
- Max analog gain: 16DB
- Frame speed: 640*480 30fps
- Scan mode: Progressive scan
- Viewing angle: 60 degrees
- Monitoring distance: 10 meters, maximum 15meters (adjustable)
- Image size: VGA (640*480), QVGA (320*240), QQVGA (160*120)
- Baud rate: Default 38400 (the datasheet claims you can change the baud rate with a command but it does not work reliably)
- Current draw: 75mA
- Operating voltage: DC +5V
- Communication: 3.3V TTL (Three wire TX, RX, GND)

Sample Images

Here are two example images, one of outside during a cloudy day, and one inside on a sunny day.

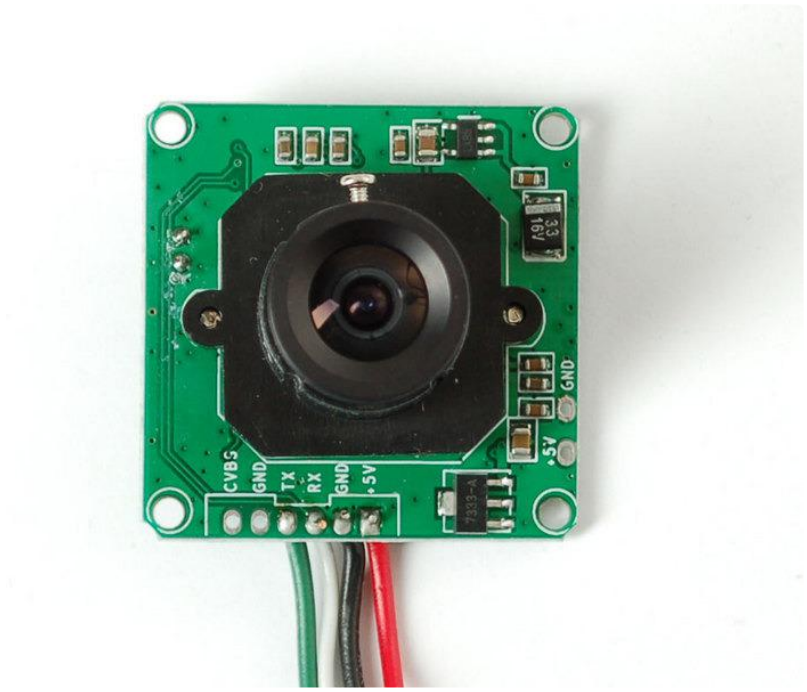


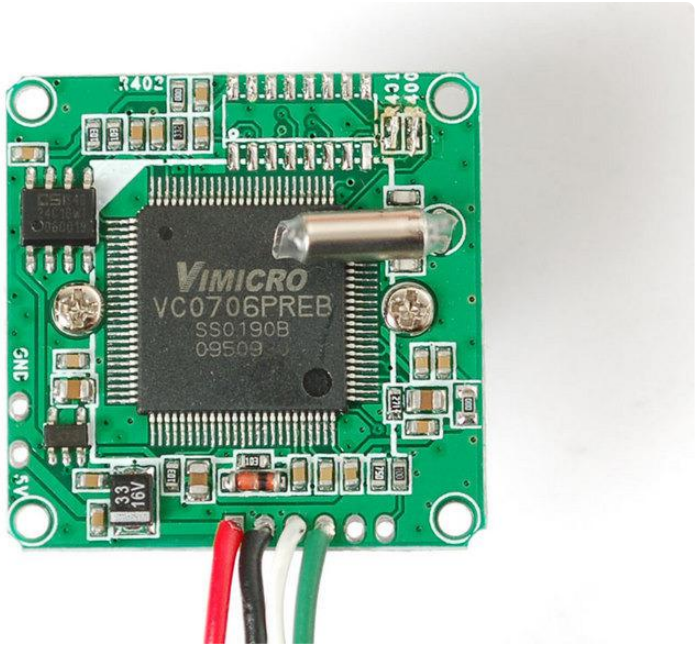


Wiring the Camera

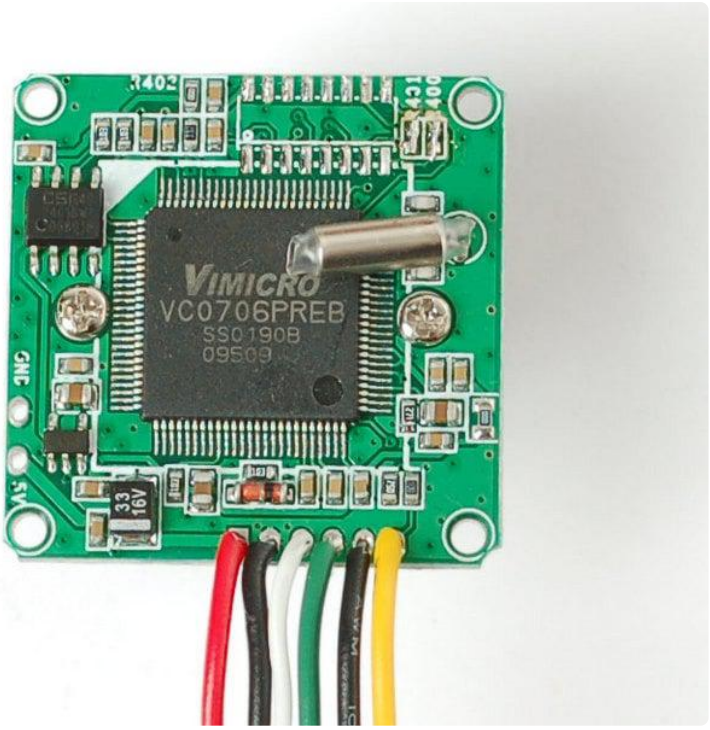
The module comes without any connector so you'll need to solder wires into the connection pads. The good news is the pads are not too close togehter (about 2mm) and you can use any stranded or solid-core wire.

If you aren't planning to use the video output abilities, you can use 4 wires. We will use red for the +5V pin, black for the Ground pin, white for the RX pin (data into the module) and green for the TX pin (data from the module)





If you'd like to get NTSC video out to connect to a TV or monitor, solder another black wire to the second Ground pin, and a yellow wire to the CVBS pin.





If you have the weatherproof version of this camera, it comes prewired with the following:

- Red is connected to +5V in
- Black is connected to Ground
- Green is RX
- White is TX
- Yellow is NTSC Video signal out
- Brown is NTSC Video ground

Testing the Camera

The quickest way to test out the modules is to use the NTSC video out connection. That way, when you adjust the view & focus you can immediately see the results. Paired with the next section (using the Comm Tool), its the ideal method of introducing yourself to the module.

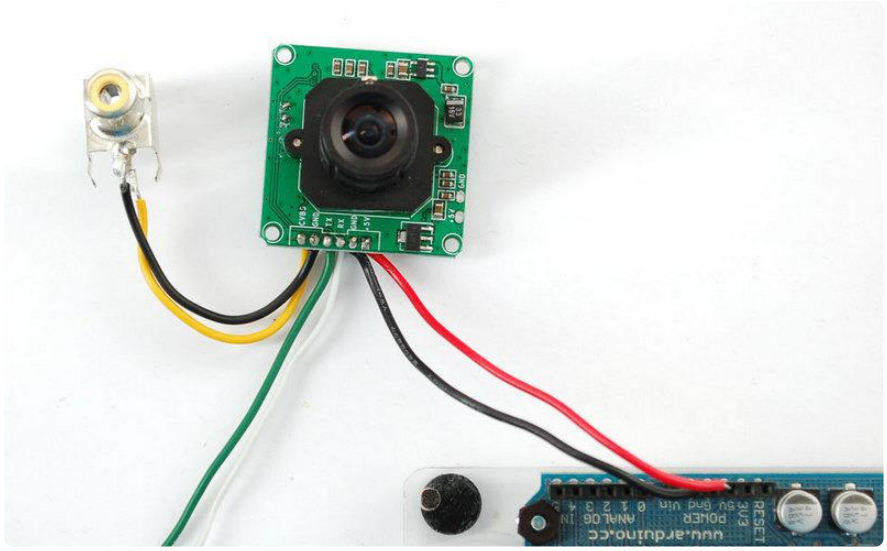
Most TV's and monitors require an RCA jack or plug input. We just soldered a spare RCA jack to the camera, with black being the case ground and yellow signal. You can get RCA cables and accessories in any hobby/electronics shop like Radio Shack.

