Lessons from design, construction, and use of various multicameras

Henry Dietz, Clark Demaree, Paul Eberhart, Chelsea Kuball, and Jong Wu

PMII, 8:50AM January 29, 2018

University of Kentucky
Electrical & Computer Engineering

University of Kentucky

What is a multicamera?

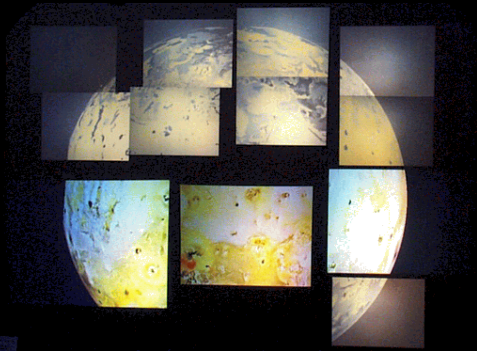
- Incorporates 2 or more component cameras
- Behaves like a single system
- Offers better performance or special abilities
- Aka:
  Array camera
  Cluster camera
  Super-camera

Why bother?

- Built 1st linux cluster supercomputer, 1994

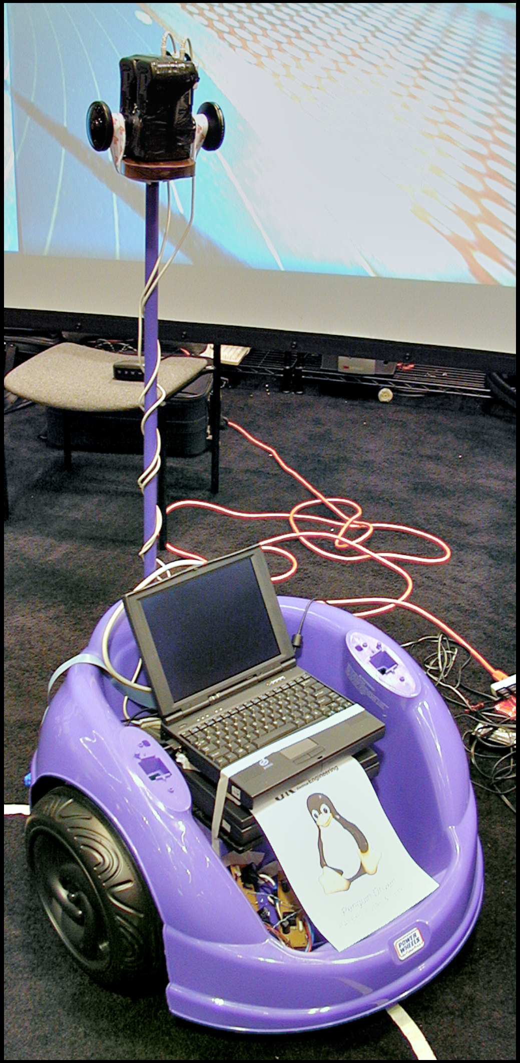
University of Kentucky

Why bother?



- Built 1st linux cluster supercomputer, 1994
- Built video walls to prove tight coupling

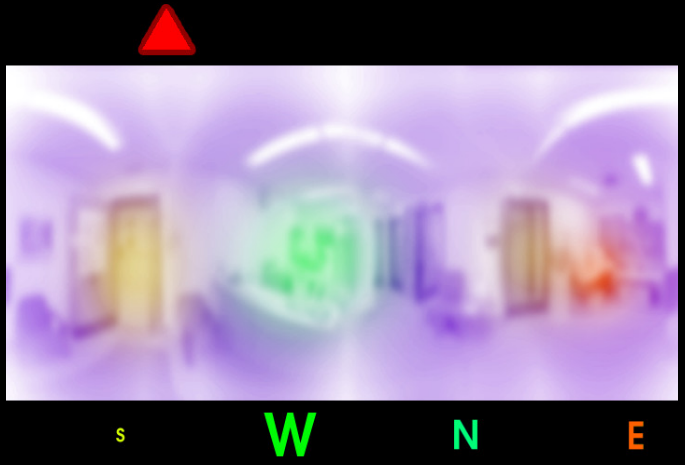
Autonomous 360° system, 1999

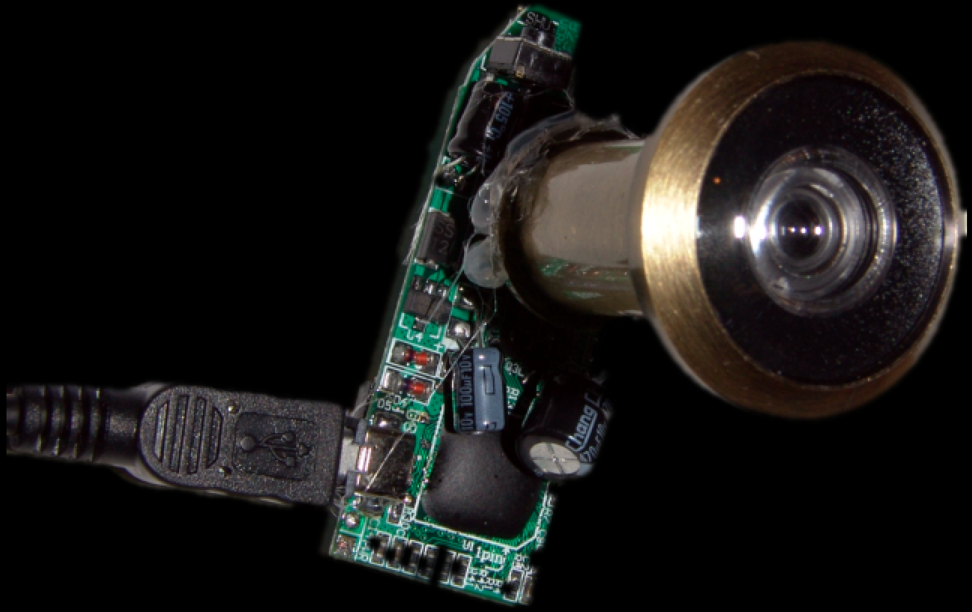


Cameras: 2 Nikon 950
2 Olympus D-340R
Control: RS232C tether
Purpose: autonomously wander SC99 exhibit hall capturing 360° images sent to cluster video wall

