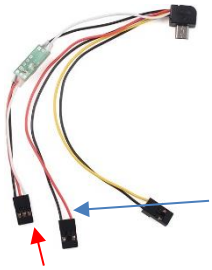
Controlling a RunCam2 Camera Directly from an Arduino, Version 1.4

By R. G. Sparber

For an overview, please view this video:
https://www.youtube.com/watch?v=ZTLdzBgyD5I

The challenge was to have a RunCam2 camera[2] fully controlled by an Arduino based Pro Micro[3]. I also needed to have the camera power up when the Arduino received power.



One piece of the puzzle was a RunCam2 Remote cable[4] which takes the slow toggling of a digital line (red arrow) and somehow gets the RunCam2 to switch between video and still. It also will start and stop video or take a single still picture. External power is supplied to the RunCam2 by the connector with the red and black wires. RF output[5] is on the yellow and black wires.

This would work except that it depends on someone first pushing the power-on/shutter button on the RunCam2. Furthermore, only two vendors claimed to have this cable in stock.

I bought the cable from the first vendor only to discover that they were crooks. Eventually, PayPal got my money back. Then I bought from the second vendor. After waiting almost a month, the cable arrived. This motivated me to ***not*** use this cable!

---

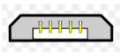
[1] This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
[2] https://shop.runcam.com/runcam2/
[3] https://www.sparkfun.com/products/12640
[4] https://shop.runcam.com/runcam2-remote-cable/
[5] This signal is present regardless of picture taking state. I had hoped that it could be used to confirm when we were in the video mode.

I tried out the control cable and it worked exactly as advertised[6]. Using my digital memory oscilloscope, I was able to record the signal generated by the control cable that fed into the RunCam2.

When the camera was in the video/standby mode, the cable's output would cause the camera to change to record mode. If in still-picture/standby, it would take a picture. The 'scope showed these two pulses. Each pulse is 80 ms wide and there is 80 ms between them. The voltage is 0 at the bottom of each pulse and 3.6 volts at the top.

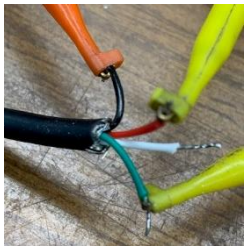
When the camera was in video/standby, the cable's output would toggle to picture/standby. The 'scope showed a single pulse as the stimulus. This pulse is also 80 ms wide and 3.6 volts high.

I found that if I left 100 ms between the end of a double pulse and the start of a single pulse, the camera did not always respond correctly. Going to a 2-second delay solved that problem. This delay value could probably be reduced but waiting this long is not a problem in the current application.



Next, I needed to figure out which wires went with which functions. The control cable uses a USB Micro-B connector into the camera.

I cut off the USB-A connector at the other end of the cable, removed the outer ground shield, and stripped off about 1/8 of an inch from each of the wires.



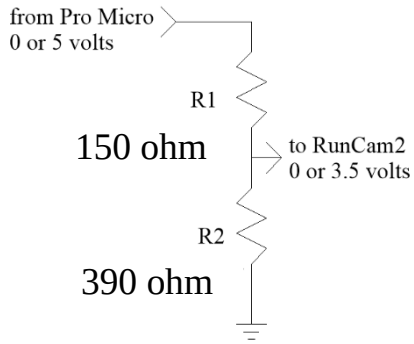
The **black** wire connects to ground.
The **red** wire connects to power. The camera can run on 5 to 17 volts but I chose to use 5V just in case something else got plugged into this cable.
The **green** wire is our control line.
The white wire is not used.

A word of warning. The camera dissipates around 2.5 watts when idle and 3.25 watts when recording. Do not restrict air flow or it will overheat and shut down.

[6] https://www.youtube.com/watch?v=Uspf9eORuPA

Now that I know the “secret sauce” of the control cable, it was easy to get the Arduino based software to generate the pulses. More on this later.

If I was using a 3.3V Pro Micro, its output would be directly compatible with the RunCam2’s green control wire. I’m using the 5V Pro Micro so its output must be reduced from 0/5V down to about 0/3.6V.



The solution involved configuring two resistors as a voltage divider. A Pro Micro digital output connects to the top of R1 and the RunCam2’s green control line connects at the junction of R1 and R2. Both resistors are +/- 5% and have a power rating of 0.1 watts.