Unfortunately, it is not possible to change the camera from NTSC to PAL - its hardcoded by a pin soldered to the board and there's no easy way to extract it and change it (we tried!)

Plug in the NTSC cable to your monitor, and connect the red and black power wires to

+5V supply - you should get monochrome video output on the monitor immediately!

We have some NTSC television modules in the Adafruit shop you can use to test with ()





Using CommTool

To use the Comm Tool, a windows utility, we need to set up a serial link to the camera. There's two ways we suggest doing this. One is to use something like an FTDI friend or other USB/TTL serial converter. If you have an Arduino you can 'hijack' the serial chip (FTDI chip or similar) by uploading a blank sketch to the Arduino:

```
// empty sketch

void setup()
{
}

void loop()
```

```
{
}
```

Note: 'Hijacking' the serial port only works on Arduinos with a separate USB interface, like the Uno. It won't work on a Leonardo!

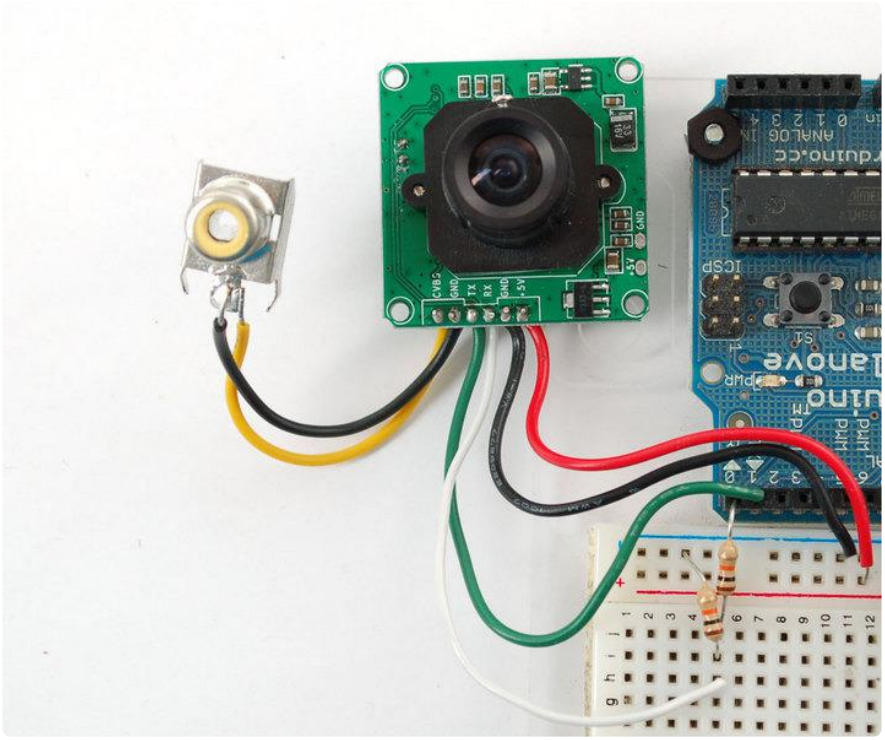
If you're using a Leonardo, Micro, Yun, or other ATmega32U4-based controller, use this Leo_passthru sketch instead of the "blank" sketch.

```
//Leo_passthru
// Allows Leonardo to pass serial data between
// fingerprint reader and Windows.
//
// Red connects to +5V
// Black connects to Ground
// Green goes to Digital 0
// White goes to Digital 1

void setup() {
  Serial1.begin(57600);
  Serial.begin(57600);
}

void loop()
{
  while (Serial.available())
    Serial1.write(Serial.read());
  while (Serial1.available())
    Serial.write(Serial1.read());
}
```

Now, wire it up as follows:

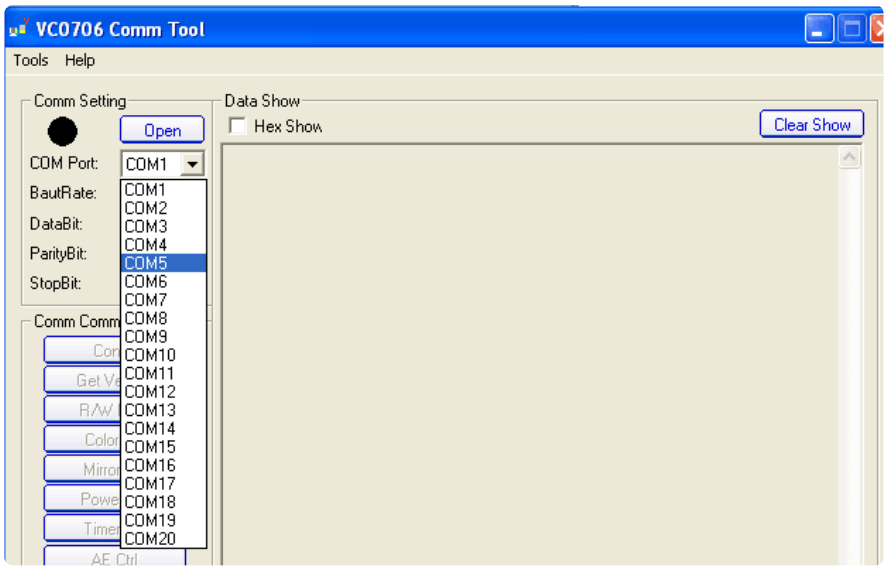


For the weatherproof camera, the white and green wires are swapped on some cameras! So please flip the white and green wires indicated if using the metal camera. Red should still be connected to +5 and Black to Ground

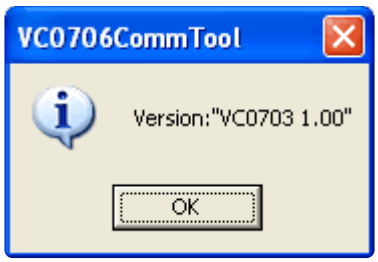
Note the 10K resistor divider, the camera's serial data pins are 3.3v logic and its a good idea to divide the 5V down so that its 2.5V. Normally the ouput from the digital 0 pin is 5V high, the way we connected the resistors is so the camera input (white wire) never goes above 3.3V

Now download and install the VC0706 CommTool software (see below in the Download section)

Start up the software and select the COM port that the Arduino is on.