University of Kentucky

FireScape, 2006





Cameras: 3 webcam
Control: USB tether
Purpose: 360° augmented reality to guide firefighters in burning buildings

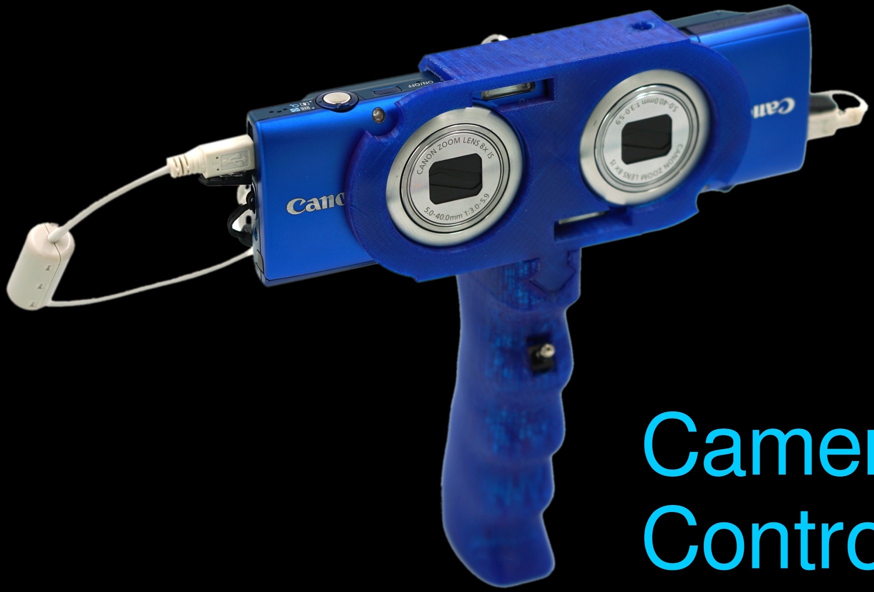
University of Kentucky

AVA:
Ambient Virtual Assistant, 2008

Cameras: 23 UniBrain Fire-i400
Control: FireWire tether
Purpose: surveillance and “smart space”

