Self-Contained, Passive, Non-Contact, Photoplethysmography: Real-Time Extraction Of Heart Rates From Live View Within A Canon PowerShot

Henry Dietz, Chadwick Parrish, Kevin Donohue

COIMG-146, 9:10AM, January 17, 2019

University of Kentucky
Electrical & Computer Engineering

University of Kentucky

Photoplethysmography (PPG)

- The detection of blood volume (or flow or pressure) changes by optical means
- Various types of PPG:
  - Nearly all are Non-Invasive
  - Active vs. Ambient (passive) Lighting
  - Contact vs. Non-Contact
- A very crowded field, >100 papers/year!
  - Google Scholar shows ~26,900 articles
  - Low-cost active contact sensors common

Our Goal

- Make a PPG device that is:
  - Non-Invasive
  - Able to use Ambient Lighting
  - Non-Contact
  - Operates in Real Time
  - Fully Self-Contained
  - Cheap, even as a prototype (e.g., <$100)
  - Potentially scaleable to Simultaneously Monitor Multiple People

Not so many folks trying to do that… ;-)

University of Kentucky

Algorithms

- Many different algorithms are viable, but:
  - Poor **SNR** for ambient non-contact signal
  - The **shape** of the waveform isn't simple
  - Ambient lighting and autoexposure drift over time, often adding complex **trends**
- FFT-based analysis is complicated by SNR and waveform shape (many components)
- COIMG-132, "Autocorrelation-Based, Passive, Non-Contact Photoplethysmography: Computationally-Efficient, Noise-Tolerant, Extraction of Heart Rates from Video"

University of Kentucky

The Autocorrelation Algorithm

- The COIMG-132 autocorrelation approach has four major logical components:
  - Reduction of each sync. image to 1 value
  - Detrending of the value waveform
  - Autocorrelation
  - Selection of the “best” correlation
- Floating-point implementation in MatLab

- All we have to do is run this in a camera...

University of Kentucky

The Target Platform: Canon PowerShot (ELPH180)

- Canon makes a wide variety of compacts
- Typical features for ***under $100*** :
  - Full-featured 20MP 12BPP camera
  - Body less than 4” x 2.5” x 1”
  - Includes battery and charger

University of Kentucky

The Target Platform: Canon PowerShot CHDK

- **CHDK: Canon Hack Development Kit**
  - Control of ***all known*** camera functions
  - Run compiled C / native ARM code
  - Scripts in BASIC and Lua
  - CHDKPTP tethering, etc.

University of Kentucky

The Target Platform: Canon PowerShot CHDK

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/unloack 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

v=`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 cw; ccw

`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

le=`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(`*le*,*a*`)`
set attributes of *le* 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: `Year`, `Month`, `Day`, `hour`, `minute`, second; or simply seconds since midnight

oc,*ms*=`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*,*c* *fl*,*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(` *le*`)`
adds raw *le* to the merge

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

`set_raw_develop(` *le*`)`
next shot develops raw *le* 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`

i[,*i2*[,*s*][,*t*]]=`get_config_value(`*Con gId*[,*def*]`)`
get specified CHDK configuration value

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

`set_config_value(`*Con gId*[,*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

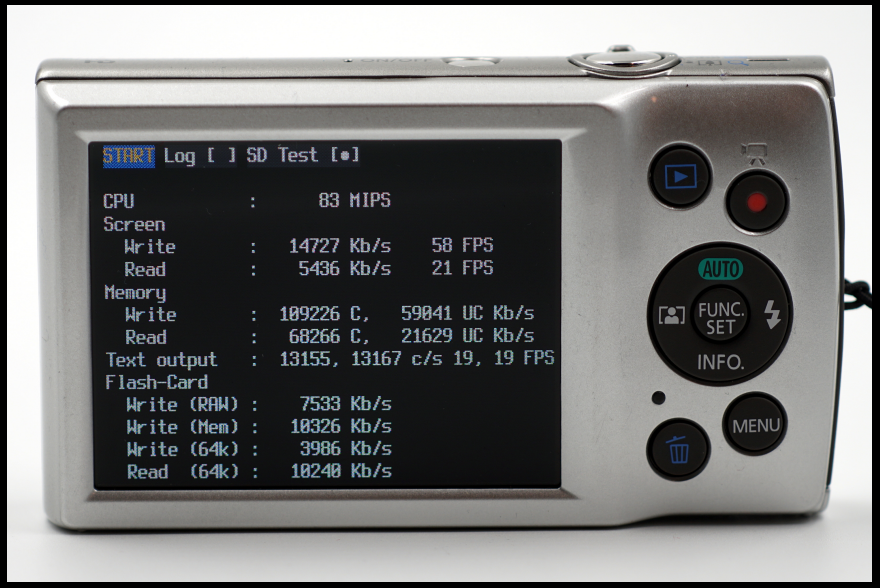
Only for cameras using 10-bit raws. There are 5 states, 0-4: no curve, custom file, +1 Ev, +2 Ev, and auto dynamic range enhancement.