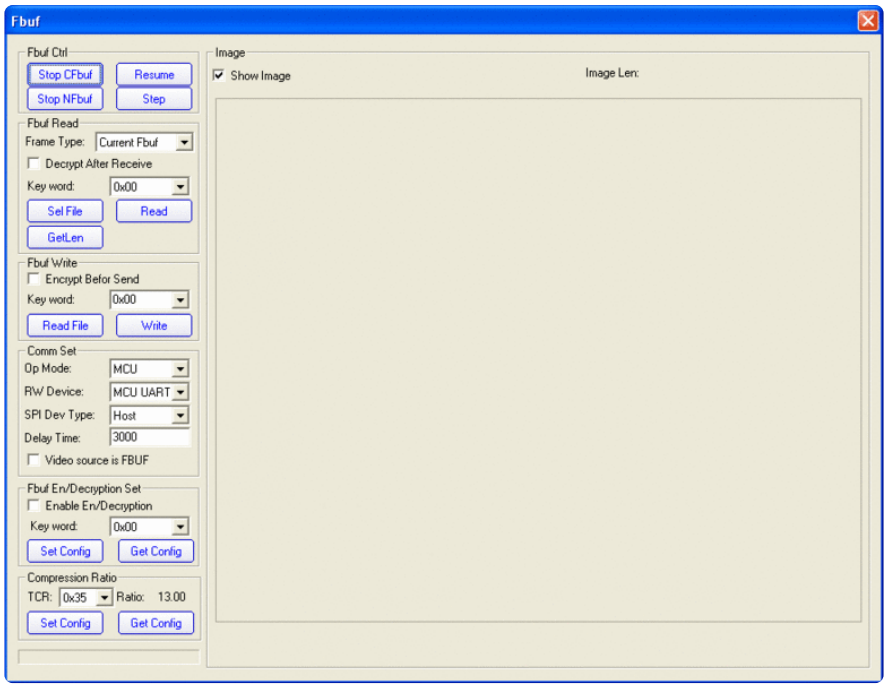


Then Open the port and click Get Version



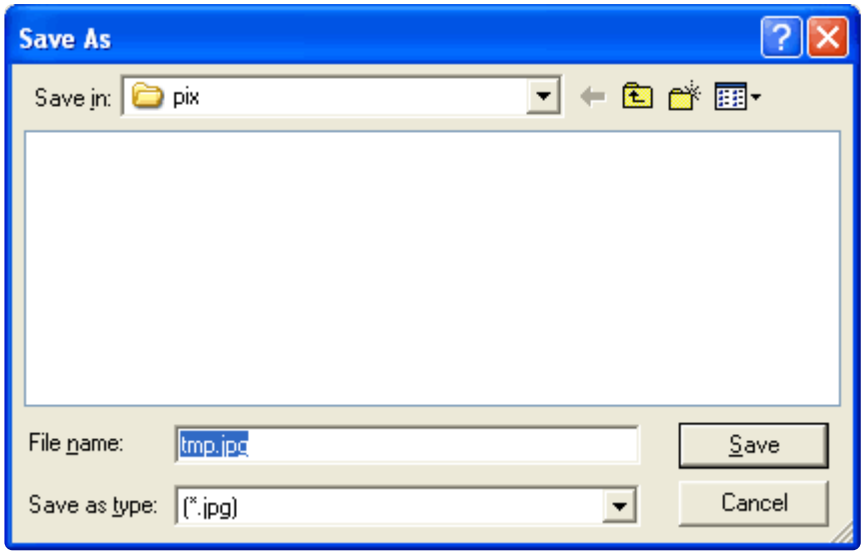
Note it says VC0703 - we don't know precisely why the DSP is programmed with a different number - its one of those mysteries! Still, you should get a response

The next button you should press is near the bottom FBUF CTRL.

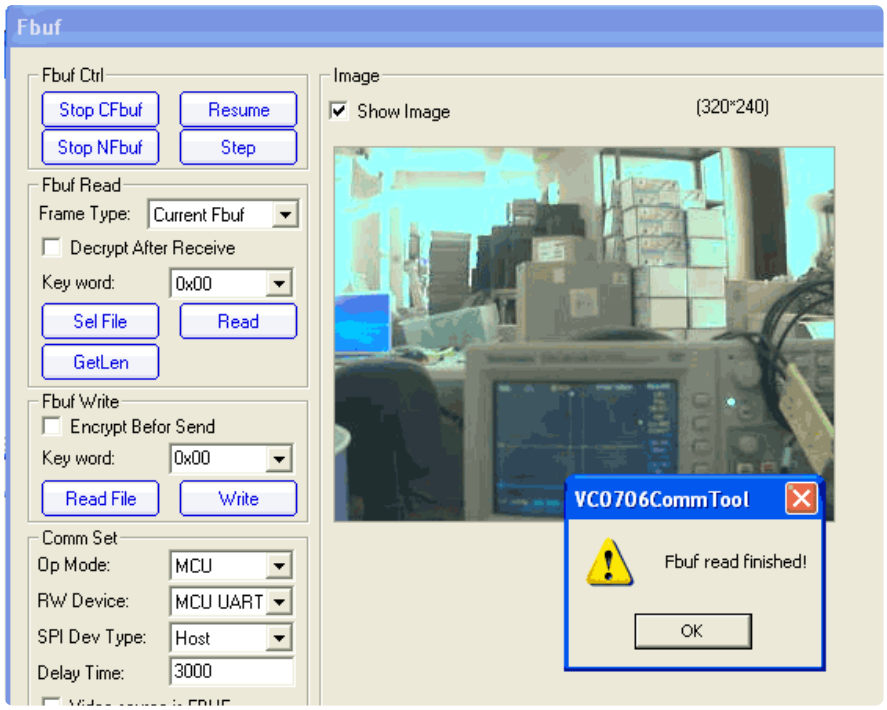


This is quite a panel, but we can use this to get images directly from the camera which is good for debugging.

- Point the camera at something you want to take a photo of
- Click Stop FBuf to freeze the frame buffer
- Click Sel File to select the file to save the JPG as



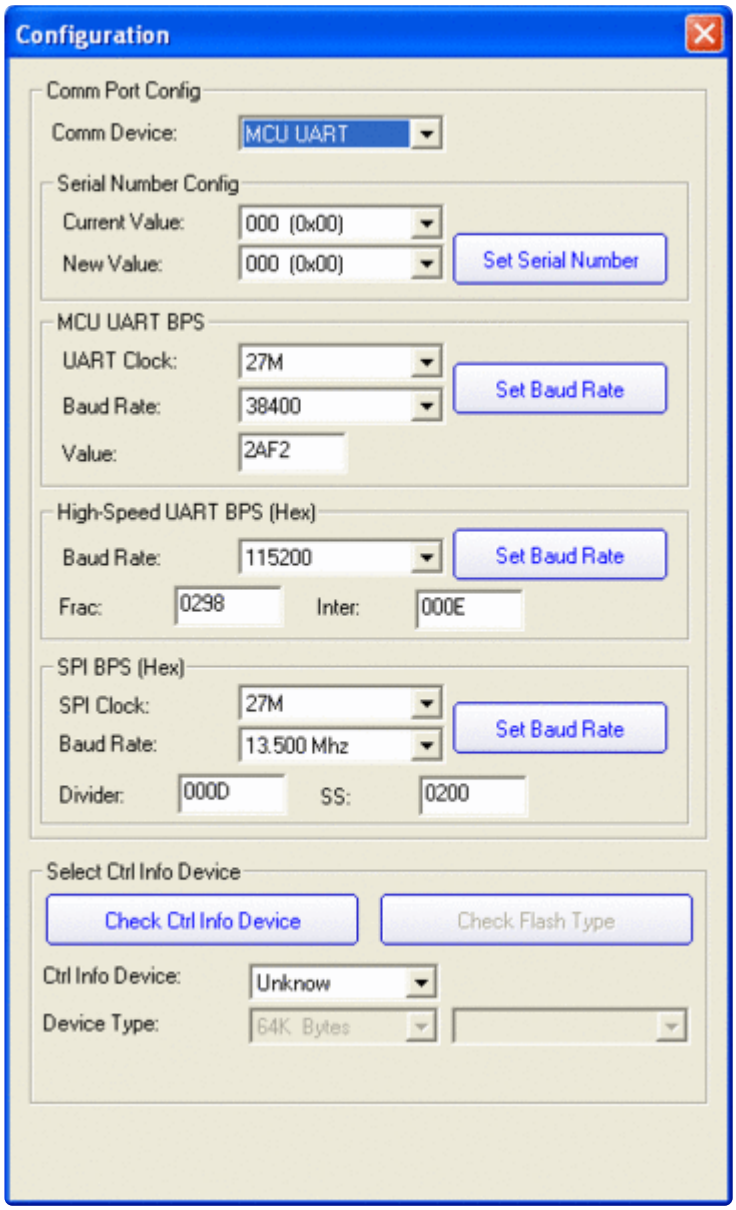
Next press Read (next to Sel File) to read the jpeg image off the camera