University of Kentucky

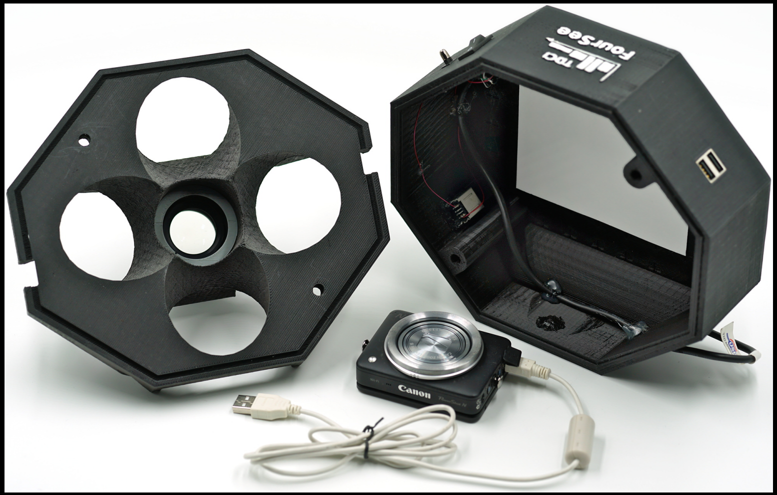
A4K2: Stereo Capture, 2014

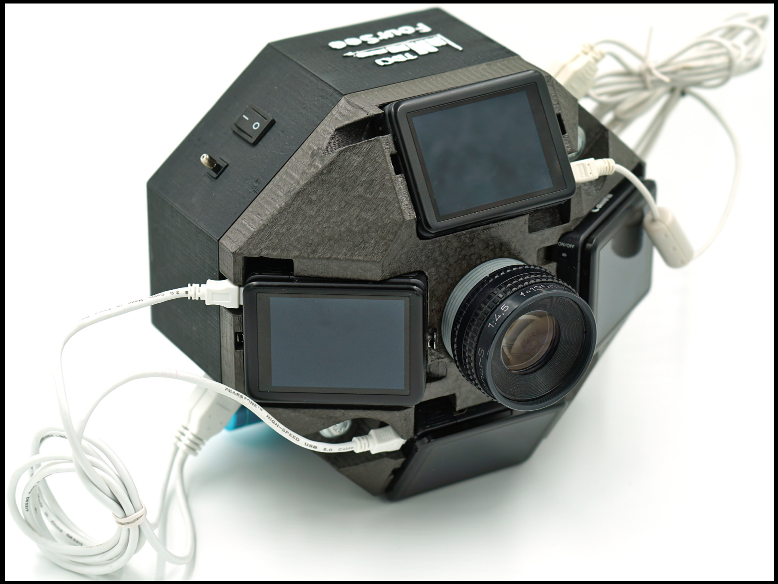


Cameras:	2 Canon A4000
Control:	USB/CHDK program
Purpose:	stereo capture

University of Kentucky

FourSee, 2015



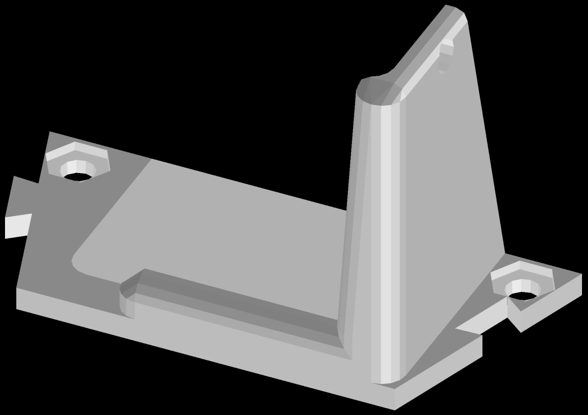


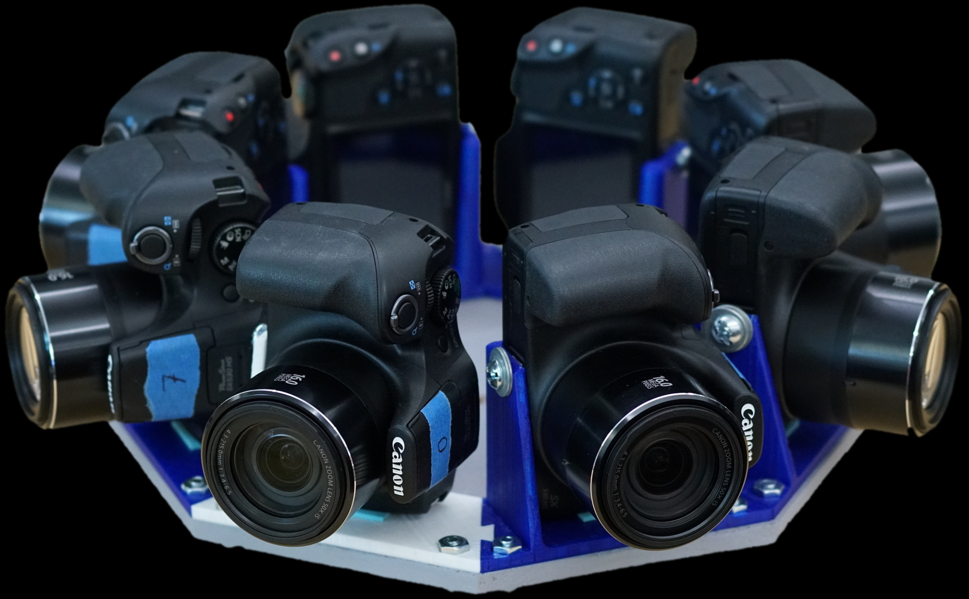
Cameras: 4 Canon N
Control: USB/CHDK program
Purpose: TDCI capture

University of Kentucky

KREight, 2017

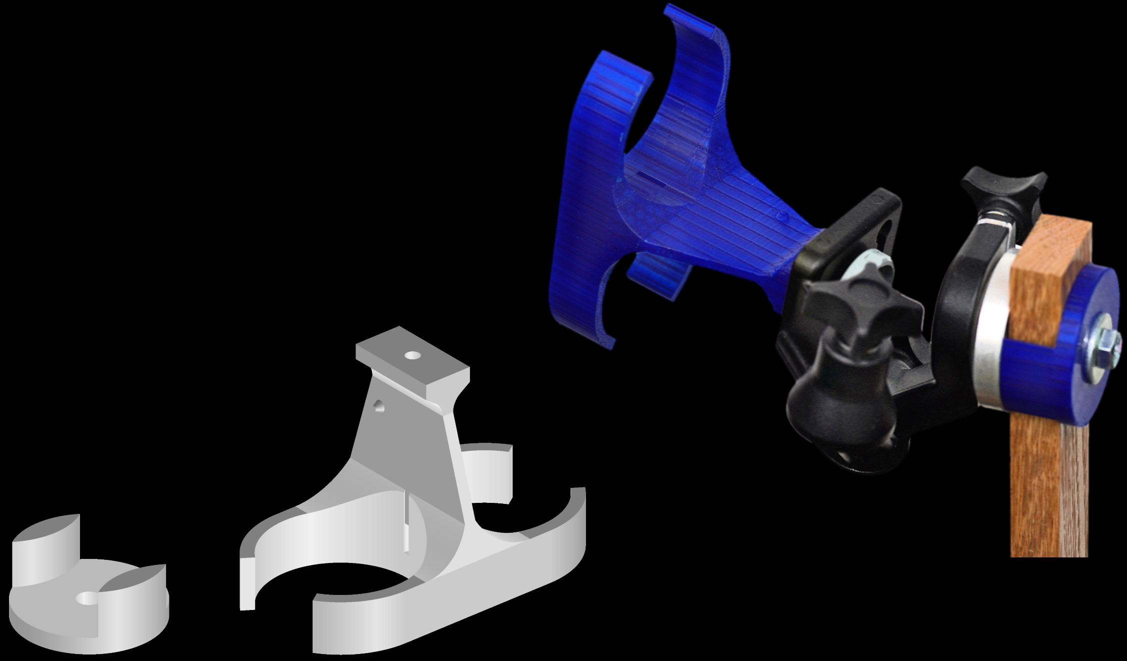
Cameras: 8 Canon SX530 HS
Control: USB/CHDK program
Purpose: 360° capture

