Arduino Camera Controller

Written by: Steven Stuber

What is it?

The Arduino Camera Controller is a means to control the Sony Handycam HD camcorders to take pictures and perform functions such as record, pause, turn off and turn on. The controller is a command line application that has been compiled and tested on Windows, Linux and Mac. The application controls a box containing an Arduino and several 3.5mm audio jacks. These audio jacks are connected by a cable to the Sony Handycam and carry the LANC Sony camera controller protocol signal. This device can handle cable lengths of over 180 feet without signal degradation. Further statistics on the functionality or limitations of the device can be found in the Appendix.

How to use it

A brief overview of the most common uses for the device will be discussed here. Several other specific instructions and notes will be provided in the Appendix.

Plug the USB cable into the Arduino box and the other end into the computer.
If using several cameras, it is recommended to plug in the external power supply.
Plug the 3.5mm cables into the top and have the other ends converted to a 2.5mm plug and plugged into the Sony Handycams.

Now that the cabling is set up, ensure that the cameras are in the On position to allow for proper controlling.

Turn on the computer attached to the Arduino box and open a command line in which the ArduinoCameraController or CameraController executable is accessible.

Run the application using the command line arguments " –d find". This will return the COM port (in Windows) or tty device (in Unix) to which the Arduino is currently connected.

Now run the application again with " –d ____" where ____ represents the value found in step 6; for further functionality, we can add more commands or switches.
Steps 6 and 7 are optional but will significantly decrease the delay for the first command in each application call.

Now, to specify a meta-command or another switch, we simply look at the list of available commands and decide how we would like to proceed.

Ex1. *"ArduinoCameraController.exe –d COM4 report"* would be a simple command to display the currently turned-on devices attached to the Arduino box (which would be on COM4 in this case) — mostly used to double-check that all the cameras are in working order before issuing record or snap commands.

Ex2. *"./CameraController –d /dev/ttyUSB0 record wait 1500 pause wait 1500 –r 5"*; this call would be made on a Unix machine and would issue the following commands: record, wait 1500, pause, wait 1500 and repeat five times on ALL cameras. Notice how, when no specific cameras are specified, all cameras are automatically the default.

Ex3. *"ArduinoCameraController.exe –d COM4 –c 1,4,5 snap"* this is another Windows call specifying cameras 1, 4 and 5 to take pictures. This will not affect any of the other cameras connected to the Arduino, and is useful for making selective changes to specified cameras.

Meta-Commands

turnoff	turns cameras off
turnon	turns cameras on
report	shows which cameras are on
snap	takes a picture
record	starts recording (slower than toggle but checks for proper state)
pause	pauses recording (slower than toggle but checks for proper state)
toggle	toggles the record/pause without checking for changes
video	enters video mode
picture	enters picture mode
query	queries for camera status value
init	zooms all the way out and changes to video mode
wait [time]	waits [time] milliseconds (using sleep commands on computer side)
zoomin [speed(0-14)]	zooms in
zoomout [speed(0-14)]	zooms out
focus	autofocus
ftog	focus toggle from manual to automatic
fnear	focuses nearer (not very precise)
ffar	focuses farther (not very precise)

Command Line Argument Switches

Usage in Windows:	ArduinoCameraController.exe [<options>...] <commands...>
Usage in Linux/Mac:	./CameraController [<options>...] <commands...>

Valid options:

-h | --help	Outputs a message similar to this section
-r | --repeat <times>	Send commands <times> number of times
-l | --list	Lists valid meta-commands
-d | --device <device-string>	Specifies device path of Arduino, e.g., “COM4”, “COM5”, “/dev/ttyUSB2” or “/dev/tty.usbserial”. Can also specify “find” and have the program return which port it finds the Arduino on
-c | --camera <camera-numbers...>	Specifies which cameras will be affected (default = all); can list several together, e.g., “–c 5,6,7”
-nw | --no-wait	Bypasses regular delay after commands Useful if you don’t care if the first camera is ready before you send the next command. The built in delays ensure that the camera is in a ready state before it continues with the next command.
-rc | --raw-command | --raw	Sends raw commands rather than meta-commands
-v | --verbose	Displays more status information
-db | --debug	Displays a lot more status information, including how many bytes were sent or received via serial communication