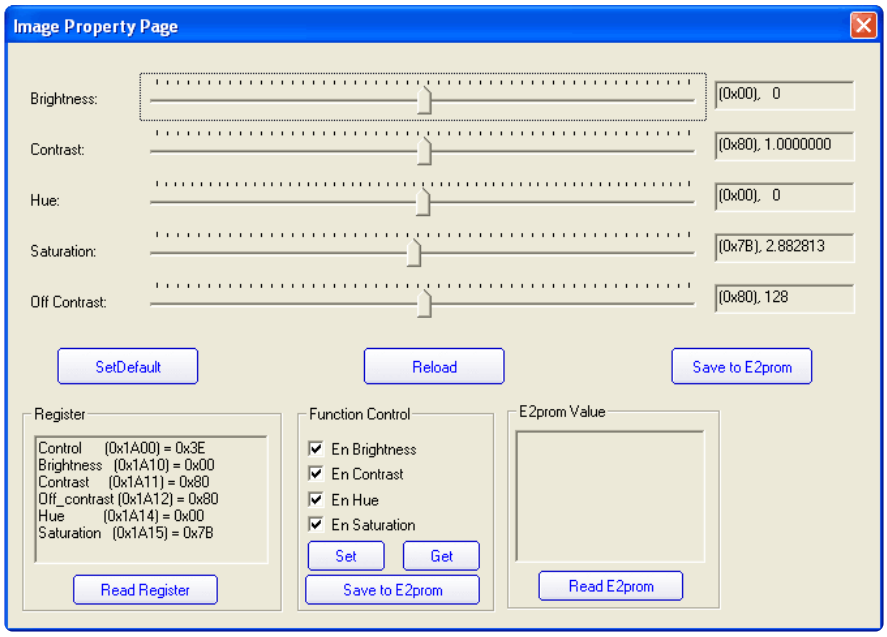
Thats it! You can now easily test reading camera images. To take another photo. Press Resume up at the top to have the video start up again. Then click Stop CFbuf when you want to snap another photo. Finally you can select the Compression Ratio which will improve or degrade the image quality but also change the image transfer time. There's no way to change the image size from this program (easily) but we can do it using the Arduino sketch so just try it out here to start.

You might notice there's a dropdown for changing the baud rate. By default the baudrate is 38400 baud.

Despite the software letting you change the baud rate this is a very flaky setting and even if it works, when you power up the camera again it will reset. Some experimenters have accidentally disabled their cameras by trying to change the baud rate. We do not suggest you mess with the baud rate settings. If you do, you may permanently disable your camera and we will not replace it!



The only other thing we suggest checking out is the Image Property button, which will let you adjust settings for the camera, we bumped up our saturation a bit to get better images. Dragging the sliders will make the video output change immediately so this is a handy place to get a TV connected up so you can check out how it works



There are many options for this software, here's what we think of the other buttons. Personally, we don't suggest going in to any of them unless you really need to.

- Config - see above
- Get Version - see above
- R/W Data - this is for writing raw data to the DSP chip processor. Don't do this unless you're sure you know what you're doing since it will mess with the camera's ability. Even we don't know what it would be good for
- Color Ctrl - this is for selecting Color or Black&White or Auto select (probably based on lighting conditions). You probably want to keep it at Auto
- Mirror Ctrl - we think this is so you can flip the display (if its bouncing off a mirror)
- Power Ctrl - this is for testing the power down mode, and it seems like you might be able to have it auto-power down when there's no motion.
- Timer Ctrl - there is an RTC built into the DSP which you can set, however there's no battery backup so if power is lost the RTC will be reset so we don't think its terribly useful
- AE Ctrl - this is for controlling the auto-contrast/brightness. By default its set to auto-select for indoor or outdoor use. Probably best to leave it as is
- Motion Ctrl - this is for the motion detection system. You can tweak the settings and also test it. We have an Arduino sketch for interacting with the motion detection system. By default it works pretty good but you can super tweak it out if you want to.
- OSD Config - The protocol sheet and this seem to imply you can do on-screen-display but after much time spent on it, we determined its not activated somewhere in the DSP. We've never seen a VC0706 camera that could do it. :(
- Image property - see above

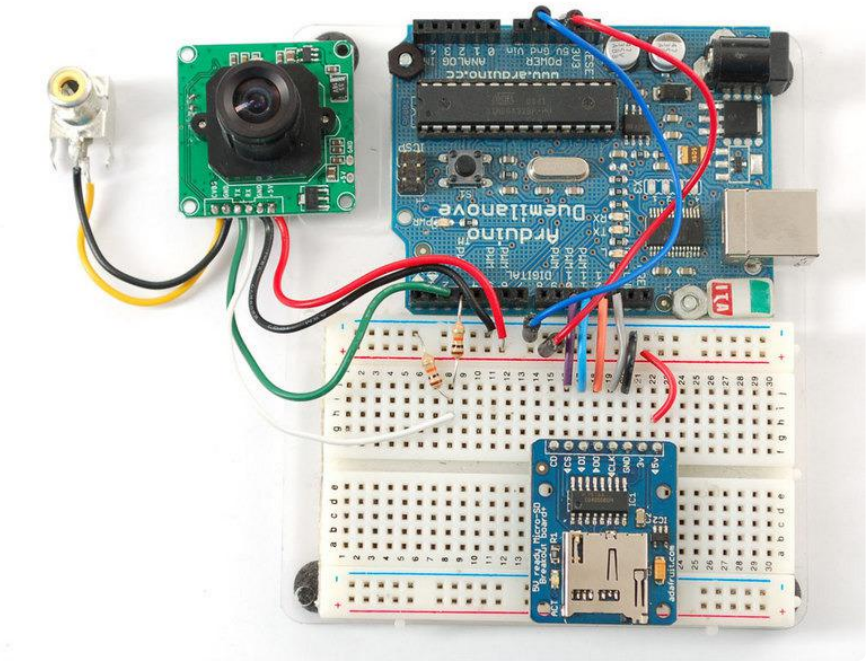
- Gamma - this is for more precise gamma control of the CMOS sensor. It seems to be preset to be OK but you can mess with this if you'd like
- SPI Flash - for reading/writing to the SPI storage? Not sure if its a good idea to mess with this
- Other Ctrl - for playing with the DAC? No idea what this is for.
- Up/Down Load - this is for reading and writing to the flash probably to upload new DSP code. We dont suggest messing with this
- System Reset - does a reset of the module. Press this if its not responding
- FBuff Ctrl - see above
- Zoom Ctrl - The module has built in 'Pan Tilt Zoom' ability BUT its for video only and wont affect photos snapped. You can play with the PTZ here, its pretty basic but could be useful for someone

---

Arduino Usage

Next up, we will wire the camera to our microcontroller (in this case an Arduino). This is pretty similar to the above except we will be using two digital pins and a software serial port to talk to the camera. To save images, you'll need some sort of external storage like our microSD breakout board (http://adafru.it/254).

Connect up the camera like this:



For the weatherproof camera, the white and green wires are swapped on some cameras! So please flip the white and green wires indicated if using the metal camera. Red should still be connected to +5 and Black to Ground

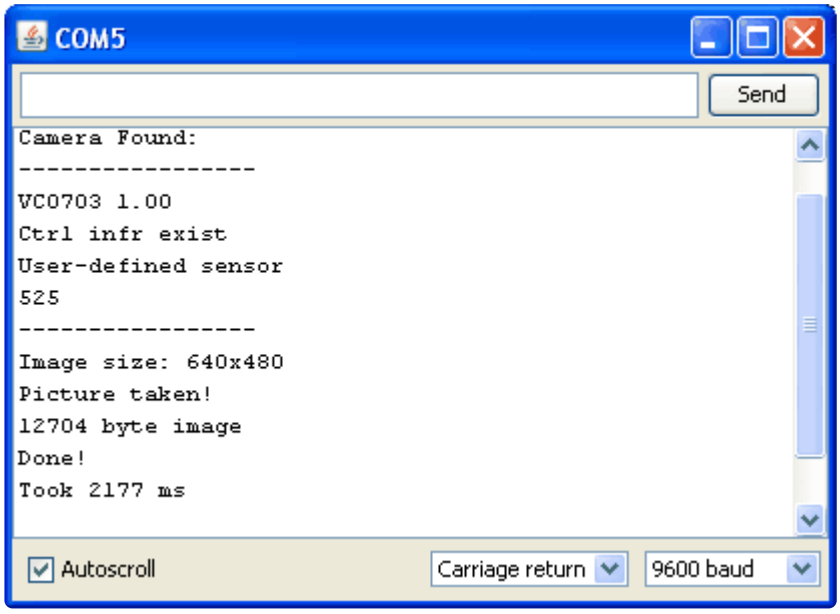
We suggest testing the microSD card first. Check out our microSD breakout board tutorial and verify that you can read from the card by listing the files. Once you have verified the microSD card wiring, you can come back here and install the VC0706 camera library.

