Microcomputer-based artificial vision support system for real-time image processing for camera-driven visual prostheses

Wolfgang Fink
California Institute of Technology
Division of Physics, Mathematics, and Astronomy
Visual and Autonomous Exploration Systems Research Laboratory
Pasadena, California 91125

and
University of Arizona
Department of Electrical and Computer Engineering
and
Department of Biomedical Engineering
Tucson, Arizona 85721

Cindy X. You
Mark A. Tarbell
California Institute of Technology
Division of Physics, Mathematics, and Astronomy
Visual and Autonomous Exploration Systems Research Laboratory
Pasadena, California 91125

Abstract. It is difficult to predict exactly what blind subjects with camera-driven visual prostheses (e.g., retinal implants) can perceive. Thus, it is prudent to offer them a wide variety of image processing filters and the capability to engage these filters repeatedly in any user-defined order to enhance their visual perception. To attain true portability, we employ a commercial off-the-shelf battery-powered general purpose Linux microprocessor platform to create the microcomputer-based artificial vision support system ($\mu$AVS$^2$) for real-time image processing. Truly standalone, $\mu$AVS$^2$ is smaller than a deck of playing cards, lightweight, fast, and equipped with USB, RS-232 and Ethernet interfaces. Image processing filters on $\mu$AVS$^2$ operate in a user-defined linear sequential-loop fashion, resulting in vastly reduced memory and CPU requirements during execution. $\mu$AVS$^2$ imports raw video frames from a USB or IP camera, performs image processing, and issues the processed data over an outbound Internet TCP/IP or RS-232 connection to the visual prosthesis system. Hence, $\mu$AVS$^2$ affords users of current and future visual prostheses independent mobility and the capability to customize the visual perception generated. Additionally, $\mu$AVS$^2$ can easily be reconfigured for other prosthetic systems. Testing of $\mu$AVS$^2$ with actual retinal implant carriers is envisioned in the near future. © 2010 Society of Photo-Optical Instrumentation Engineers. [DOI: 10.1117/1.3292012]

Keywords: portable artificial vision system; microprocessor; image processing; visual prostheses; smartphone; real-time image processing.

Paper 09245R received Jun. 11, 2009; revised manuscript received Oct. 26, 2009; accepted for publication Nov. 24, 2009; published online Jan. 25, 2010.

1 Introduction

Systems providing artificial vision are becoming a reality.\textsuperscript{1–23} In particular, (extraocular or intraocular) camera-driven epiretinal implants are being used in chronic patient trials already\textsuperscript{6,5} (Fig. 1). With blind subjects, it is difficult to predict exactly what they can perceive with such camera-driven visual prostheses, especially since they (currently) provide only tens of stimulating retinal electrodes, thereby allowing only for limited visual perception (pixelation). Thus it is important to offer them a wide variety of image processing filters and the capability to engage these filters repeatedly in any user-defined order to enhance their visual perception in daily life.

Image processing systems (Fig. 2), such as the artificial vision simulator (AVS)\textsuperscript{24–26} (Fig. 3), provide these capabilities and perform real-time (i.e., 30 fps) image processing and enhancement of digital camera image streams before they enter the visual prostheses. AVS, in particular, comprises numerous efficient image manipulation and processing modules, such as user-defined pixelation, contrast and brightness enhancement, grayscale equalization for luminance control under severe contrast and brightness conditions, user-defined grayscale levels for potential reduction of data volume transmitted to the visual prosthesis, blur algorithms, and edge detection.\textsuperscript{27,28} AVS provides the unique ability and flexibility for visual prosthesis carriers to further customize their individual visual perception afforded by their respective vision systems by actively manipulating parameters of individual image processing filters, even altering the sequence of these filters.