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

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

University of Kentucky

The Target Platform:
Flexible, Self-Contained, Wimpy



- Dual-core ARM around 80MIPS, integer; BASIC and Lua scripts run even slower
- Small main memory (mostly frame buffer)
- CCD with slow readout, live view

University of Kentucky

Approximating COIMG-132 Inside A Wimpy PowerShot

- All integer math
- Completely incremental processing
- **KOBRE** is CHDK Lua script + C code
  - Less than 200 lines of Lua script
  - Intercepts live view feed using C code compiled-into motion detection module
- Real-time display of BPM on rear LCD

University of Kentucky

Approximating COIMG-132:
Reduction to 1 value/sample

- Frame image capture is too slow; intercept **YUV**-encoded live view feed
- Live view isn't constant update rate
  - Lua script uses system time in ms to stall for constant rate, abort if late
  - Call `md_detect_motion()` to sample
  - Pixel values are added over region of interest (ROI)

University of Kentucky

Approximating COIMG-132: Detrending of values

- Trends come from:
  - Changes in subject lighting
  - Changes in live-view autoexposure
- Detrending a waveform takes computation
- Initially, simply didn't detrend...
  but would often latch at highest allowed BPM
- Detrending by color:
  - Use color channel difference, Y-Red
  - Too slow in Lua
  - Modified `md_detect_motion()`

University of Kentucky

Approximating COIMG-132: Autocorrelation

- Autocorrelation is basically differencing, which can be done fully incrementally using circular `vals[]` and `difs[]` arrays
- For each new sample:
  - For all BPM periods `i` (index deltas)
    - Compute difference squared, note max
    - Subtract oldest from `difs[i]`
    - Add new to `difs[i]`
  - Overwrite old value in `vals[]` with new
- This is fast enough to do in Lua...

University of Kentucky

Approximating COIMG-132: Autocorrelation

```
old = vals[1+bitand(mask, fno)]

-- for all possible wavelengths
for i=minfw,maxfw do
 if fno >= i then
  pv = vals[1+bitand(mask, fno-i)]
  a = v - pv
  d = difs[i] + (a * a)

  -- subtract out old value
  if fno >= i+depth then
   r = old - pv
   d = d - (r * r)

   -- new biggest (worst)?
   if d > maxd then
    maxd = d
   end
  end
  difs[i] = d
 end
end

-- save new value
vals[1+bitand(mask, fno)] = v
```

University of Kentucky

Approximating COIMG-132:
Selection of “best” correlation

- Originally, “best” correlation was simply the period `i` of the smallest `difs[]` value… that was inconsistent across runs
- Selection of “best” is now a weighted average:
  - For each BPM period `i`
    - Add `max-difs[i]` into `sumd`
    - Add `i*(max-difs[i])` into `sumi`
  - Best is BPM with period `sumi/sumd`

Approximating COIMG-132:
Selection of “best” correlation

```
-- compute weighted average
if maxd > 0 and fno >= (maxfw + mask) then
 for i=minfw,maxfw do
  d = maxd - difs[i]
  sumd = sumd + d
  sumi = sumi + (d * i)
 end
 besti = sumi / sumd
end

fno = fno + 1

return besti
```

University of Kentucky

Parameters To The Lua Script

- The Lua script has several parameters that can be set at run time:

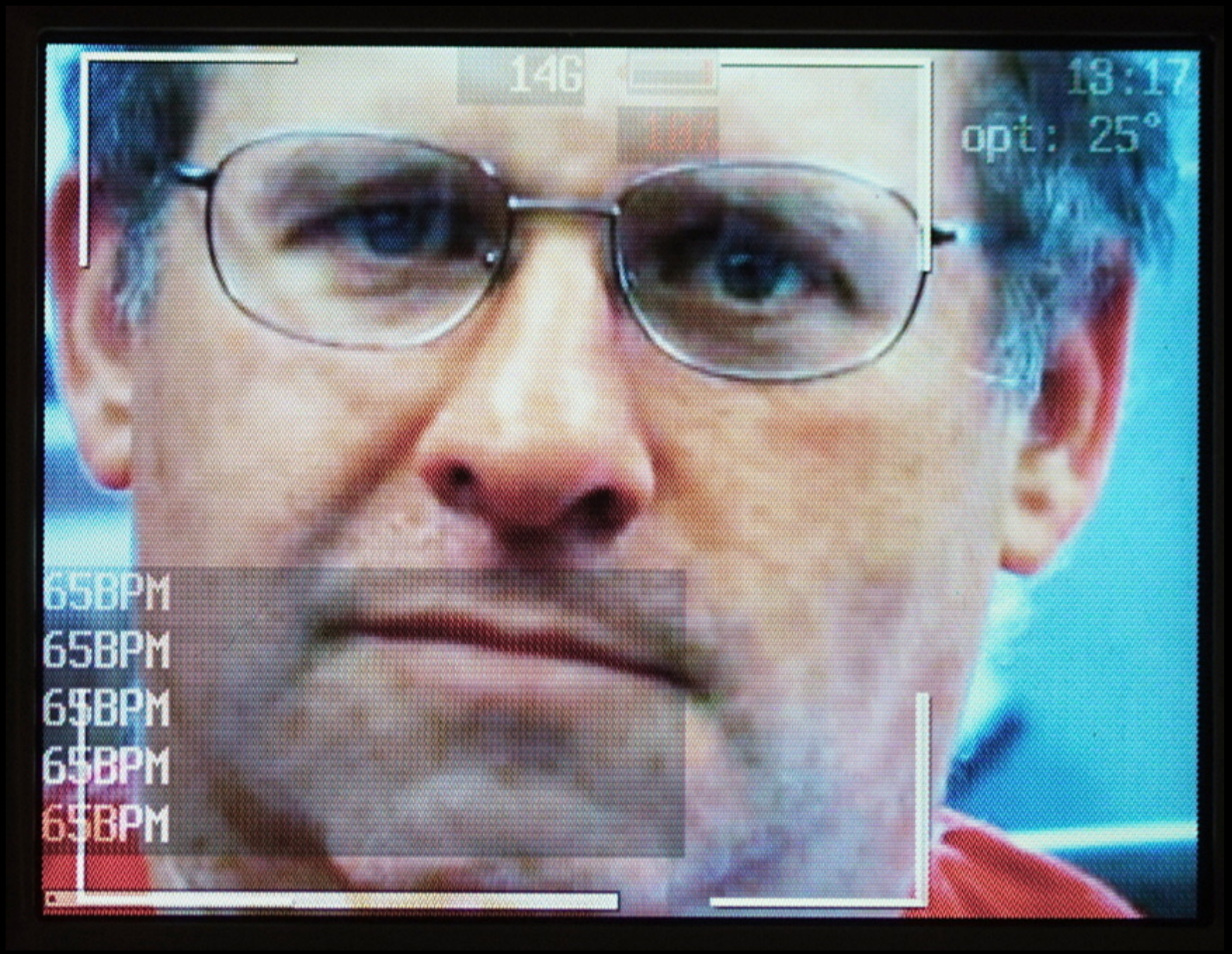
Parameter	Default	Range
`Lowest BPM`	60	30-60
`Highest BPM`	100	100-200
`FPS Target`	24	10-30
`UYVRGB Color`	2 (Y)	0-6 (6 is Y-R)

University of Kentucky

A Sneaky Way To Test It...

- E.g., MIT "Video Magnification" work provides their input test and reprocessed videos
- Videos were played on a monitor as a loop at a selected framerate (thus controlling BPM)
- PowerShot pointed at the screen, BPM read from its display

University of Kentucky



University of Kentucky

Results (Using Video Playback)

- Before detrending and averaged selection, only about 1/3 of runs settled on the correct value – most latched at the maximum BPM
- Detrending brought that closer to ½
- Weighted-average selection alone seems to be much more stable
- Detrending + weighted-average selection can drop FPS significantly, but still works well

University of Kentucky

Conclusions and Future Work

- It is absolutely viable to do this!
- Testing with video playback allowed control over more parameters, helped tune algorithms
- Undergraduate senior project team improving this now, testing with live subjects
- PowerShots already tracks faces: multi-person BPM monitoring may also be viable

Aggregate.Org
UNBRIDLED COMPUTING

University of Kentucky