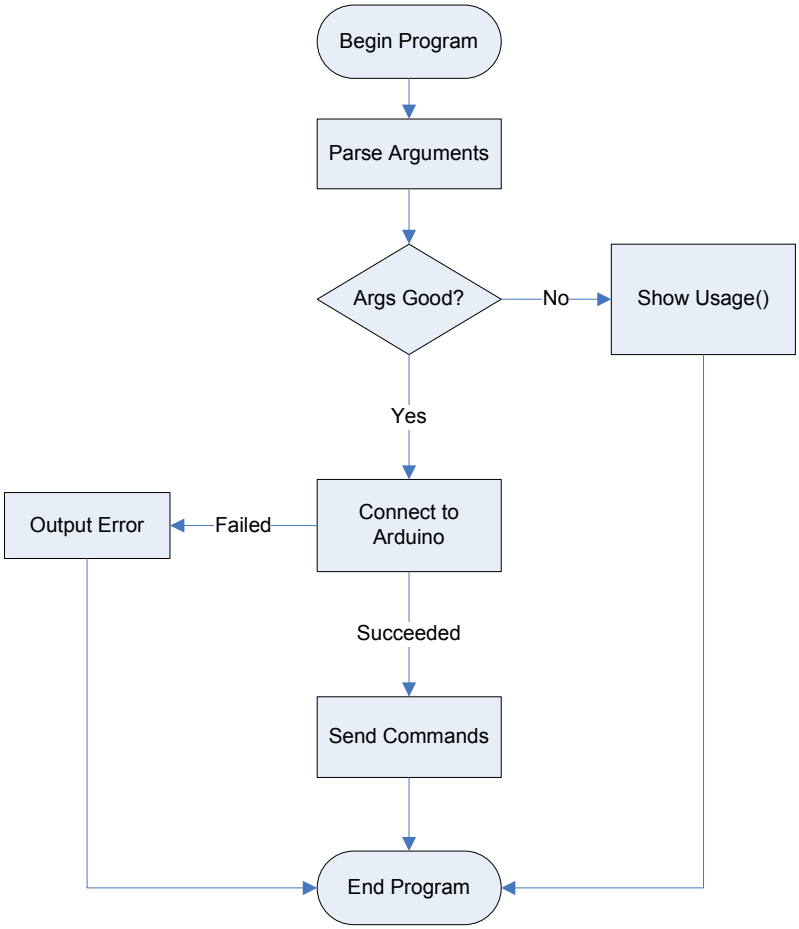
Raw Commands

Raw Commands follow the format “XY”, where X is the operation/command and Y is the camera number. When Y is “z”, this specifies all cameras; otherwise we use “0” for camera 0, “1” for camera 1, etc., continuing up through the ASCII values. The character value for any given camera number N is: “0” + N. The one-letter codes for the X portion of the raw command are as follows:

a – o	control zooming in (a = speed 1 o = speed 15)
A - O	control zooming out (A = speed 1 O = speed 15)
<	zoom in (speed=14-15)
>	zoom out (speed=14-15)
[	zoom in (speed=6)
]	zoom out (speed=6)
:	autofocus
;	turn autofocus on/off
(	manually focus near
)	manually focus far
@	backlight (doesn’t exist on new cameras)
&	grid on/off (for new cams only)
#	turn off viewfinder (sometimes doesn’t work)
s	shoot photo
R	record (checks resulting state is recording)
P	pause (checks resulting state is paused)
T	record toggle (does not check result)
!	turn camera on
x	turn camera off
v	change mode to video (checks resulting mode is video)
p	change mode to picture (checks resulting mode is picture)
q	query camera (cannot specify for all cameras)
r	report cameras (camera number won’t do anything)
~	send hello to Arduino (camera number won’t do anything)