Visit the Github repository here. () To download. click the DOWNLOADS button in the top right corner, rename the uncompressed folder Adafruit_VC0706. Check that the Adafruit_VC0706 folder contains Adafruit_VC0706.cpp and Adafruit_VC0706.h Place the Adafruit_VC0706 library folder your arduinosketchfolder/libraries/ folder. You may need to create the libraries subfolder if its your first library. Restart the IDE.

If you're using Arduino v23 or earlier, you'll also need to install the NewSoftSerial library. Download it by clicking this link () and install it as you did the Adafruit_VC0706 library. Arduino 1.0 has this built in now (called SoftwareSerial)

Taking a Snapshot

OK now you're finally ready to run the snapshot demo. Open up the Arduino IDE and select File-> Examples-> Adafruit_VC0706-> Snapshot sketch and upload it to the Arduino. Open up the serial monitor and you can see the sketch will take a 640x480 photo and save it to the microSD card. You can then pop the card into your computer to see the JPG file



There are a few things you can change once you get it working. One is changing the pins the camera uses. You can use any two digital pins, change this line:

```
// This is the camera pin connection. Connect the camera TX
// to pin 2, camera RX to pin 3
NewSoftSerial cameraconnection = NewSoftSerial(2, 3);
```

You can also change the snapshot image dimension to 160x120, 320x240 or 640x480 by changing these lines:

```
// Set the picture size - you can choose one of 640x480, 320x240 or 160x120
  // Remember that bigger pictures take longer to transmit!

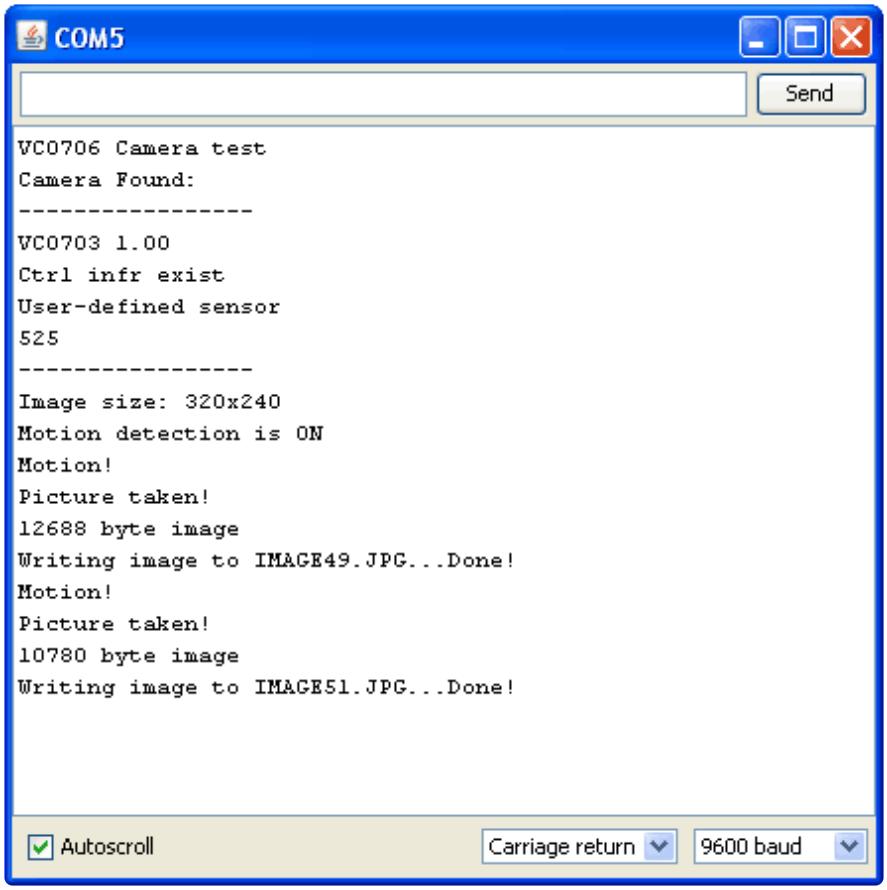
  cam.setImageSize(VC0706_640x480);        // biggest
  //cam.setImageSize(VC0706_320x240);        // medium
  //cam.setImageSize(VC0706_160x120);          // small
```

Simply uncomment the size you want, and comment out the others. Bigger pictures will take longer to snap, so you will want to think about how fast you need to grab data and save it to the disk

Detecting Motion

A neat thing that the camera has built in is motion detection. It will look for motion in the video stream and alert the microcontroller (by sending a serial data packet) when motion is detected. IN this way you can save a bit of cash and skip on having a PIR sensor (although a PIR sensor will be better at detecting warm mammalian things).

Load up the File-> Examples-> Adafruit_VC0706-> MotionDetect sketch and upload it to the Arduino. It will take a photo immediately because it just turned on. Then wait a few minutes and wave you hand in front of the camera, it will take another photo.



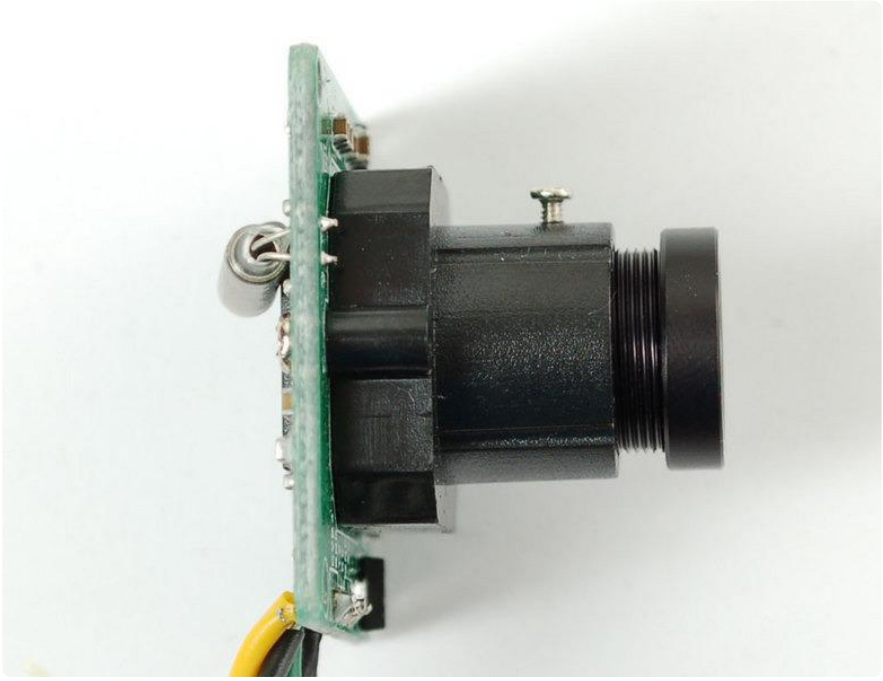
You can turn motion detection on or off by calling setMotionDetect()

```
//  Motion detection system can alert you when the camera 'sees' motion!
  cam.setMotionDetect(true);           // turn it on
  //cam.setMotionDetect(false);        // turn it off   (default)
```

You'll need to 'poll' the camera to ask it when motion is detected, by calling motionDetected()- it will return true if motion was recently detected, and false otherwise.