The original implementation of the AVS processing system has been laptop computer based and hence not truly portable. To provide independent mobility for the blind and visually impaired using camera-driven artificial visual prostheses (Fig. 1), we introduce in the following a stand-alone, battery-powered, portable microcomputing platform and software system for real-time image processing for camera-driven artificial vision prostheses, the microcomputer-based artificial vision support system ($\mu$AVS$^2$).\textsuperscript{29}
2 Methods

2.1 Hardware Platform

For the purpose of creating a small, stand-alone, battery-operated, and portable version of AVS, namely μAVS², we employed a commercial off-the-shelf, battery-powered, general purpose miniaturized Linux processing platform: the Gumstix™ by Gumstix, Incorporated, Portola Valley, California microcomputer environment. It is lightweight (8 g), fast (600-MHz clock speed or better), and equipped with USB, Ethernet, and RS-232 interfaces.

In particular, the Verdex-Pro class Gumstix microcomputer has the following specifications (see http://www.gumstix.net/Hardware/view/Hardware-Specifications/Verdex-Pro-Specifications/112.html):

- Marvell® PXA270 CPU with XScale™
- 600-MHz clock speed
- Ethernet
- USB
- RS-232
- dynamic RAM+Flash RAM
- 80 × 20 mm, 8 g, −25 to 85 °C, 3.5 to 5.0 VDC
- battery powered (includes driving USB cameras and onboard USB/Ethernet/RS-232 interfaces).

2.2 Microcomputer-Based Artificial Vision Support System Processing Architecture

For visual processing purposes, μAVS² imports raw video frames from a camera connected via its built-in USB port (or...
alternatively via its built-in Ethernet port, see Fig. 5. μAVS² then reduces the resolution of the video frames to match the pixel resolution of the patient’s visual prosthesis, i.e., pixelation or downsampling, see Fig. 5. μAVS² subsequently processes the downsampled video frames through user-selected image filters in a linear, sequential-loop fashion, resulting in vastly reduced memory and CPU requirements during execution, making the use of a microprocessor possible. The frequency and order of the image processing modules are determined via a predefined, user-selectable “control string” or script without recompilation of the image processing code (Fig. 5). μAVS² then issues the processed video frame data over an outbound connection via RS-232, wired Ethernet, or Internet to the visual prostheses system in real time (Fig. 5).

The control string or script defines the order and frequency that the image filters are applied to the input video frames. Each image filter in a specific sequence is applied to the results of the filter before it, in a net cascading effect. Thus, filter 1 is applied to the raw pixelated video frame; filter 2 is then applied to that result, and so on to the last filter, at which time the processed image is ready to be issued for stimulation of the prosthesis. This cycle repeats anew for each incoming video frame. Before use, the blind subject is tested and the specific sequence of filters that provides him the best results is determined. This sequence is incorporated into the control script, which is then downloaded onto the device. In the current implementation of μAVS², the control script is not changeable by the user; however, implementing μAVS² on an advanced embedded platform such as an Apple iPhone could allow the user the selection of a number of prepared control scripts in real time (e.g., one for daytime, one for nighttime, etc.).

In particular μAVS² adheres to the following modular processing architecture (Fig. 5).

Stage 1: camera image acquisition. μAVS² utilizes access to an “image generator,” i.e., a digital video camera. The camera may be local, directly connected via the built-in USB port, or it may be a local or remote IP camera on an accessible network. For a locally connected (hardwired) camera, μAVS² will open an exclusive read channel to the device over the USB bus to gain access to the camera’s video buffer. For an IP camera, μAVS² will access the camera at its IP address, using the access protocol specific to the type of camera in use (e.g., HTTP, UDP).

Stage 2: image capture. Once μAVS² has established a connection to a digital video camera, it must extract discrete image frames from the camera. It does this by reading the camera’s YUV frame buffer.

Stage 3: custom image processing including pixelation. The standardized image representation is reduced in resolution (pixelated) to match the pixel resolution of the patient’s retinal implant electrode array. For example, if the original image representation has a resolution of 640 × 480 raw pixels, the image is downsampled to the patient’s electrode array size, e.g., 16 × 16 pixels, thus becoming a 256-byte pixel array. This downsampling affords μAVS² its real time capability, as it allows for a subsequent dramatic speed-up of otherwise computationally expensive image processing filters that can now be equally efficiently executed on the reduced-size pixel array rather than the full-resolution camera video frames (for more detail see Sec. 2.4). Specialized image processing...
filters (e.g., contrast, brightness, gray equalization, gray levels, inversion, edge detection, and image blur\textsuperscript{27,28}) are then applied to the resolution-reduced/pixelated image pixel array according to a predefined/user-selectable control string or script to enhance its essential characteristics, cleaning up the image before issuance to the retinal implant.