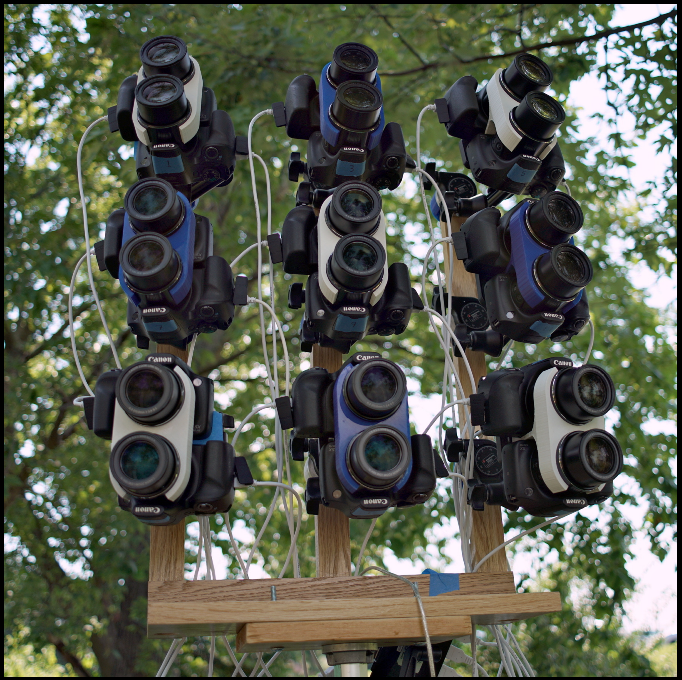




University of Kentucky

KREighteen, 2017

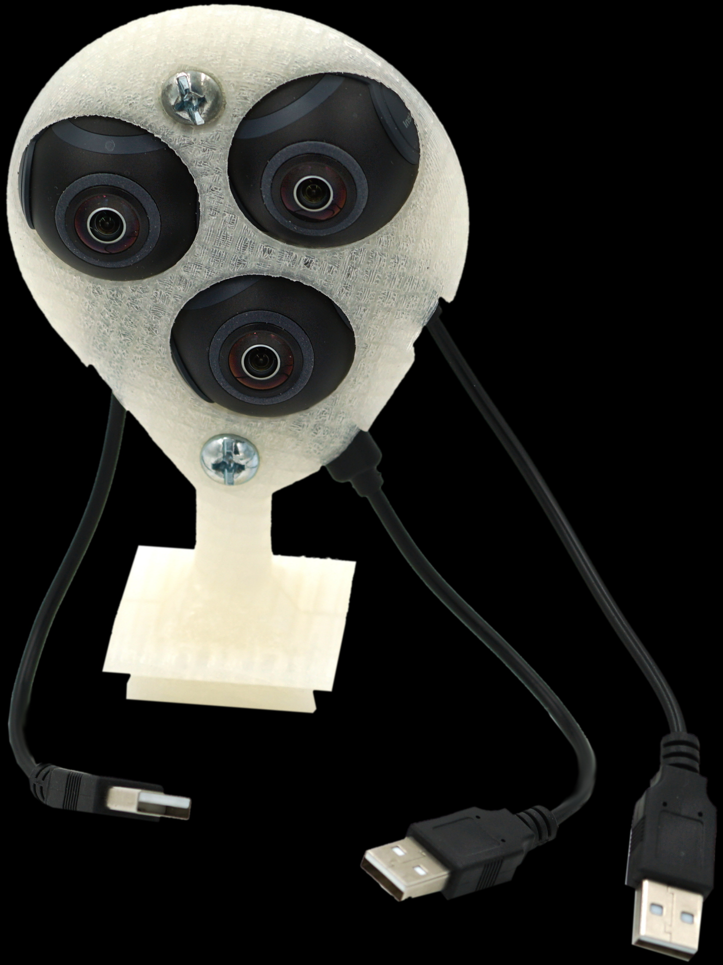




Cameras: 18 Canon SX530 HS
Control: USB/CHDK program
Purpose: TDCI capture

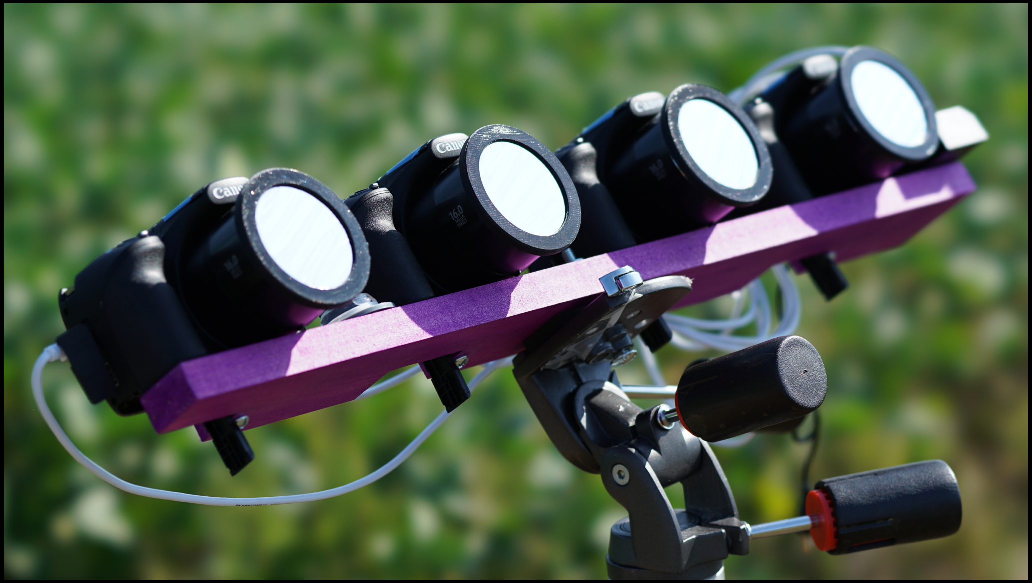
Kodama, 2017

Cameras: 3 Insta360 Air
Control: USB tether
Purpose: 360° TDCI capture



University of Kentucky

MASK: Multicamera Array Solar from Kentucky, 2017



Cameras: 4 Canon SX530 HS

Control: USB/CHDK program

Purpose: Multispectral/HDR TDCI

University of Kentucky

Lessons learned

- Programmable camera modules
- Synchronization of local clocks
- Local storage and processing
- Physical mounting and alignment
- Live view
- Fault tolerance

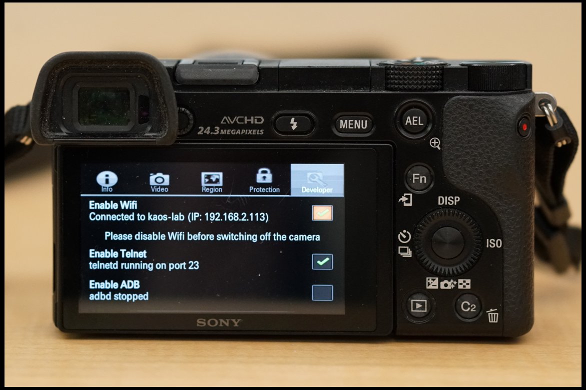
Programmable camera modules

- Cameras = computers, **NOT** film exposers
  - Offload computation to coprocessors
  - Smarter response to tethered control
- Frankencamera
- Raspberry Pi camera modules
- Consumer programmable cameras:
  - Canon Hack Development Kit (CHDK)
  - Magic Lantern (ML)
  - OpenMemories

Programmable camera modules

Consumer programmables

- OpenMemories in most Sony;
  Linux + Android app (PlayMemories API)
- Magic Lantern (ML) in some Canon EOS;
  DOS + C (compiled/scripts), low-level access
- Canon Hack Development Kit (CHDK) in
  most Canon PowerShots (including <$100);
  DOS + C (compiled) + BASIC/Lua (scripts)



Enable Wifi ☑
Connected to kaos-lab (IP: 192.168.2.113)

Please disable Wifi before switching off the camera

Enable Telnet ☑
telnetd running on port 23

Enable ADB □
adbd stopped

CHDK Lua

Canon Hack Development Kit
Lua scripting reference card

Version 20131022 for `CHDK 1.3.0`

`http://aggregate.org/DIT/CHDK/`

Prof. Hank Dietz
Electrical and Computer Engineering Dept.
University of Kentucky
Lexington, KY 40506-0046
`hankd@engr.uky.edu`