Adjusting the Manual Focus

One last thing, the camera modules use a manual focus system - there's no auto focus. This can be good or bad. The camera comes with a far depth of focus which is good for most stuff. If you want to change the focus, we strongly recommend plugging it into a video monitor as shown above so you can see exactly how the camera focus looks. You can then lock the focus with the set screw



The version in the weatherproof housing is a little tougher to adjust but it can be done by unscrewing the housing (it takes a few steps but its all easy to do) and then adjusting the focus before reassembly

---

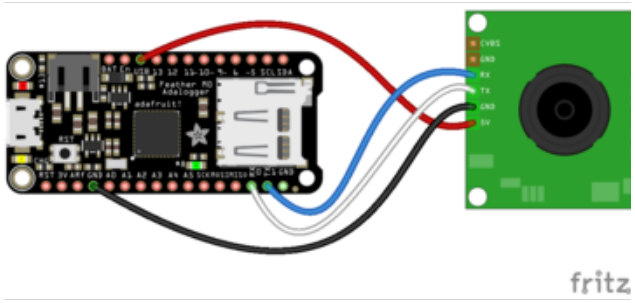
CircuitPython & Python Usage

In addition to taking pictures with the camera in Arduino, you can also use Python and CircuitPython to snap photos and save them to a SD card, computer or Raspberry Pi! The Adafruit CircuitPython VC0706 () library is your key to accessing the TTL camera and grabbing images over a serial connection.

You can use this camera with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library ().

CircuitPython Microcontroller Wiring

First you'll need to connect the TTL camera and a micro SD card holder to your CircuitPython board. The easiest and recommended option is to use a Feather M0 Adalogger board loaded with CircuitPython. This gives you a micro SD card holder that's pre-wired and ready to go, just connect the camera to the board. Here's an example of connecting the camera to a Feather M0 Adalogger:



Just like connecting the camera to an Arduino you need to connect these wires:

Camera 5V to board USB or 5V power (note this means you must have the board plugged into a USB / 5V power supply to properly power the camera).
Camera GND to board GND.
Camera RX to board TX.
Camera TX to board RX.
In addition, please make sure a micro SD card formatted with the FAT32 filesystem (highly recommended to use the official SD card formatter here () and not your operating system's formatter!) is inserted in the SD card holder.

Python Computer Wiring

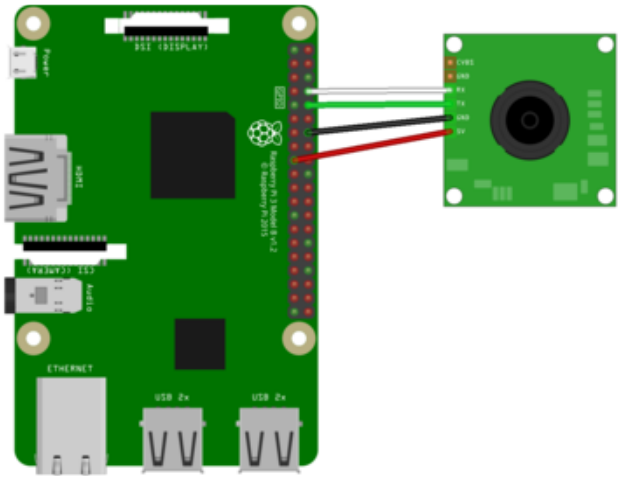
Since there's dozens of Linux computers/boards you can use, we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported ().

Here you have two options: An external USB-to-serial converter, or the built-in UART on the Pi's TX/RX pins. Here's an example of wiring up the USB-to-serial converter ():



Camera Vin to USB 5V or 3V (red wire on USB console cable)
Camera Ground to USB Ground (black wire)
Camera RX (white wire) to USB TX (green wire)
Camera TX (green wire) to USB RX (white wire)

Here's an example using the Pi's built-in UART:



Camera 5V (black wire) to PI 3V or 5V
Camera GND (black wire) to Pi Ground
Camera RX (white wire) to Pi TX
Camera TX (green wire) to Pi RX

If you want to use the built-in UART, you'll need to disable the serial console and enable the serial port hardware in raspi-config. See the UART/Serial section of the CircuitPython on Raspberry Pi guide () for detailed instructions on how to do this.

All single board computers are a bit different. Some expose the serial port/UART, others have it soft connected to the console, while others do not allow UART use by the user. Please see your board documentation to see what using your board UART may entail.

CircuitPython Installation of VC0706

As mentioned, you'll also need to install the Adafruit CircuitPython VC0706 () library on your CircuitPython board. In addition, the Adafruit CircuitPython SD () library is used to read and write data to the SD card.

First make sure you are running the latest version of Adafruit CircuitPython () for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (). The Welcome to CircuitPython guide has a great page on how to install the library bundle ().

If your board supports sdcardio (), then this is the preferred method to do things. sdcardio is a built-in module on boards that support it, so you don't have to copy it over.

After downloading the bundle, copy the necessary libraries from the bundle:

- adafruit_vc0706.mpy
- adafruit_sdcard.mpy (if your board doesn't support sdcardio)
- adafruit_bus_device

Before continuing, make sure your board's lib folder has the adafruit_vc0706.mpy, adafruit_sd.mpy, and adafruit_bus_device files and folders copied over.

Next connect to the board's serial REPL () so you are at the CircuitPython >>> prompt.

Python Installation of VC0706 Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling UART on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready ()!

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-vc0706`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

Microcontroller CircuitPython Usage (not for Linux/SBC)

To demonstrate the usage of the camera, let's look at an example that will capture an image and save it to the micro SD card as a jpeg file. Load up the example below and save it as code.py on your CIRCUITPY drive, then open the serial REPL to see the output:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""VC0706 image capture to SD card demo.
You must wire up the VC0706 to the board's serial port, and a SD card holder
to the board's SPI bus.  Use the Feather M0 Adalogger as it includes a SD
card holder pre-wired to the board--this sketch is setup to use the Adalogger!
In addition you MUST also install the following dependent SD card library:
https://github.com/adafruit/Adafruit_CircuitPython_SD
See the guide here for more details on using SD cards with CircuitPython:
https://learn.adafruit.com/micropython-hardware-sd-cards"""

import time

import board
import busio

# import digitalio # Uncomment if your board doesn't support sdcardio
import storage

# import adafruit_sdcard # Uncomment if your board doesn't support sdcardio
import sdcardio  # Comment out if your board doesn't support sdcardio
import adafruit_vc0706


# Configuration:
SD_CS_PIN = board.D10  # CS for SD card (SD_CS is for Feather Adalogger)
IMAGE_FILE = "/sd/image.jpg"  # Full path to file name to save captured image.
# Will overwrite!

# Setup SPI bus (hardware SPI).
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Setup SD card and mount it in the filesystem.
# Uncomment if your board doesn't support sdcardio
# sd_cs = digitalio.DigitalInOut(SD_CS_PIN)
# sdcard = adafruit_sdcard.SDCard(spi, sd_cs)
sdcard = sdcardio.SDCard(
    spi, SD_CS_PIN
)  # Comment out if your board doesn't support sdcardio

vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")

# Create a serial connection for the VC0706 connection, speed is auto-detected.
uart = busio.UART(board.TX, board.RX)
# Setup VC0706 camera
vc0706 = adafruit_vc0706.VC0706(uart)

# Print the version string from the camera.
print("VC0706 version:")
print(vc0706.version)
```

```
# Set the baud rate to 115200 for fastest transfer (its the max speed)
vc0706.baudrate = 115200

# Set the image size.
vc0706.image_size = adafruit_vc0706.IMAGE_SIZE_640x480  # Or set IMAGE_SIZE_320x240
or
# IMAGE_SIZE_160x120
# Note you can also read the property and compare against those values to
# see the current size:
size = vc0706.image_size
if size == adafruit_vc0706.IMAGE_SIZE_640x480:
    print("Using 640x480 size image.")
elif size == adafruit_vc0706.IMAGE_SIZE_320x240:
    print("Using 320x240 size image.")
elif size == adafruit_vc0706.IMAGE_SIZE_160x120:
    print("Using 160x120 size image.")