Appendix

Computer Side Flow Chart

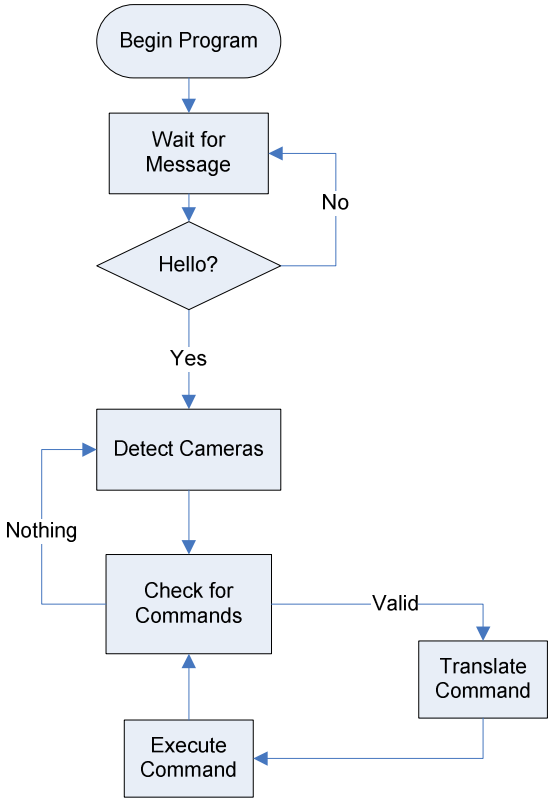


Computer Side Code Notes

The computer side application starts by parsing the arguments supplied. If either not enough or invalid parameters were supplied, the program terminates after outputting the proper usage instructions. Once a useful set of arguments is acquired, we attempt to connect to the Arduino by sending several “hello” commands on the serial port. If we receive no response, we can skip that serial port if searching; or, if given one, we should end with an error message notifying the user that no Arduino is on that port. Once the Arduino is found, we can translate the meta-commands supplied and send the raw commands to the Arduino. The Arduino returns various small messages back to the computer depending on the result of the most recent command. If it failed to send properly, we receive an “F”. If it finished properly and was a command that required camera status feedback, we receive “D”, telling the computer it is done and

allowing us to move on directly to the next command. If we receive “S”, it signifies that the command was sent, but does not mean that the Arduino is finished; we need to wait a specific length of time (depending on what was sent) to allow the particular command to finish. This delay can also be bypassed by specifying “-nw” in the argument list. This delay is tuned so that by the time the delay is over the camera will be back in a ready state and will respond to further commands. If you do not wait, the command you send may not function. Skipping the delay is useful when you are sending commands to multiple cameras and don’t care if the previous camera is ready before you send to the next one. Various outputs may be generated by the Arduino in commands such as “query” or “report”. Report will actually output a total number of cameras detected at the end (very useful in scripts to automate camera processes).

Arduino Side Flow Chart



Arduino Side Code Notes

The program begins by waiting for contact from the computer. Once first contact has been received, the main loop begins. This initial contact loop helps to ensure that communication with the computer does not fail. If contact was not initiated before entering the main loop, reliable communication was found to be very difficult to achieve. To achieve communication properly it is recommended to just plug the USB cable in

and wait several seconds before trying to use the application. This allows the operating system time to set the USB port up.

The main loop consists of checking camera plugs for a signal and periodically checking for a command sent from the computer. If no commands are found, it simply checks all the camera lines and loops through updating whether a camera is on or off. This explains in large part why sometimes you can report and find fewer cameras than expected. This is simply because the program did not have enough time to detect every camera, and usually reporting a few seconds later should fix the problem.

If a command is received, it is translated and executed. Translation turns the one-letter raw command and one-letter raw camera number into a camera pin number, global command, global status and global mode. Global command refers to the actual two-digit hex value for the LANC command we wish to send. The global mode is the mode in which that command exists (normal or special). Normal or special mode refer to another two-digit hex value used in the LANC protocol. Normal is 0x24 and special is 0x40. These two modes allow the use of a separate set of commands. Global status is the status we wish to achieve, if possible, when sending the command. This status is critical to how the “record” and “pause” meta-commands function. They check if the status is reached for a particular camera. If it is, then they stop; if not, they send the command and wait for any changes to the status. This status is used for the record, pause, video and picture commands.

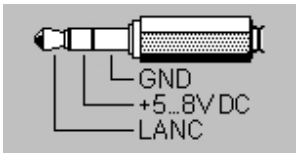
After the command is translated, it is executed. The execution starts by syncing up to whatever camera we wish to send to. We sync to the camera’s LANC signal using a function called pulseIn(). This waits for a signal to stay unchanged for a certain length of time. If we see this, we know we are in the long pause between frames. This is what we are hoping to find. At this point, we wait for a stop bit and assume we are at Bit 0. Then the specific command and mode are written to bytes 1 and 0. This is repeated six times, then the program goes back to the camera-detection and command-monitoring loop.

Cable Limitations

So far, I successfully communicated across fifteen 12-foot extensions chained together along with a three-foot cable plugging into the camera itself. This translates to immunity to signal degradation up to 180 or more feet of cabling. However, extended testing has not been performed on cables of this length.

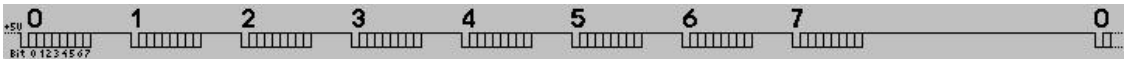
LANC Details

The LANC plug located on the back of the camera is a 2.5mm stereo audio jack with a pin out shown in the figure at right.



GND is what we connect to all of the other cameras — and also the Arduino — to ensure that all signals are using the same reference voltage. The +5/+8 volt section is not used for anything in our application. The LANC signal is fed directly into an Arduino digital I/O port.

The LANC protocol is a series of signals organized in the following fashion:



The figure above shows one “frame” that is sent at 9600 BAUD. Because the BAUD rate is 9600 and the frame contains 8 bytes at 8 bits each, we have 150 frames per second coming out of the camera. The pertinent information is in byte 5 (actually the sixth byte), and is where the status information is found for determining if the camera is in record, pause, video or picture mode. The commands we send are imposed onto bytes 0 and 1. In short, I sync up with the frames; then, once I know I am at the beginning of byte 0, I start writing what I wish the camera to do.

