Phase-sensitive detection in the undergraduate lab using a low-cost microcontroller

K. D. Schultz

Citation: American Journal of Physics **84**, 557 (2016); doi: 10.1119/1.4953341
View online: https://doi.org/10.1119/1.4953341
View Table of Contents: http://aapt.scitation.org/toc/ajp/84/7
Published by the American Association of Physics Teachers

APPARATUS AND DEMONSTRATION NOTES

The downloaded PDF for any Note in this section contains all the Notes in this section.

Frank L. H. Wolfs, *Editor*
Department of Physics and Astronomy, University of Rochester, Rochester, New York 14627

This department welcomes brief communications reporting new demonstrations, laboratory equipment, techniques, or materials of interest to teachers of physics. Notes on new applications of older apparatus, measurements supplementing data supplied by manufacturers, information which, while not new, is not generally known, procurement information, and news about apparatus under development may be suitable for publication in this section. Neither the *American Journal of Physics* nor the Editors assume responsibility for the correctness of the information presented.

Manuscripts should be submitted using the web-based system that can be accessed via the *American Journal of Physics* home page, http://ajp.dickinson.edu and will be forwarded to the ADN editor for consideration.

Phase-sensitive detection in the undergraduate lab using a low-cost microcontroller

K. D. Schultz[a)]
Hartwick College, Oneonta, New York 13820

(Received 25 January 2015; accepted 23 May 2016)

Phase-sensitive detection is an important experimental technique that allows signals to be extracted from noisy data. Commercial lock-in amplifiers, often used for phase-sensitive detection, are expensive and host a bewildering array of controls that may intimidate a novice user. Low-cost microcontrollers such as the Arduino family of devices might seem like a good match for learning about such devices, but making a self-contained device that includes a reference signal, a voltage input, a signal mixer, a filter, and a display is difficult. Here, we present the construction of a phase-sensitive detector (PSD) using an Arduino.
[http://dx.doi.org/10.1119/1.4953341]

I. INTRODUCTION

Lock-in amplification and phase-sensitive detection are important techniques in experimental physics, and there have been many papers describing the pedagogical uses of these techniques.[1–3] Commercial devices are expensive and can be intimidating for new users. Building a home-made instrument can be instructive;[4] however, doing so requires advanced electronics skills that a student may not already have, putting the emphasis on the electronics and not the method. Conversely, it is possible to perform the mixing and filtering on a computer using a computer's sound card or other low-cost data acquisition devices to handle the input and output.[5]

In this paper, we describe a phase-sensitive detector (PSD) that uses the popular Arduino microcontroller and the Processing programming environment.[6] The major design goal was to make the device as self-contained as possible, a task made difficult by the memory and hardware constraints of a typical microcontroller. While these devices are fantastic for controlling robots and basic data-logging, turning them into scientific instruments requires techniques that go beyond what is normally found in the literature. An added benefit is that these techniques can be used to build other types of instrumentation such as arbitrary function generators and fast digital multimeters (DMMs) out of a low-cost microcontroller that typically costs just a few tens of dollars.

II. ARDUINO

Arduino[7,8] is a catch-all term for a family of open-source hardware based on Atmel micro-controllers with a pre-loaded bootloader, allowing the user to program the Arduino with the simpler Arduino language. Arduino programs, which are called *sketches* in the Arduino environment, are based on C and C++, but with a simpler instruction set. If one wishes, the more expansive AVR instruction set can be used.[7,8] Programming and communication can be done via USB or through a set of on-board communication pins. There have been sixteen different Arduino-labeled boards produced at the time of writing, each of which has its own unique hardware and memory specifications. In this paper, the Arduino Uno R3 is used, simply because it was what was on hand and is one of the cheaper and more basic Arduinos. The heart of the Uno R3 is the Atmel ATmega328 microcontroller.[9] The Uno R3 has 14 digital input/output pins, six of which provide Pulse Width Modulation (PWM) output. This unit also has six analog inputs and an on-board 16-MHz oscillator. The ATmega 328 has 32 kB of Flash memory to store programs and 2 kB of SRAM for variable storage. The program given in this paper is easily stored in the flash memory, but the limited SRAM places severe restrictions on the amount of data that can be taken and manipulated on-board the Arduino.