# Take a picture.
print("Taking a picture in 3 seconds...")
time.sleep(3)
print("SNAP!")
if not vc0706.take_picture():
    raise RuntimeError("Failed to take picture!")

# Print size of picture in bytes.
frame_length = vc0706.frame_length
print("Picture size (bytes): {}".format(frame_length))

# Open a file for writing (overwriting it if necessary).
# This will write 50 bytes at a time using a small buffer.
# You MUST keep the buffer size under 100!
print("Writing image: {}".format(IMAGE_FILE), end="")
stamp = time.monotonic()
# pylint: disable=invalid-name
with open(IMAGE_FILE, "wb") as outfile:
    wcount = 0
    while frame_length > 0:
        # Compute how much data is left to read as the lesser of remaining bytes
        # or the copy buffer size (32 bytes at a time).  Buffer size MUST be
        # a multiple of 4 and under 100.  Stick with 32!
        to_read = min(frame_length, 32)
        copy_buffer = bytearray(to_read)
        # Read picture data into the copy buffer.
        if vc0706.read_picture_into(copy_buffer) == 0:
            raise RuntimeError("Failed to read picture frame data!")
        # Write the data to SD card file and decrement remaining bytes.
        outfile.write(copy_buffer)
        frame_length -= 32
        # Print a dot every 2k bytes to show progress.
        wcount += 1
        if wcount >= 64:
            print(".", end="")
            wcount = 0
# pylint: enable=invalid-name
print()
print("Finished in %0.1f seconds!" % (time.monotonic() - stamp))
# Turn the camera back into video mode.
vc0706.resume_video()
```

You should see output like the following as the program prints information about the camera and saves an image to the micro SD card:

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
main.py output:
VC0706 version:
VC0703 1.00
Ctrl infr exist
OV7720
525
Init end

Using 640x480 size image.
Taking a picture in 3 seconds...
SNAP!
Picture size (bytes): 51716
Writing image: /sd/image.jpg.....
```

Be aware saving the image to the card takes some time, as the data is transferred over both a serial connection from the camera and the SPI connection to the micro SD card. A full image capture at 640x480 pixels takes about 30 seconds, but might take longer depending on your board and micro SD card speed.

Once the image capture finishes, you'll see a message printed:

```
Picture size (bytes): 51716
Writing image: /sd/image.jpg.........................
Finished!
```

Exit the REPL and power down the board, then remove the SD card and connect it to your computer. You should see an image.jpg file saved on it, and inside will be a picture captured from the camera:



Woo hoo, that's all there is to the basics of capturing an image with the serial TTL camera and CircuitPython! Let's look at the code in a tiny bit more detail to understand the usage.

First the example needs to setup the SD card and mount it on the filesystem. This is all boilerplate code from the CircuitPython SD card guide () (highly recommended to read it too!):

```
# Configuration:
SD_CS_PIN = board.D10  # CS for SD card (SD_CS is for Feather Adalogger)
IMAGE_FILE = "/sd/image.jpg"  # Full path to file name to save captured image.
# Will overwrite!

# Setup SPI bus (hardware SPI).
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Setup SD card and mount it in the filesystem.
# Uncomment if your board doesn't support sdcardio
# sd_cs = digitalio.DigitalInOut(SD_CS_PIN)
# sdcard = adafruit_sdcard.SDCard(spi, sd_cs)
sdcard = sdcardio.SDCard(
    spi, SD_CS_PIN
)  # Comment out if your board doesn't support sdcardio

vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")
```

Now the VC0706 module is setup and an instance of the VC0706 class is created. Notice we need to create a UART device on whatever pins have hardware support and then this is passed to the camera creator.

```
# Create a serial connection for the VC0706 connection, speed is auto-detected.
uart = busio.UART(board.TX, board.RX, timeout=250)
# Setup VC0706 camera
vc0706 = adafruit_vc0706.VC0706(uart)
```

Once the VC0706 instance is created you can read some interesting properties, like the version string:

```
# Print the version string from the camera.
print('VC0706 version:')
print(vc0706.version)
```

Or even set and get the size of the image (640x480, 320x240, 160x120):

```
# Set the image size.
vc0706.image_size = adafruit_vc0706.IMAGE_SIZE_640x480 # Or set IMAGE_SIZE_320x240
or
                                                       # IMAGE_SIZE_160x120
# Note you can also read the property and compare against those values to
# see the current size:
size = vc0706.image_size
if size == adafruit_vc0706.IMAGE_SIZE_640x480:
    print('Using 640x480 size image.')
elif size == adafruit_vc0706.IMAGE_SIZE_320x240:
    print('Using 320x240 size image.')
elif size == adafruit_vc0706.IMAGE_SIZE_160x120:
print('Using 160x120 size image.')
```

Now the real fun, you can capture an image! This works by first telling the camera to 'freeze' the current image frame in memory with the `take_picture` function. Then you need to make a loop that calls the `read_picture_into` function repeatedly to grab buffers of image data from the camera. Once you have image data it's up to you to do something with it, like write it to a SD card file (although you don't have to do that, you could send it to a web service or do other fun thing!).

The code in this example will capture an image and then save it to a file on the SD card:

```
# Take a picture.
print('Taking a picture in 3 seconds...')
time.sleep(3)
print('SNAP!')
if not vc0706.take_picture():
    raise RuntimeError('Failed to take picture!')

# Print size of picture in bytes.
frame_length = vc0706.frame_length
print('Picture size (bytes): {}'.format(frame_length))

# Open a file for writing (overwriting it if necessary).
# This will write 50 bytes at a time using a small buffer.
# You MUST keep the buffer size under 100!
print('Writing image: {}'.format(IMAGE_FILE), end='')
with open(IMAGE_FILE, 'wb') as outfile:
    wcount = 0
    while frame_length &gt; 0:
        # Compute how much data is left to read as the lesser of remaining bytes
        # or the copy buffer size (32 bytes at a time).  Buffer size MUST be
        # a multiple of 4 and under 100.  Stick with 32!
        to_read = min(frame_length, 32)
        copy_buffer = bytearray(to_read)
        # Read picture data into the copy buffer.
        if vc0706.read_picture_into(copy_buffer) == 0:
            raise RuntimeError('Failed to read picture frame data!')
        # Write the data to SD card file and decrement remaining bytes.
        outfile.write(copy_buffer)
        frame_length -= 32
        # Print a dot every 2k bytes to show progress.
        wcount += 1
        if wcount &gt;= 64:
            print('.', end='')
            wcount = 0
```

One thing to be aware of is that the size of the buffer passed to `read_picture_into` must be a multiple of 4. This is an requirement of the camera hardware itself. In addition, it must be below 100 to fit within an internal buffer. Stick with using a value of 32 like the example here shows!

That's all there is to capturing and saving an image to an SD card using CircuitPython!

Saving Images to CircuitPython Internal Filesystem

Instead of using the SD card to store images it's also possible with CircuitPython or Python to save images to the internal filesystem where your code and other data files live. This is possible with a few caveats, in particular once you enable writing to the internal storage you can't set or change your code over the USB drive connection to your computer. This means you probably want to get your program working first on SD storage or ignoring the file save, and then switch to using internal storage when you know your code is working and ready to write files.

Also be aware internal storage is quite limited on some boards. The non-express boards only have ~64kb or space and a single 640x480 JPEG image from the camera can occupy 50 kilobytes of more of space alone! You likely only want to save images to the internal storage for Express boards that have 2 megabytes of space, however even on those boards take care to not store too many images as they will quickly add up

Activate Internal storage on Microcontrollers

This step is not used on Linux / Single Board Computers

To get started first follow the steps on the CircuitPython Storage page of the CircuitPython Essentials guide () to enable writing to internal storage. In particular edit the boot.py on your CIRCUITPY drive (creating it if it doesn't exist) and add these lines:

```
import digitalio
import board
import storage

switch = digitalio.DigitalInOut(board.D5)
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

# If the D5 is connected to ground with a wire
# you can edit files over the USB drive again.
storage.remount("/", not switch.value)
```

Remember once you remount("/") you cannot edit code over the USB drive anymore! That means you can't edit boot.py which is a bit of a conundrum. So we configure the boot.py to selectively mount the internal filesystem as writable based on a switch or even just alligator clip connected to ground. Like the CPU temperature guide shows ( ). In this example we're using D5 but select any available pin.