Overview

CHDK, the Canon Hack Development Kit, gives various Canon powerShot cameras new abilities, including the ability to run scripts written in uBASIC or Lua. Recent improvements even allow Lua commands to be exected via USB tethering.

There are many alternative ways to do things in Lua, both functions and constants: 0/1 usually can be `false`/`true`. Some functions listed on a single line to save space.

Focus, IS, & Zoom

`mm=get_focus(); set_focus(mm)`
focus distance in mm when shooting

`v=get_focus_mode()`
0=auto, 1=manual, 3=∞, 4=macro, 5=supermacro

`v=get_focus_ok()`
0=focus not ok, 1=ok *iff* `get_focus_state()˜=0` and `get_shooting()==1`

`v=get_focus_state()`
0=failed, >0=auto success, <0=manual

`set_aflock(lock)`
lock/unlock autofocus

`v=get_IS_mode()`
image stabilization mode; 0 continuous, 1 shoot only, 2 panning, 3 off

`s=get_zoom(); set_zoom(s); set_zoom_rel(s)`
zoom position in steps, or +/- relative steps

`set_zoom_speed(speed)`
set zoom to *speed*% of maximum (typically 5% to 100%)

`v=get_zoom_steps()`
number of zoom steps supported

`v=get_dofinfo()`
depth of field fields: `hyp_valid`, `focus_valid`, `aperture`, `coc`, `focal_length`, `eff_focal_length`, `focus`, `near`, `far`, `dof`, `hyp_dist`, `min_stack_dist`

Exposure

Exposure parameters can be measured in many different units. APEX (Additive system of Photographic EXposure) uses a log scale in which Ev=Av+Tv=Bv+Sv; Canon/CHDK uses APEX*96 for exposure. Ev is exposure, Av is aperture, Tv is shutter time (-96*log2(seconds)), Bv is luminance, and Sv is ISO sensitivity. Values can be actual `real` (aka `direct`) or rounded `market` values. Functions named `user` are for Manual exposure mode and ones with `id` select by index in table of camera values. Functions use `aperture`*1000; `rel` means +/- offset from current value.