III. PHASE SENSITIVE DETECTION

A block diagram for phase-sensitive detection is shown in Fig. 1. The input stage of the PSD consists of two signals,

the input of interest and the reference signal. After the signals are acquired, the input signal may be amplified and filtered. After the input stage comes the mixing stage where the input and reference signals are combined. There are a number of ways of doing this, but in this implementation the signals will simply be multiplied. Finally, the output of the mixing stage is heavily filtered by a low-pass filter. Once again this can be done using analog circuitry, but here it will be done mathematically.

A. Generic phase sensitive detection

Let us assume that the input signal V_i and the reference signal V_r are given by

$$V_i = V_1 + V_1 \sin(\omega_i t - \phi_i),$$
$$V_r = V_2 \sin(\omega_r t - \phi_r). \quad (1)$$

Here, V_i has a dc offset because the Arduino, like most microcontrollers, operates only with positive voltages. During mixing the offset for V_r is removed and therefore is not included in the following derivation. Upon multiplication of the two signals in Eq. (1), and making use of a trigonometric identity, the output of the mixing stage is

$$V_{\text{mix}} = V_1 V_2 \sin(\omega_r t - \phi_r) + \frac{V_1 V_2}{2}\Big\{\cos\big[(\omega_r - \omega_i)t - (\phi_r - \phi_i)\big] - \cos\big[(\omega_r + \omega_i)t - (\phi_r + \phi_i)\big]\Big\}. \quad (2)$$

At this stage Eq. (2) is nothing more than what we find in heterodyne detection in radio and optical engineering.[10] The power of coherent detection is that small signals get amplified by a larger local oscillator signal, as is evident in Eq. (2). Upon mixing of the two signals, there are contributions to the signal at the original frequencies and at the sum and difference frequencies. This is the heart of PSD in that the output filter is set to reject all frequencies other than the difference frequency of the inputs. Typically in lock-in detection V_i and V_r are made to oscillate at the same frequency before entering the PSD. With this constraint the output V_o of the PSD simply depends on the phase difference between the two signals

$$V_o = \frac{V_1 V_2}{2}\cos(\phi_r - \phi_i). \quad (3)$$

The final effect of this filtering is to move our output from $\omega_{r,i}$ to dc (the aforementioned radio and optical engineers would call this homodyne detection). The stronger the filtering, the more noise is rejected so the signal-to-noise increases. However, in effect, the PSD is performing signal averaging, so with each factor of two increase in the signal-to-noise the collection time increases by a factor of four. The other disadvantage of a PSD is that by making the measurements at dc we are placing our signal where $1/f$-noise dominates.[10] Even though we have added no active amplification, the presence of a strong local oscillator can boost a weak experimental signal.

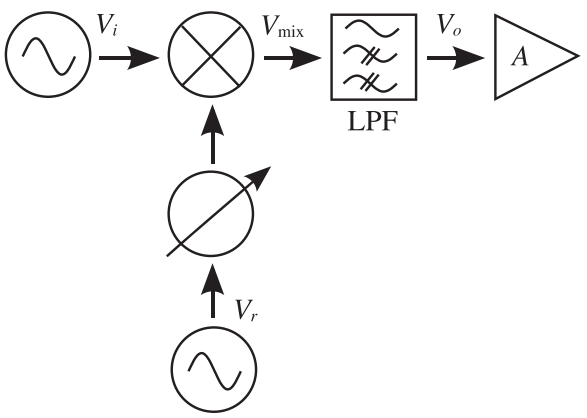

Fig. 1. Block diagram of a traditional phase sensitive detector; V_i and V_r are the input and reference voltages, respectively, LPF is a low-pass filter, and A is an amplifier. A phase-shifter is shown after V_r, but this component and the amplifier are not used in this project. See text for more details.