This code will look at the D5 digital input when the board starts up and if it's connected to ground (use an alligator clip or wire, for example, to connect from D5 to board ground) it will disable internal filesystem writes and allow you to edit code over the USB drive as normal. Remove the alligator clip, reset the board, and the boot.py will switch to mounting the internal filesystem as writable so you can log images to it again (but not write any code!).

Remember when you enable USB drive writes (by connecting D5 to ground at startup) you cannot write files to the internal filesystem and any code in your code.py that attempts to do so (like the example below) will fail. Keep this in mind as you edit code, once you modify code you need to remove the alligator clip, reset the board to re-enable internal filesystem writes, and then watch the output of your program.

If you ever get stuck, you can follow the steps mentioned in https://learn.adafruit.com/cpu-temperature-logging-with-circuit-python/writing-to-the-filesystem to remove boot.py from the REPL if you need to go back and edit code!

Example Code for saving to internal file system (CircuitPython or Linux / SBC)

Now we can use a slightly modified version of the example that will save to the internal filesystem instead of a SD card. The code is exactly the same as for SD cards except instead of mounting the SD card and opening a file there, we open a file on the internal storage. The exact same VC0706 functions and control loop are used because Python's read and write functions don't care if they're writing to a SD card or internal storage--it's all the same to Python!

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""VC0706 image capture to local storage.
You must wire up the VC0706 to a USB or hardware serial port.
Primarily for use with Linux/Raspberry Pi but also can work with Mac/Windows"""

import time
import busio
import board
import adafruit_vc0706

# Set this to the full path to the file name to save the captured image. WILL
OVERWRITE!
# CircuitPython internal filesystem configuration:
IMAGE_FILE = "/image.jpg"
# USB to serial adapter configuration:
# IMAGE_FILE = 'image.jpg'  # Full path to file name to save captured image. Will
overwrite!
# Raspberry Pi configuration:
# IMAGE_FILE = '/home/pi/image.jpg'  # Full path to file name to save image. Will
overwrite!
```

```
# Create a serial connection for the VC0706 connection.
uart = busio.UART(board.TX, board.RX, baudrate=115200, timeout=0.25)
# Update the serial port name to match the serial connection for the camera!
# For use with USB to serial adapter:
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=115200, timeout=0.25)
# For use with Raspberry Pi:
# import serial
# uart = serial.Serial("/dev/ttyS0", baudrate=115200, timeout=0.25)

# Setup VC0706 camera
vc0706 = adafruit_vc0706.VC0706(uart)

# Print the version string from the camera.
print("VC0706 version:")
print(vc0706.version)

# Set the image size.
vc0706.image_size = adafruit_vc0706.IMAGE_SIZE_640x480
# Or set IMAGE_SIZE_320x240 or IMAGE_SIZE_160x120

# Note you can also read the property and compare against those values to
# see the current size:
size = vc0706.image_size
if size == adafruit_vc0706.IMAGE_SIZE_640x480:
    print("Using 640x480 size image.")
elif size == adafruit_vc0706.IMAGE_SIZE_320x240:
    print("Using 320x240 size image.")
elif size == adafruit_vc0706.IMAGE_SIZE_160x120:
    print("Using 160x120 size image.")

# Take a picture.
print("Taking a picture in 3 seconds...")
time.sleep(3)
print("SNAP!")
if not vc0706.take_picture():
    raise RuntimeError("Failed to take picture!")

# Print size of picture in bytes.
frame_length = vc0706.frame_length
print("Picture size (bytes): {}".format(frame_length))

# Open a file for writing (overwriting it if necessary).
# This will write 50 bytes at a time using a small buffer.
# You MUST keep the buffer size under 100!
print("Writing image: {}".format(IMAGE_FILE), end="", flush=True)
stamp = time.monotonic()
# Pylint doesn't like the wcount variable being lowercase, but uppercase makes less
sense
# pylint: disable=invalid-name
with open(IMAGE_FILE, "wb") as outfile:
    wcount = 0
    while frame_length > 0:
        t = time.monotonic()
        # Compute how much data is left to read as the lesser of remaining bytes
        # or the copy buffer size (32 bytes at a time).  Buffer size MUST be
        # a multiple of 4 and under 100.  Stick with 32!
        to_read = min(frame_length, 32)
        copy_buffer = bytearray(to_read)
        # Read picture data into the copy buffer.
        if vc0706.read_picture_into(copy_buffer) == 0:
            raise RuntimeError("Failed to read picture frame data!")
        # Write the data to SD card file and decrement remaining bytes.
        outfile.write(copy_buffer)
        frame_length -= 32
        # Print a dot every 2k bytes to show progress.
        wcount += 1
        if wcount >= 64:
```

```
            print(".", end="", flush=True)
            wcount = 0
print()
# pylint: enable=invalid-name
print("Finished in %0.1f seconds!" % (time.monotonic() - stamp))
# Turn the camera back into video mode.
vc0706.resume_video()
```

Saving Images to Computer or Raspberry Pi / Linux

Saving images to a Raspberry Pi or other Linux computer is very similar to the CircuitPython internal filesystem. You simply need to comment out a line and uncomment two more depending on what set up you're using.

To comment out a line of code, put a '#' before the line of code. To uncomment a line of code, delete the '# ' (including the space!) before the line of code.

Regardless of which set up you're using, you'll need to comment out the following line:

```
uart = busio.UART(board.TX, board.RX, baudrate=115200, timeout=0.25)
```

USB to Serial Converter

If using a USB to serial converter, uncomment the following lines:

```
# import serial
```

```
# uart = serial.Serial("/dev/ttyUSB0", baudrate=115200, timeout=0.25)
```

Raspberry Pi / Linux

If using a Raspberry Pi, uncomment the following lines (if you're using a different single board computer, you may need to update the serial port!):

```
# import serial
```

```
# uart = serial.Serial("/dev/ttyS0", baudrate=115200, timeout=0.25)
```

The rest of the code works the same way. Smile!

Python Docs

Python Docs ()

F.A.Q.

Can I change the baud rate on this Camera?

You might notice there seems to be a command for changing the baud rate. By default the baudrate is 38400 baud.

Despite the software letting you change the baud rate this is a very flaky setting and even if it works, when you power up the camera again it will reset. Some experimenters have accidentally disabled their cameras by trying to change the baud rate. We do not suggest you mess with the baud rate settings. If you do, you may permanently disable your camera and we will not replace it!

How fast can I get pictures from the camera?

This is a pretty slow UART camera, it can take up to 30 seconds to transfer an image! It is meant for snapshots or time-lapse type photography, not for any kind of real-time analysis

Why is the color washed out? It looks like a monochrome image.

Because it was designed for surveillance, the sensitivity of the camera extends into the infrared range. This means that objects that reflect or emit infrared rays will appear lighter than the do to the human eye. In some cases the image will appear washed out and almost monochromatic.

A more natural rendering can be achieved using an IR blocking filter such as a B+W 486 (). (Thanks to forum member azhilyakov for the comparison photos!)



Adafruit VC0706
No filter

Adafruit VC0706
with UV/IR cut-off filter

Actual

Buy a TTL Serial Camera

Buy a TTL Serial Camera (http://adafru.it/397)

Downloads

VC0706 Comm Tool - Windows control software () (works in Parallels in MacOSX. We do not have source code for this tool in order to directly port it to Mac/Linux)

Adafruit VC0706 Arduino library Github repository ()

NewSoftSerial library download ()

Unsupported Alternate libraries

https://github.com/oskarirauta/Adafruit-VC0706-Serial-Camera-Library () is a version for the Maple - we didn't write this code and don't support it but it might be handy for Maple users!