I did find one surprise. Originally I used a 1.5K and 3.9K resistor and measured 1.38V when the Pro Micro output 0V. This means that the camera was sourcing about 1.7 mA. This is a logic ½ which would be seen by the camera’s logic as random swings between logic 0 and 1. By going with the values shown here, the voltage rise was measured at 0.16V which is a solid logic 0.

I had one last problem to solve. How do I simulate pushing the power-on button? The obvious solution is to open up the camera and solder wires to the switch contacts. I ***really*** dislike doing this kind of microsurgery because it turns the off-the-shelf RunCam2 into what I call a “moon rock”. If the camera dies or we need another camera, this microsurgery must be repeated.

Even worse would be an electro/mechanical means of pushing the button. It would be complex and possibly larger than the camera. The thought of building such a device drove me back to thinking about that microsurgery.

But then a crazy question came to mind: What happens if I push down on the power-on/shutter button and don’t let up? Bingo! The camera powers up when power is applied at the cable and the control pulses still work.

Instead of an electromechanical device to push and release the button, I just needed a simple clamp to hold the button down.

The clamp was made from a piece of aluminum extruded channel. The lower clamp face was lined with a disk of stick on rubber. The upper clamp face was drilled and tapped for a 10-24 set screw. That notch was needed to clear the USB connector.

A second disk was stuck onto the button's face to protect it from the twisting force of the set screw.

The disk on the button's face was cut using a ¼ inch diameter paper punch. The lower clamp face's ½ inch diameter disk was precut on the sheet.

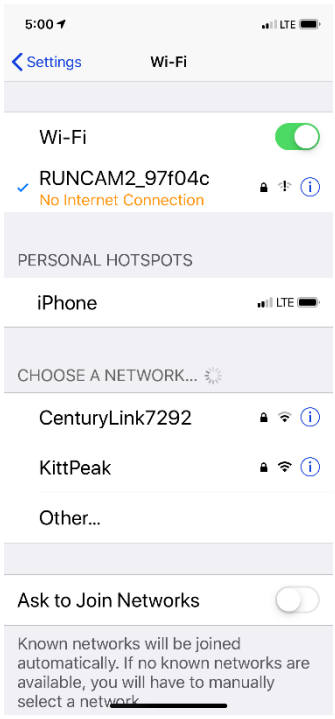
The clamp is placed around the camera and the set screw turned until you can hear the button click. Overtightening this screw can damage the camera.

The remaining steps are needed to configure the RunCam2 so it treats the USB input as a remote cable and not as a connection to a PC. You will need a Smartphone that can run the RunCam app. The following steps were performed on my iPhone.

Connect the camera to a power supply and not to a computer. You can only turn on the camera's WiFi if it is not connected to a computer.

The first step was to power up the RunCam2 by pushing the power-on/shutter button for about 3 seconds. Then I released the button.
Then I briefly pushed the small Wi-Fi button next to it.

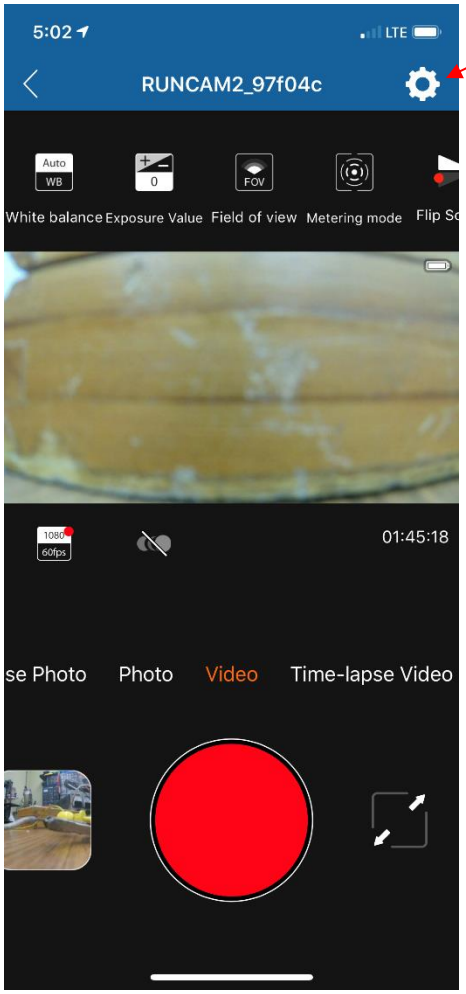
If all goes well, this blue bar will start to flash. If it doesn't, quickly push and release the power-on/shutter button again and then push and release the Wi-Fi button. Hopefully, it will now be flashing. I often ran into this annoying intermittent behavior with the camera and app but, once configured, all was solid.