`v=get_av96(); set_av96_direct(a)`

`set_av96(a)`

`v=aperture_to_av96(a)`

`v=av96_to_aperture(a)`

`v=get_bv96()`

`v=get_ev(); set_ev(a)`

`v=get_sv96(); set_sv96(s)`

`v=get_iso_real(); set_iso_real(a)`

`v=get_iso_market()`

`v=get_iso_mode(); set_iso_mode(a)`
market value or 0=auto ISO

`v=iso_to_sv96(s); v=sv96_to_iso(s)`

`v=iso_real_to_market(s)`

`v=iso_market_to_real(s)`

`v=sv96_real_to_market(s)`

`v=sv96_market_to_real(s)`

`t=get_tv96(); set_tv96_direct(t)`

`set_tv96(t)`

`v=get_user_av_id(); set_user_av_id(a)`

`v=get_user_av96(); set_user_av96(a)`

`set_user_av_id_rel(a)`

`set_user_tv96(t)`

`set_user_tv_id(t); set_user_tv_id_rel(t)`

`v=usec_to_tv96(t); v=tv96_to_usec(t)`

`v=seconds_to_tv96(n,d)`
converts *n/d* seconds into tv96 units

`v=get_nd_present()`
have neutral density filter? 0=no, 1=yes, 2=yes+aperture

`set_nd_filter(v)`
controls neutral density filter: *v*=0 off, 1 in, 2 out

`h,t=get_live_histo()`
returns live histogram and total number of pixels

Camera Functions

`v=get_drive_mode()`
0=single shot, 1=continuous, 2,3=self timer

`v=get_flash_mode()`
flash mode: 0=auto, 1=on, 2=off

`v=get_flash_params_count()`
number of flash memory (not strobe) parameters

`v=get_flash_ready()`
flash ready to fire? 0=no, 1=yes

`v=get_meminfo()`
fields: `name`, `chdk_malloc`, `chdk_start`, `chdk_size`, `start_address`, `end_address`, `allocated_size`, `allocated_peak`, `allocated_count`, `total_size`, `free_block_max_size`, `free_block_count`, `free_size`

`rec,vid,mode=get_mode()`
rec `true` if in record mode, *vid* `true` if in video mode, *mode* is magic mode number

`v=get_movie_status()`
video recorded to SD? 0,1=stopped/paused, 4=recording, 5=stopped but writing to SD card

`v=get_orientation_sensor()`
returns camera orientation in degrees

`str,num=get_parameter_data(id)`
reads flash memory parameter *id*

`v=get_prop(p); v=set_prop(p,v)`
access PropertyCase value

`v=get_prop_str(p); s=set_prop_str(p,v)`
access PropertyCase string value

`v=get_propset()`
identifies PropertyCase set used by this camera

`v=get_shooting()`
ready to shoot? (half press, focus, and exposure set)

`v=get_temperature(w)`
reads temperture of 0=optics, 1=sensor, 2=battery

`v=get_vbatt()`
read battery voltage in mV

`v=get_video_button()`
does camera have a video button? 0=no, 1=yes

`v=is_capture_mode_valid(n)`
true if *n* is a valid mode number

`v=set_capture_mode(n)`
sets mode and returns true if in record mode

`v=set_capture_mode_canon(n)`
sets mode by PropertyCase and returns true if camera is in record mode

`set_led(a,b[,c])`
a is LED number; *b*=0 off or 1 on; *c* is brightness 0-200

`set_movie_status`(*v*)
1=pause recording video, 2=resume recording, 3=stop recording

`set_record`(*v*)
0 (or false) sets play mode, 1 (or true) sets record

`shut_down()`
like `post_levent_to_ui('PressPowerButton')`

Buttons

Buttons are camera dependent, although all have `"shoot_half"` and `"shoot_full"`.

`click`(*button*)
simulate press, then release, of button *b*

v=`is_key`(*button*); *v*=`is_pressed`(*button*)
1 if *button* was; is being pressed

`press`(*button*); `release`(*button*)

`shoot()`

`wait_click`(*[t]*)
wait up to *t*/1000s for any key to be clicked

`wheel_left()`; `wheel_right()`
simulate wheel move one click ccw; cw

`set_exit_key`(*b*)
set *b as the key to terminate this script*

SD Card Functions

v=`get_disk_size()`
size of SD card in KB (1024B) units

v=`get_exp_count()`
get number of shots in a session

v=`get_image_dir()`
directory where most recent exposure was written

file=`file_browser`(*path*)
lets user select a file

v=`get_free_disk_space()`
space remaining on SD card in KB (1024B) units

v=`get_jpg_count()`
number of JPG shots that would fit on SD card

part=`get_partitionInfo()`
fields: `count`, `active`, `type`, `size`

`set_file_attributes`(*file*,*a*)
set attributes of *file* to bits in *a*: 0x1=read only, 0x2=hidden, 0x20=archive

`swap_partition`(*n*)
make partition *n* active

Time & Scheduling

v=`autostarted()`
return 1 (true) is script was autostarted

v=`get_autostart()`; `set_autostart`(*v*)
autostart can be 0=off, 1=on, 2=once

v=`get_tick_count()`
clock time in 1/1000s units

v=`get_time`(*unit*); *v*=`get_day_seconds()`
time specified by *unit* string: `Y`ear, `M`onth, `D`ay, `h`our, `m`inute, `s`econd; or simply seconds since midnight

oc,*oms*=`set_yield`(*c*,*ms*)
set maximum number of Lua VM instructions to contiguously execute as *c**100 and maximum time as *ms*; old values are returned

`sleep`(*time*)
Sleep for time in 1/1000s units

Display & Text Console

`set_backlight`(*v*)
LCD backlight on/off

i=`get_draw_title_line()`;`set_draw_title_line`(*i*)
CHDK <ALT> line on LCD on/off

`cls()`; `console_redraw()`
clear/redraw mini-console screen

`print`(...)
write args to mini-console

`print_screen`(*nnnn*)
if *nnnn*=0, disables echo to log file; >0 logs to new file `LOG_`*nnnn*`.TXT`; <0 appends to log file

`set_console_autoredraw`(*n*)
n=1 enables auto update of log file and LCD; 0 disables; -1 updates log file only

`set_console_layout`(*x1*,*y1*,*x2*,*y2*)
position and size in characters; 0,0,45,14 is full screen

LCD Graphics

Drawn on LCD, but overwritten by any update. Colors are non-portable 0-255 Canon palette or portable: 256 (`transparent`), 257 (`black`), 258 (`white`), 259 (`red`), 262 (`green`), 265 (`blue`). Edge thickness also can be set.