Stage 4: image data transmission. The final stage in $\mu$AVS\textsuperscript{2} processing causes the processed pixel array to be sent to a power and data telemetry system (Fig. 5) that subsequently outputs the image data, e.g., by means of electrical current pulses on the patient’s retinal implant electrode array to generate visual perceptions, i.e., so-called phosphenes.

2.3 Microcomputer-Based Artificial Vision Support System Connectivity

$\mu$AVS\textsuperscript{2} supports a variety of interfaces that enable its connectivity to input devices such as digital cameras, and to output devices such as artificial vision systems. The interfaces currently supported are as follows.

2.3.1 Universal Serial Bus Camera Interface

$\mu$AVS\textsuperscript{2} contains a fully supported, general purpose USB port, allowing for the connection of a wide range of USB devices, including many types of USB web cameras, without the need of an external power supply. Alternatively, accessing an IP camera is possible as well, using an HTTP protocol to an IP camera connected to the onboard Ethernet port. When using a YUV-capable USB or IP camera, the camera’s grayscale video frame buffer is directly accessible to the $\mu$AVS\textsuperscript{2} image processing filters.

2.3.2 Ethernet-Based (Transmission Control Protocol/Internet Protocol) Data Exchange Protocol

To integrate the $\mu$AVS\textsuperscript{2} with artificial vision prostheses, an extensible Ethernet-based (i.e., TCP/IP) data exchange protocol (Fig. 6) has been developed (for details see Ref. 26) to facilitate the transfer of the $\mu$AVS\textsuperscript{2}-manipulated (video) output data to, e.g., power and data telemetry systems for retinal prostheses.\textsuperscript{30} This Ethernet-based protocol allows for maximum flexibility between systems, and is capable of two-way data transfer as well as status message transmission. The interface protocol is sufficiently flexible, so that it is also possible for a power and data telemetry system to transmit its own data (e.g., measurement data obtained from the vision implant) back to $\mu$AVS\textsuperscript{2} for feedback control, further processing, and analysis. Additionally, contingency measures are integrated into the protocol, providing for negative acknowledgements and data resend requests, should the need arise in which data is required to be resent.

2.3.3 RS232 Communication Protocol

Using the onboard universal asynchronous receiver/transmitter, it is possible to write the $\mu$AVS\textsuperscript{2}-manipulated (video) output data to the Gumstix serial port at speeds of up to 921,600 baud, using a RS-232 8N1 format: each data byte is actually nine bits long, i.e., eight data bits and one stop bit.

2.4 Microcomputer-Based Artificial Vision Support System Real-Time Image Processing Capability

The real-time image processing capability of $\mu$AVS\textsuperscript{2} is enabled by the downsampling of the native resolution of the artificial vision prosthesis camera system before image processing filters are applied. For example, if the original image representation has a resolution of $640 \times 480$ raw pixels (i.e., 307,200 pixels), and if the implanted electrode array of an epiretinal vision implant has a dimension of $16 \times 16$ stimulating electrodes (i.e., 256 electrodes), then the downsampling/binning of the $640 \times 480$ pixel image to $16 \times 16$ pixels affords a speed-up factor of 1200 (i.e., 3 orders of magnitude) in subsequent filter processing. This process allows otherwise computationally expensive image processing filters (e.g., blur-
ring) to be executed in real time (i.e., in excess of 30 fps) without loss of filter validity/efficiency. In the Appendix in Sec. 5 we show mathematically and numerically, for a set of relevant image processing filters, that downsampling first and subsequent filtering of the reduced size image yields the exact or nearly exact same end result (i.e., is commutable or nearly commutable) compared to downsampling after image processing filters have been applied to the full-resolution camera video frames. This speed-up enables the employment of computationally low power, portable, and battery-powered microcomputing platforms to perform real-time image processing within an artificial vision system, thereby affording users enhanced mobility and independence.