IV. IMPLEMENTATION

A self-contained phase-sensitive detector requiring only a computer for display, the Arduino, and as few passive circuit components as possible, places severe constraints and therefore requires some software and hardware exploits that are not commonly presented in introductions to microcontrollers. The structure of this section is to discuss the implementation of each of the sub-systems shown in Fig. 1. Briefly, and following Fig. 2, the Arduino synthesizes a sine wave output from a wavetable. As the output is updated from the wavetable, the input signal is quickly read. Finally, after the Arduino has cycled through the complete wavetable, mixing is performed mathematically onboard the Arduino, and the resulting waveforms are sent to the host computer running Processing for filtering and display.

A. Creating a reference signal

Creating the reference signal is in many ways the most significant constraint on the project. The Arduino does not have a true analog output. The only way to approximate an analog signal is to use the PWM pins of the Arduino. The internal clocks of the Arduino need to be manipulated such that the duty-cycle of the PWM is varied in such a way that with an external RC low-pass filter, a sine wave is created. The exact manipulations are discussed in what follows.

To generate a reference signal under these limitations a technique sometimes called "bit-banging"[11] is employed. Bit-banging takes advantage of the timers on the Arduino and PWM. In PWM, the duty-cycle (ratio of "on" to "off"

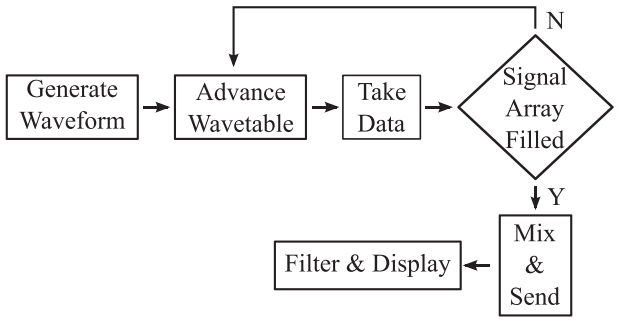

Fig. 2. All the steps through the mixing of the signals are performed on the Arduino; the filtering and display are accomplished in a Processing sketch.

times) of a square wave is modulated. The greater the duty-cycle, the longer the PWM pin on the Arduino is held HIGH. The value HIGH for an Arduino can take on different meanings depending on whether a pin is being used as an input or an output. Here HIGH or LOW refers to a digital write command to an Arduino pin, resulting in 0 V for LOW and 5 V or 3.3 V for HIGH, depending on the Arudino model. Passing that square-wave through a low-pass filter, effectively averaging the signal, produces a dc voltage that increases with the duty-cycle. The final step is to rapidly change this dc voltage so that the desired waveform, in this case a sine-wave, is synthesized.

This paper will only briefly sketch the relevant idea. For details, readers should refer to the comments in the source code for this project, the "bit-banging" paper,[11] or the ATmega328 datasheet.[9] The technique relies on two counters on the Arduino. Timer 1, TCNT1, is a 16-bit timer/counter that is configured as an 8-bit counter. TCNT1 runs at the full Arduino clock frequency (16 MHz) and produces the PWM output signal. Timer 2, TCNT2, runs slower than TCNT1 by a factor of eight and is used to step through the pre-generated wavetable representing the reference waveform. In what follows, the convention with "n" at the end of a name represents a generic counter, and if it is replaced by a number that will designate a specific counter.

Figure 3 shows the basic idea behind "bit-banging." Figure 3(a) shows the slow counter (TCNT2) while Fig. 3(b) shows the fast counter (TCNT1) before and after its Output Compare Registers (OCRnA) are changed. Finally, Fig. 3(c) shows the PWM output of the Arduino. When the value of the timer TCNTn matches the register OCRnA a flag is set. The outcome of the OCRnA flag can exhibit different behavior depending on how the Arduino is programmed. For the Arduino PSD, when the OCR1AL flag is set, the PWM pin is made to go LOW [Fig. 3(c)], but TCNT1 keeps incrementing until it overflows and TCNT1 starts counting again from zero [Fig. 3(b)]. At this point the PWM pin goes HIGH again, and the OCR1LA flag is reset. By changing the value of OCR1AL the duty-cycle of the PWM signal is changed, as illustrated for later times in Fig. 3, and consequently the dc voltage that is used to construct V_r is also changed. The pre-programmed wavetable for V_r is used to update OCR1AL. The Arduino is programmed to call an interrupt when the TCNT2 timer, running eight times slower than TCNT1, reaches the value in the register OCR2A. This interrupt does two things: updates OCR1AL to the next value of the wavetable, and gets an input voltage for the ADC. OCR2A is pre-set in the Arduino code such that the frequency of V_r is given by