`draw_clear()`

`draw_ellipse`(*x,y,a,b,c*)

`draw_ellipse_filled`(*x,y,a,b,c*)

`draw_line`(*x1,y1,x2,y2,c*)

`draw_pixel`(*x,y,c*)

`draw_rect`(*x1,y1,x2,y2,c,thick*)

`draw_rect_filled`(*x1,y1,x2,y2,cfill,c,thick*)

`draw_string`(*x,y,text,cf,cb*)

v=`textbox`(*title,prompt,def,maxlen*)
gets a string from user input

Raw

v=`get_raw()`; `set_raw`(*v*)
enable/disable saving raw images

v=`get_raw_count()`
number of raw shots that would fit on SD card

v=`get_raw_nr()`; `set_raw_nr`(*v*)
noise reduction enabled/disabled

`raw_merge_start`(*op*)
start raw merging; *op* can be 0 (sum) or 1 (average)

`raw_merge_add`(*file*)
adds raw *file* to the merge

`raw_merge_end()`
completes merge; result is `SND_`*XXXX*`.CRW`, where *XXXX* is `get_exp_count()` % 10000

`set_raw_develop`(*file*)
next shot develops raw *file* into JPEG

CHDK Functionality

`enter_alt()`; `exit_alt()`
enter/exit CHDK <ALT> mode

v=`get_buildinfo()`
fields: `platform`, `platformid`, `platsub`, `version`, `os`, `build_number`, `build_revision`, `build_date`, `build_time`

i1[,i2][,s][,t]=`get_config_value`(*ConfigId[,def]*)
get specified CHDK configuration value

v=`get_histo_range`(*lo,hi*)
percentage raw buffer pixels in [*lo*, *hi*]

`set_config_value`(*ConfigId[,i1][,i2][,s1][,t]*)
set specified CHDK configuration value

`shot_histo_enable`(*v*)
enable/disable computing shot histograms

Programming

v=`bitand`(*a,b*)
bitwise and; also `bitor`, `bitxor`, `bitshl` (<<), `bitshri` (int >>), `bitshru` (unsigned >>)

v=`bitnot`(*a*)

v=`peek`(*addr[,size]*); *s*=`poke`(*addr,v[,size]*)
load/store memory[*addr*]; size is 1/2/4, default 4, for char/short/int

v=`call_func_ptr`(*fptr*`,...`)
calls compiled C function at ARM address *fptr*, returns `R0`

Motion Detection

v=`md_motion_detect`(...)
number of zones in which motion was detected; many arguments control detection

v=`md_get_cell_diff`(*x,y*)
returns unsigned [0,255] difference in last two readings of cell *x,y*

v=`md_get_cell_val`(*x,y*)
returns unsigned [0,255] value of cell *x,y* (for Y, U, V, R, G, or B channel specified)

`md_af_on_time`(*d,t*)
show motion detected by autofocus assist lamp; delay *d**10ms before on; *t**10ms before off; 0,0 disables

Tone Curves

Only for cameras using 10-bit raws. There are 5 states, 0-4: no curve, custom file, +1 Ev, +2 Ev, and auto dynamic range enhancement.

v=`get_curve_state()`; `set_curve_state`(*v*)
get/set tone curve state

file=`get_curve_file()`; `set_curve_file`(*file*)
get/set currently loaded tone curve

Synchronization of local clocks

- Capture synchronization is hard, right?
  - **No.**
  - Open-loop triggering of complex behavior is unreliable; camera might not be ready
- Synchronized local clocks allow cameras to internally schedule preparation for actions

University of Kentucky

Local storage and processing

- Multicameras often create a huge volume of data in real time; sending it to a central unit for storage/processing is a serial bottleneck
  - E.g., Kodama saturates most USB
  - Path off sensor much faster than off camera
- Local storage faster than link off camera
- In-camera compression/filtering:
  - Region of interest (ROI)
  - Local feature extraction, ROI selection (e.g., PowerShots recognize faces)