### 3 Results

Table 1 demonstrates the binning/downsampling performance efficiency of μAVS² for various artificial vision prostheses with electrode array dimensions ranging from 4×4 to 32×32. For example, μAVS² is capable of binning an in-memory source image (i.e., a frame) of 160×120 pixels down to 8×8 pixels at a rate of 354 frames per second (fps).

Table 2 shows individual and total filter performance efficiencies for various electrode array dimensions, obtained with an in-memory frame of 160×120 pixels. The results show that the application of filters to the workflow impact the processing only marginally. For example, adding a contrast filter to the 8×8 case reduces the frame rate by only 3, from 354 (no filters) to 351 fps. This demonstrates that binning/downsampling represents the bulk of the processing, whereas the individual filters (or even a sequence of all listed filters) pose a minimal additional processing burden, i.e., the image processing filters are “lightweight.” In practice, the real-time frame rate is much more dependent on the maximum frame rate of the camera used, some of which are limited to 10 fps, such as the camera used for our development.

As far as the actual CPU utilization of the Gumstix is concerned, using a camera frame resolution of 160×120 pixels with a sustained frame rate of 10 fps (including the bidirectional communication between μAVS² and the camera to generate the YUV frame buffer) results in only a 10% CPU load (measured using the Unix “top” command). In practice, the CPU load is linearly dependent on the camera frame rate, theoretically allowing for a maximum of 100 fps at a camera frame resolution of 160×120 pixels, whereas many implanted prostheses require only 60 or fewer frames per second for optimal functioning.

Figure 7 displays application of the previous filters to a typical scenario (i.e., a darkened hallway). Here the simulated electrode array dimensions of the artificial vision prostheses (i.e., binning) are 32×32.

Furthermore, duration tests with two common battery types were performed. The first test used a single 6-V 2CR5 alkaline battery, resulting in an average duration of 2.7 h. The second test used a NiMH 6-V×5-Ah rechargeable battery pack, resulting in an average duration of 12 h.

### Table 1

<table>
<thead>
<tr>
<th>Camera frame resolution</th>
<th>Electrode array dimension</th>
<th>160×120 pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4×4 pixels</td>
<td>8×8 pixels</td>
</tr>
<tr>
<td>Binning performance</td>
<td>370</td>
<td>354</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>Binning dimension</th>
<th>4×4 pixels</th>
<th>8×8 pixels</th>
<th>16×16 pixels</th>
<th>32×32 pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contrast filter</td>
<td>369</td>
<td>351</td>
<td>296</td>
<td>244</td>
</tr>
<tr>
<td>Brightness filter</td>
<td>368</td>
<td>349</td>
<td>293</td>
<td>233</td>
</tr>
<tr>
<td>Gray equalization filter</td>
<td>300</td>
<td>288</td>
<td>253</td>
<td>221</td>
</tr>
<tr>
<td>Gray level filter</td>
<td>370</td>
<td>352</td>
<td>301</td>
<td>256</td>
</tr>
<tr>
<td>Inversion filter</td>
<td>371</td>
<td>353</td>
<td>302</td>
<td>262</td>
</tr>
<tr>
<td>Edge detection filter</td>
<td>368</td>
<td>341</td>
<td>266</td>
<td>170</td>
</tr>
<tr>
<td>Image blur filter</td>
<td>367</td>
<td>349</td>
<td>293</td>
<td>236</td>
</tr>
<tr>
<td>All filters engaged</td>
<td>302</td>
<td>280</td>
<td>217</td>
<td>137</td>
</tr>
</tbody>
</table>
The Gumstix natively supports a multiboot versatility. If a microSD memory card is inserted into the Gumstix’s card slot, the Gumstix will boot from the system installed on it instead of booting from its own internal flash RAM. Thanks to this capability, we have been able to provide several custom-tailored image filter cascades, each on its own microSD card, such that a change from one setting to another can easily be accomplished by swapping cards. On reboot, μAVS2 automatically executes the new processing cascade established on the microSD card without further user interaction, and continuously performs real-time image processing.