When a command is being sent to the camera, I need to sync to the frame and write onto bytes 0 and 1 as described before; however, the camera must also see this happen at least four times or it will not do anything with it. I send six commands to deal with this issue, and this allows two of the commands to be lost due to noise or the camera not looking at the right time. So far, this has been quite reliable.

The current set of usable commands is the full set I was able to find. No exposure or menu commands were found to work. More information is available in the link below.

If the LANC signal is grounded for >140 ms, the cameras are turned on.

More information (and most of the information gathered) is available here.

COM Port Details

Some things I have noticed about the COM port are as follows.

> It randomly hangs when you send one character at a time. To get around this, I sometimes send "a " when I desire to send an "a" character, because if I do not pad it with the space I will eventually hang the port. This could be a Windows side issue, but the solution works in Linux, Mac and Windows.
>
> The serial port can cause the Arduino to reset by changing the value of the DTR line. I set the DTR line to be unset on creating COM connections for this application because I do not wish the Arduino to be resetting constantly as we ask it to perform operations. The default is to set the DTR line on connect and on close. This can be either useful or annoying depending on the application. Also, the ability to toggle the line in the middle of communication is possible for both operating systems through the use of the ioctl() function in Linux/Mac or the EscapeCommFunction() in Windows.
>
> Com port communication is done at 4800 BAUD between the computer and the Arduino for reliability.

Arduino Details

The Arduino used for this application is an Arduino Duemilanove using the ATmega168 and running at 16MHz. The compiler must be set to this processor type — and specifically the Atmega168 — or the upload will not be successful. The Arduino compiler on which this application was developed was version 0015.

Downloading Camera File Notes

When downloading files from the camera's hard drive, the first step is to attach the USB plug from the computer to the camera. This will bring a blue screen up on the camera and a number of buttons from which to select. This menu is not able to be skipped using LANC commands, and you are required to physically press the computer button before the computer will recognize and mount the drive. On the other hand, once downloading is complete, the cables can be removed; and although the cameras are stuck on a connecting screen, a simple turnoff and turnon command sequence using the camera controller will set them back to the idle state for further recording.

Camera Zoom Levels

The camera controller can cause the cameras to zoom in or out; however, the ability to be precise and determine the exact level of zoom is absent. I have found through testing that the last five or six highest speeds for zooming are somewhat unreliable. To perform zooms that are repeatable and accurate, you would be best using the lowest five or so zoom speeds. These will not jump sporadically and should be a repeatable (or at least somewhat repeatable) means of resetting zoom levels.

Parts and Circuitry Details

The 3.5mm audio connectors are stereo SWITCHCRAFT 35RAPC4BV4 and were purchased from Newark Canada. This audio jack was screwed into a plate made with 36 holes and was made out of conductive metal. Because the plate was conductive, the audio connectors all share a common ground (otherwise soldering would have needed to be done on each one). Thus, if any happen to get loose and come out, it should be noted that it would not function properly, if at all. The signal we are interested in is being carried across a wire from the audio jack to the Arduino input directly. All of the audio jacks have one such wire that connects them to the board. One special audio jack also takes its ground and connects it to the Arduino ground, allowing the whole device to share the same ground. This is critical, as the Arduino would not be able to see a proper signal if it was not sharing ground with the audio jacks.

The case, pin headers, Arduino and shrink tubing all came from Lee's Electronics.

How to replace a Camera

A small recommendation would be to label the plug coming from the cable and connect it to the corresponding labeled port on the camera controller box. Otherwise, any camera will work perfectly using the camera controller. However, if you wish to automatically download the video and specify a camera number using the USBDriveController application, you will need to refer to the manual for that application.

Troubleshooting

If a problem arises that deals with communication there are several lines in the software that may be tweaked to try and fix it.

In serial.h:

Lines 39,40 deal with serial port reads and if a lot of time outs occur then increasing these values may help fix that problem.

In serial.cpp

Line 463-542 is the serial delay function which will attempt to delay enough time so that the camera is responsive again for further calls. If you are having chains of commands fail or having various commands not work then try increasing the delays in this function.

In CameraController.pde

In the first several lines there are some defines and a set of uint16_t variables defined that all correspond to special timeouts or delay times. The set of uint16_t correspond to the 9600 baud LANC signal spacing between bits and bytes. The various timeout defines can be increased if it is thought that it is the Arduino that is failing to operate properly (should only happen if a totally new camera was implemented or a different Arduino was being used). None of these values should really need to be changes so do so with caution.