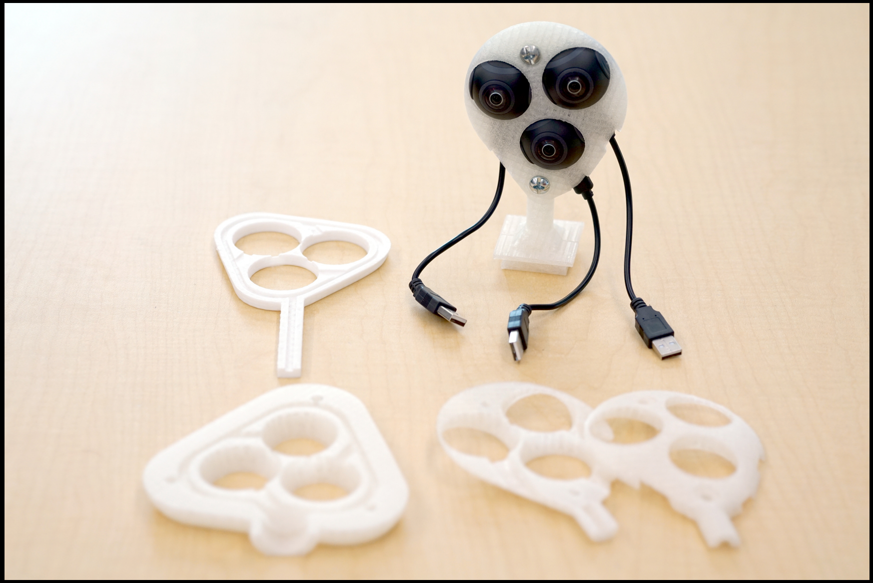
Physical mounting & alignment

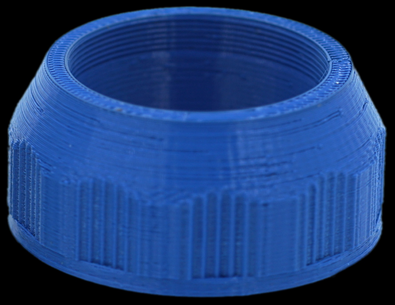
- Most common reason for problems!
- Computationally correct for misalignment?
  - Computationally expensive
  - Might require calibration process
  - Somewhat inferior image quality
- Approaches:
  - Rapid prototyping (e.g., 3D printing)
  - Fixed vs. adjustable mounts

Physical mounting & alignment

Rapid prototyping (3D printing)

- Complex shapes → modular components (consumer stuff often has complex shapes)
- Fast & cheap to produce & ***iterate*** design
- Strong parts with tight tolerances

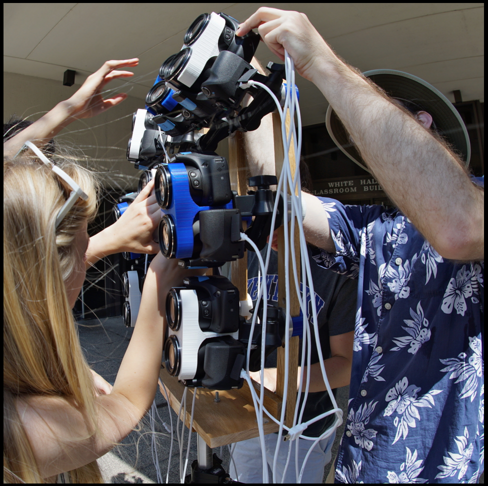


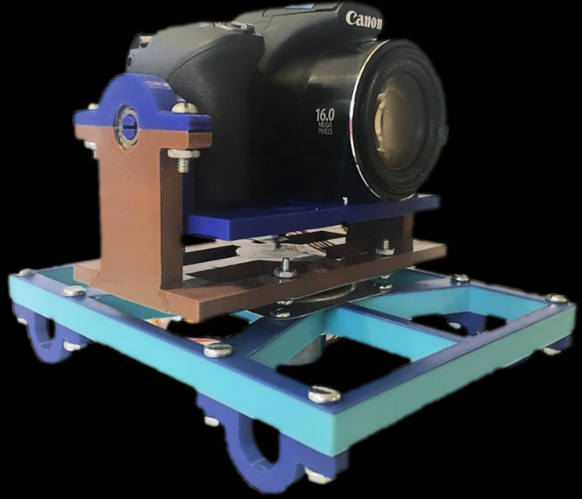


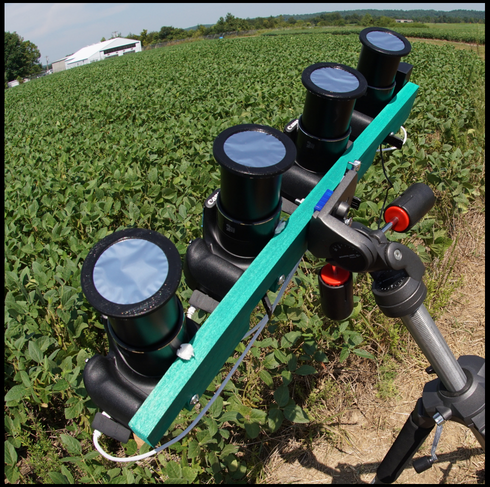
Physical mounting & alignment

Fixed vs. adjustable mounts

- Definition of “Adjustable”: **Will** be out of adjustment.
- **Carefully tweaked fixed positions work.**
- Computer-controlled adjust?







University of Kentucky

Live view

- A multicamera does **NOT** inherently have a live view even if each component does (images may require processing to view)
- Live view display must be visible from where you are to be useful
  - Unobstructed tilt/pivot LCDs
  - Remote live view (awkward for aiming)

Fault tolerance

- Many components ⇒ high system failure rate
  - Permanent failures are rare
    (SD lifetime write limit is most common)
  - Dead battery, loose cable, full SD card, …
    (bring spares and tools)
- Misconfigurations are common
  - Make configuration obvious
    (e.g., label/color-code parts, show IDs)
  - Provide for out-of-band/field configuration
  - Leave an audit trail

University of Kentucky

Conclusion

- People still think they're using film cameras
  - Users ignore programmability
  - Manufacturers don't support programming
- Component camera = computer + camera
- Multicamera should leverage commodity parts

A multicamera *is* a cluster computer.

Aggregate.Org
UNBRIDLED COMPUTING

University of Kentucky

Questions?

Aggregate.Org

UNBRIDLED COMPUTING

University of Kentucky