4 Conclusion

To support existing and future camera-driven artificial vision prostheses, we have devised the microcomputer-based artificial vision support system (μAVS2): a small, stand-alone, battery-operated, customizable image processing system that utilizes a general purpose miniaturized Linux processing platform. The μAVS2 provides real-time image processing for artificial vision systems while maintaining portability and thus independence for its users. Moreover, with its general purpose computing capabilities, it can easily be reconfigured to support prosthetic systems beyond artificial vision, such as control of artificial limbs.

Multi-purpose systems, such as smartphones (e.g., Apple iPhone, BlackBerry, or Android phones), would be ideal platforms to host μAVS2, as they provide a cell phone link and integrated GPS functionality, in addition to image processing capabilities. It should be noted that at the time of this writing, the application programming interface (API) of the iPhone software development kit (SDK) did not provide for live camera stream access. Such support is anticipated with the next SDK update. Nevertheless, we have proceeded to port a static version of μAVS2 to the iPhone, applying the suite of filters to still-image manipulation and storage functionalities. This will be updated to perform real-time video processing on the iPhone as soon as its SDK supports live camera stream access.

We would like to emphasize that the employment of μAVS2 in visual prosthetics is by no means limited to retinal implants (epi- or subretinal) only. On the contrary, μAVS2 is directly and immediately applicable to any (artificial) vision-providing/stimulating system that is based on an external (e.g., eyeglass-mounted) or internal (e.g., intraocular camera) video-camera system as the first step in the stimulation/processing cascade, such as optic nerve implants,10–12 cortical implants,13–16 electric tongue stimulators,17–20 and tactile stimulators (both electrical and mechanical21–23). In addition, μAVS2 can interface to infrared (IR) camera systems to augment the visual cues with thermal information, allowing for “supervision” at night and during adverse weather conditions such as fog.

Acknowledgments

The work described in this publication was carried out at the California Institute of Technology under support of the National Science Foundation grant EEC-0310723. Fink and Tarbell may have proprietary interest in the technology presented here as a provisional patent has been filed on behalf of Caltech. You has no proprietary interest.

Appendix

In the following we show mathematically and numerically, for a set of relevant image processing filters, that downsampling first and subsequent filtering of the reduced size image yields the exact or nearly exact same end result compared to downsampling after image processing filters have been applied to the full-resolution camera video frames. The degree of computation between both procedures for the respective image processing filters is reported in the following in units of gray value differences.

1 General Definitions for Image Processing Filter Proofs

In the following, all divisions in \([a/b]\) form denote integer divisions, and all divisions not enclosed within brackets represent regular divisions. All \(q\) (quotients, with \(0 \leq q \leq 255\) as applied to filters operating on 256 grayscale values) and \(r\) (respective remainders) are \(\in \mathbb{N}\). Note that in general, for any integer division \([a/b]=q_\ast\), with corresponding remainder \(r_\ast\), we can eliminate the integer division notation via

\[ q_\ast = \frac{a}{b} = \frac{a - r_\ast}{b}, \quad 0 \leq r_\ast < b. \]

Let \(x_i\), \(0 \leq x_i \leq 255\), denote the gray value of the pixel to be filtered, \(f(x_i)\) be the image processing filter applied to the pixel, and \(n\) denote the number of pixels to be processed into one superpixel, i.e., downsampling. The downsampling is as follows:
Let \( a, b, c \neq 0 \), and \( \epsilon \mathbb{Z} \), but \( b/c \) is not an integer. Then \( [a + b/c] \) can be rewritten by:

\[
\begin{align*}
\text{If } a > 0 & \quad b/c > 0, \quad \text{then } a + \left\lfloor \frac{b}{c} \right\rfloor \\
a < 0 & \quad b/c < 0, \quad a + \left\lfloor \frac{b}{c} \right\rfloor - 1 \\
a > 0 & \quad b/c < 0, \quad a + \left\lfloor \frac{b}{c} \right\rfloor - 1 \\
a < 0 & \quad b/c > 0, \quad a + \left\lfloor \frac{b}{c} \right\rfloor + 1.
\end{align*}
\]