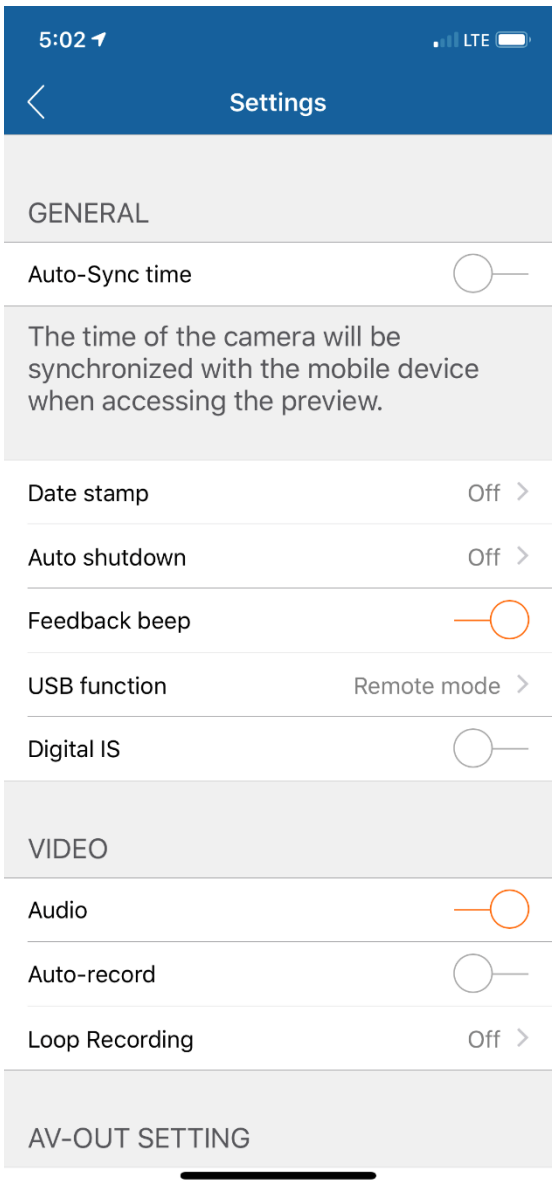
With the blue bar flashing, go to iPhone's Settings and select Wi-Fi. The RunCam2 will be broadcasting a host signal just like your router. Select RunCam2 and wait for the password prompt. The default password is 1234567890. Hopefully, that works for you.

With the iPhone now using the RunCam2 as its source of Wi-Fi, you can start the RunCam2 app. If it was already running, fully close it before proceeding. Otherwise, the app will likely just hang as it waits for Wi-Fi.

You should see something like this when the app starts up.



Touch this gear icon to go to the app's Settings page.