With the previous definitions, the downsampling first versus filtering first procedures are:

Procedure 1: downsample first, filter second.

\[
f\left( \left\lfloor \frac{\sum_{i=1}^{n} x_i}{n} \right\rfloor \right) = f(\bar{x}).
\]

Procedure 2: filter first, downsample second.

\[
\left\lfloor \frac{\sum_{i=1}^{n} f(x_i)}{n} \right\rfloor = f(\bar{x}).
\]

2.1 Type 1: Additive Filters

Filter description

\[
f(x_i) = x_i + a, \quad a \text{ is a constant } \in \mathbb{Z}.
\]

Evaluation

Procedure 1:

\[
f(\bar{x}) = \bar{x} + a.
\]

Procedure 2:

since \( r_1/n < 1 \), and \( a, q_1 \) are both integers,

\[
f(x) = \left\lfloor \frac{\sum_{i=1}^{n} (x_i + a)}{n} \right\rfloor = \bar{x} + \left\lfloor \frac{r_1}{n} + a \right\rfloor.
\]

Comparison

\[f(x) = \bar{x} + a = f(\bar{x}).\]

The order of downsample and filtering is inconsequential with respect to the resulting gray value. Both procedures are commutative.

2.2 Type 2: Multiplicative Filters

Filter description

\[
f(x_i) = [ax_i], \quad a = \frac{b}{d} \text{ is a rational constant } \in \mathbb{R} > 0.
\]

Evaluation

Procedure 1: let \( r_2 \) be the remainder of \( f(\bar{x}) = [a\bar{x}] \), such that

\[
f(\bar{x}) = \left\lfloor \frac{\frac{b}{d}\left(\sum_{i=1}^{n} x_i - r_1\right)}{n} \right\rfloor,
\]

\[
\frac{b}{d}\left(\sum_{i=1}^{n} x_i - r_1\right) - r_2 = \frac{b}{d}\left(\sum_{i=1}^{n} x_i\right) - \frac{r_1 - r_2}{n}.
\]

Procedure 2: let \( r_3 \) be the remainder of \( [bx_i/d] \), \( 0 \leq r_3 < d \), such that

\[
f(\bar{x}) = \left\lfloor \frac{\sum_{i=1}^{n} \left(\frac{b}{d}(x_i - r_1)\right)}{n} \right\rfloor = \left\lfloor \frac{\sum_{i=1}^{n} \left(\frac{b}{d}x_i - \frac{r_1}{d}\right)}{n} \right\rfloor.
\]

with \( r_3 \), \( 0 \leq r_3 < n \), being the remainder of the prior division, it follows

\[
f(\bar{x}) = \left\lfloor \frac{\sum_{i=1}^{n} \left(\frac{b}{d}x_i - \frac{r_1}{d}\right)}{n} \right\rfloor = \frac{r_3}{n}.
\]

Comparison

The two procedures differ by
The greatest difference with respect to the resulting gray values between the two procedures is
\[
\left( \frac{\sum_{i=1}^{n} r_i}{dn} + \frac{r_3}{n} \right) - \left( \frac{f_1 + r_3}{n} \right) < 2.
\]

Both procedures are nearly commutable.

2.3 Type 3: Inversion Filter

Filter description

\[ f(x_i) = 255 - x_i. \]

Evaluation

Procedure 1:

\[ f(\bar{x}) = 255 - \bar{x}. \]

Procedure 2:

\[
f(x) = \left\lfloor \frac{\sum_{i=1}^{n} (255 - x_i)}{n} \right\rfloor, \]

\[
= \left\lfloor \frac{\sum_{i=1}^{n} x_i}{n} \right\rfloor - 1, \]

\[
= 255 - \bar{x} - 1. \]

Comparison

\[ f(\bar{x}) = 255 - \bar{x} = f(x) + 1. \]

Therefore, the respective gray values of the two procedures differ by a constant value of 1. Hence, they are nearly commutable.

2.4 Type 4: Gray-Value Reduction Filter

Filter description

Let \( v \) denote the new number of gray levels. For practical purposes, \( 1 < v < 256, v \in \mathbb{N} \). Let \( c \) be the number of original grayscale values incorporated into each new grayscale value, and \( \Delta \) be the interval between gray values of the new grayscale.

\[
c = \left\lfloor \frac{256}{v} \right\rfloor, \quad \Delta = \left\lfloor \frac{255}{v - 1} \right\rfloor, \quad 1 \leq c \leq \Delta \leq 2c. \]

The gray-value reduction filter function then reads:

\[
f(x) = \left\lfloor \frac{x}{c} \right\rfloor \Delta. \]

Evaluation

Procedure 1: let \( r_4 \) be the remainder of \( \bar{x}/c, 0 \leq r_4 < c \). Recall that \( \bar{x}=\left(\sum_{i=1}^{n}x_i\right)/n \). Hence, Procedure 1 yields:

\[
f(\bar{x}) = \left\lfloor \frac{\left(\sum_{i=1}^{n}x_i\right) - r_1}{n} \right\rfloor - \frac{r_4}{c}, \Delta, \]

\[
= \frac{\Delta \sum_{i=1}^{n} x_i}{cn} - \frac{\Delta r_4}{cn} - \frac{r_4}{c}. \]

Procedure 2: let \( r_5 \) be the remainder of \( [x_i/c], 0 \leq r_5 < v \), and \( r_5 \) be the remainder of \( \left[\sum_{i=1}^{n}x_i/c\right]/\Delta/n \), \( 0 \leq r_5 < v \).

\[
f(x) = \left\lfloor \frac{n}{c} \right\rfloor \Delta - \frac{r_5}{n}, \]

\[
= \frac{\sum_{j=1}^{n} \left(\frac{x_j}{c}\right) \Delta - r_5}{n}, \]

\[
= \frac{\sum_{j=1}^{n} \left(\frac{x_j - r_1}{c}\right) \Delta - r_5}{n}, \]

\[
= \frac{\Delta \sum_{i=1}^{n} x_i - \Delta \sum_{i=1}^{n} r_i}{cn} - \frac{r_5}{n}. \]
\[
\frac{\Delta r_1}{cn} + \frac{\Delta r_4}{c} = \left( \frac{\Delta \sum_{i=1}^{n} r_i}{cn} + \frac{r_5}{n} \right).
\]

Recall that \(0 \leq r_1 < n\), \(0 \leq r_4 < c\), \(0 \leq r_5 < n\), \(1 \leq c \leq \Delta \leq 2c\).

\[
0 \leq \left( \frac{\Delta r_1}{cn} + \frac{\Delta r_4}{c} \right) < \left( \frac{\Delta}{c} + \Delta \right) < 2 + 2c,
\]

\[
0 \leq \left( \frac{\Delta \sum_{i=1}^{n} r_i}{cn} + \frac{r_5}{n} \right) < \left( \frac{\Delta \sum_{i=1}^{n} c}{cn} + 1 \right) = \Delta + 1 < 2c + 1.
\]

Therefore, since

\[
0 \leq \left( \frac{\Delta r_1}{cn} + \frac{r_4}{c} \right) < 2 + 2c, \quad 0 \leq \left( \frac{\sum_{i=1}^{n} r_i}{cn} + \frac{r_5}{n} \right) < 2c + 1,
\]

\[
\frac{\Delta r_1}{cn} + \frac{r_4}{c} < 2c + 2c.
\]

Thus, the greatest difference between the resulting gray values of the two procedures is \(2c + 2\). Hence, the lower the number of new gray values, the larger the deviation of Resulting gray values, and vice versa. Thus, the degree of commutation between both procedures depends on the number of new gray values.

\((*)\) Proof of \(1 \leq c \leq \Delta \leq 2c\)

Consider \(c = \lceil 256/v \rceil = 1\), \(\Delta = \lceil 256/(v-1) \rceil = \lfloor (255v/v) - 1 \rfloor\).