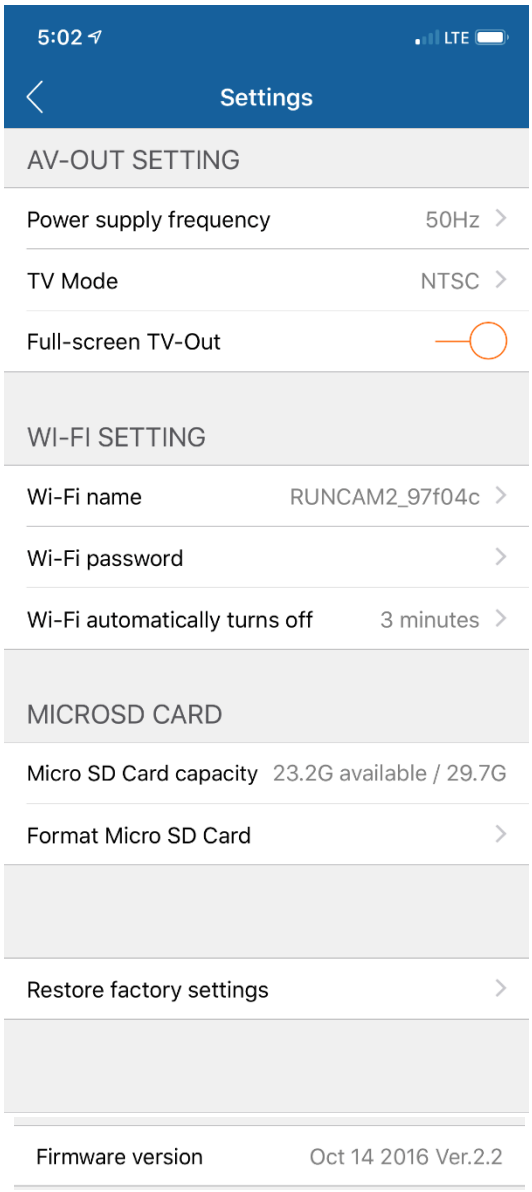


If the USB function is set to PC, touch the ">" icon and select Remote mode.

Auto-record must be off if you want to control the camera via the Arduino[7].

You may see other options that would be nice to either enable or disable. All can be changed but only some will remain in the new state after power down. I gave up fighting with some of these options but have not had any problems.

[7] If you just want to record when power is applied to the camera, turn on Auto-record but then do not connect the control line to the Arduino.



Scrolling down, you will see more options.

The only option that I found useful and consistently operational was the ability to format the micro SD card. This essentially erases all data.

Turn off the camera, re-establish your old Wi-Fi connection, and close the app.

You should now be able to control the RunCam2 camera with an Arduino. These settings should persist even when power is removed from the camera. Just to be safe, I suggest having a Smartphone and this document handy just in case the camera needs a Factory Restore.

Warning: I found that the camera stops responding correctly to the remote interface when the Micro SD card is full. At power-up, it acts like it is recording video. If in doubt, open the screen shown above and see how much memory is available.

The final topic is how to generate the control pulses in the Pro Micro.

Here is my test code.

At the end of setup() I have:

```
delay (10000); //give the RunCam2 10 seconds to stabilize
digitalWrite(RunCamControl, LOW); //insure we start out with the
control line low
```

After loop() I have my functions defined:

```
void CameraTest(){ //10 seconds of video, then takes one still, repeat
//at power up, camera is in video standby mode
      PushShutter(); //start video recording
      delay (10000); //records 10 seconds of video
      PushShutter(); //stop video recording
      ChangeCameraMode(); //change to still mode
      PushShutter(); //take one picture
      ChangeCameraMode(); //go from still to video record. 10 second
delay at start of CameraTest() is the delay
}

void PushShutter(){//two pulses pushes the shutter to either take one
picture or start video recording
      OnePulse();
      OnePulse();
      delay (2000); //give time for camera to process command
}

void ChangeCameraMode(){//single pulse toggles between still and video
      OnePulse();
      delay (2000); //give time for camera to process command
}

void OnePulse(){ // |--|__  assumes output starts out low
      digitalWrite(RunCamControl, HIGH); //80 ms wide pulse
      delay (80);
      digitalWrite(RunCamControl, LOW);
      delay (80);//80 ms of low
}
```

`RunCamControl` is the logical pin number used as my output. In setup() is:

```
const int RunCamControl = 7; //this is physical pin number 10.
```

Call these functions from within loop().

Note that the controls run blind. We know that at power-up, the camera will be in video standby mode. Sending a `PushShutter()` command will start video recording. Sending it again with stop recording. But if the camera is in any other state, unexpected behavior will result. I tried looking at the RF Output signal to see if it would be active only when in video mode. Sadly, it always outputs RF regardless of camera state. So as of now, I have no way to verify the camera's state.

I welcome your comments and questions.

If you wish to be contacted each time I publish an article, email me with just "Subscribe" in the subject line. If you are on this list and have had enough, email me "Unsubscribe" in the subject line.

Rick Sparber
Rgsparber.ha@gmail.com
Rick.Sparber.org