With \(1 < v < 256\), \(v \in \mathbb{N}\), the numerator of \(\Delta\) can take on values

\[
\frac{255^2}{254} \leq \frac{255v}{v-1} \leq \frac{2552}{1}.
\]

Note the left side of this inequality is \(>256\) but \(<257\) and \(v = 255\) when the right side corresponds to \(v = 2\), and is equal to 510. To the left side of this series of inequalities, we can then tag

\[
256 < \frac{255^2}{254} \leq \frac{255v}{v-1} \leq \frac{2552}{1},
\]

which then implies that \(c < \Delta\).

Furthermore, \(2c = \lceil 256/v \rceil = \lceil 2*256/v \rceil - 1 = \lfloor 512 - v/v \rfloor\).

The smallest possible value for the numerator of \(2c\) is \(512 - v = 512 - 2 = 510 = 255 * 2/1\). Hence, \(\Delta \leq 2c\).

2.5 Type 5: Blur Filter

For the blurring, we developed and implemented a computationally inexpensive code that consists of two subsequent sweeps across the respective image (downsampled or full resolution): first a vertical sweep (i.e., by columns) followed by a horizontal sweep (i.e., by rows). In both sweeps only NN = 1 horizontal or vertical nearest neighbors, respectively, are considered. The nearest neighbors are equally weighted. This blur procedure delivers an effective blur (e.g., Fig. 7) and is applied in lieu of computationally expensive Gaussian blur filters 27,28. We randomly generated \(320 \times 320\) pixel images, then proceeded to: 1. blur the image once with the number of nearest neighbors \(NN = 1\), and downsample the image to either \(4 \times 4\), \(8 \times 8\), \(16 \times 16\), or \(32 \times 32\); or 2. downsample the image to either \(4 \times 4\), \(8 \times 8\), \(16 \times 16\), or \(32 \times 32\), then blur it. Results between the same down sampling dimensions of the two procedures were compared, and the respective average deviation per pixel was recorded. The numerical simulations revealed that the deviation between the two procedures is minimal (i.e., they are nearly commutable): worst average deviation was \(4.4 \pm 3.4\) on \(32 \times 32\) downsampled image, and best average deviation was \(0.58 \pm 0.58\) on a \(4 \times 4\) downsampled image. 10,000 simulation runs were conducted, respectively.

2.6 Type 6: Gray Equalization Filter

For the gray equalization we employed the filter described in Refs. 27 and 28. We randomly generated \(320 \times 320\) pixel images, then proceeded to: 1. equalize the image once with slope = 0, and downsample the image to either \(4 \times 4\), \(8 \times 8\), \(16 \times 16\), or \(32 \times 32\); or 2. downsample the image to either \(4 \times 4\), \(8 \times 8\), \(16 \times 16\), or \(32 \times 32\), then equalize it (with slope = 0, see Refs. 27 and 28). Results between the same down sampling dimensions of the two procedures were compared, and the respective average deviation per pixel was recorded. Average deviations were obtained via averaging the differences between corresponding pixels in images produced through first equalization then binning and first binning then equalization. Increased binning results in increased deviation: \(32 \times 32\) resulted in an average deviation of \(58 \pm 33\), \(16 \times 16\) resulted in \(61 \pm 35\), \(8 \times 8\) resulted in \(64 \pm 37\), and \(4 \times 4\) results in \(72 \pm 34\). 10,000 simulation runs were conducted, respectively. The numerical simulations revealed that large downsampling (e.g., \(4 \times 4\) after equalization merely centered pixel gray values around the middle of the grayscale (i.e., 128), suggesting that it is more effective to perform gray equalization after downsampling to attain meaningful equalization effects, as it utilizes the entire grayscale range of 0 to 255.
26. W. Fink, M. A. Tarbell, L. Hoang, and W. Liu, “Artificial vision support system (AVS2) for enhancing and optimizing low-resolution visual perception for visual prostheses,” (to be submitted).