$$f_r = \frac{\text{rate of TCNT2}}{\text{OCR2A} \times \text{wavetable length}}. \tag{4}$$

The last step in generating V_r is to place a simple low-pass *RC* filter on the PWM pin to get a smooth sine-wave.

B. Getting the signal in

The Arduino ADC is a 10-bit successive approximation circuit connected to an 8-channel multiplexer, although only six channels are used on the Arduino UNO. The measurements are single-ended and normally referenced to a 1.1 V on-board reference voltage, although it is possible to use an external reference voltage. Because interrupts are used to create an analog output for the reference signal, it is important to ensure that anything that happens during the interrupt is quick, which is why in Fig. 2 the data is sent for filtering after the wavetable has been completely cycled through. Serial calls are slow.

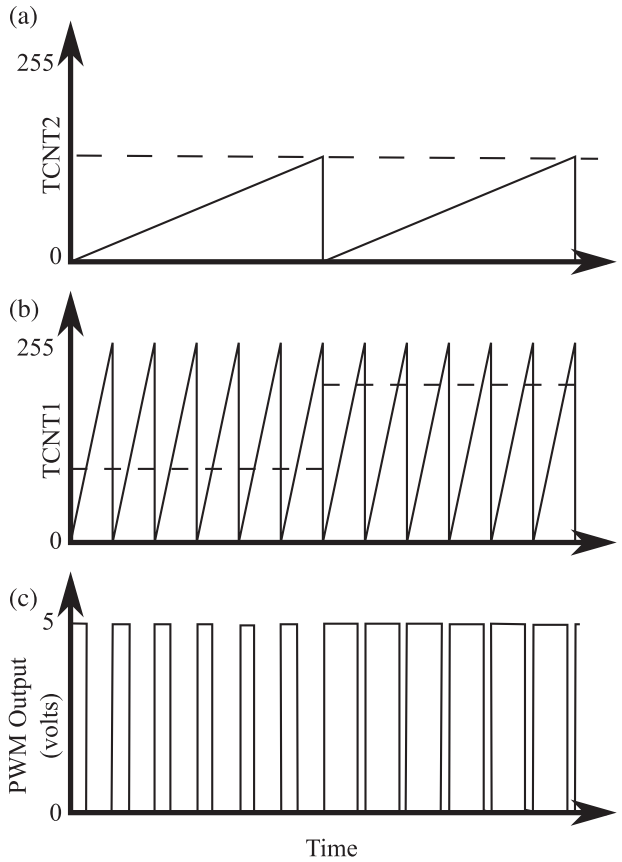

Fig. 3. The "bit-banging" technique requires a slow counter (a), a fast counter (b), and a pulse-width modulated signal (c). The dashed line in (a) represents OCR2A, the counter value that calls the necessary interrupt to update OCR1A and read in the input voltage for mixing. The dashed line in (b) represents the changing OCR1A, which sets the duty-cycle for the PWM. The reference signal that modulates the experiment is the output of a low-pass *RC* circuit with (c) as the input.

According to ATMEL, the ultimate sampling rate for single-ended measurements is limited by the ADC clock speed. The ADC clock speed is derived from the main system clock, and for maximum resolution should be between 50 and 200 kHz. However, for purposes of this project satisfactory resolution and accuracy are obtained by setting the ADC clock to a speed of 1 MHz. The successive approximation circuit requires 13 clock cycles, giving a sampling rate of 77 kHz,[12] which is much faster than this project needs.

C. Phase shifts and mixing

Because the values of the reference signal are pre-loaded into the Arduino memory, changing the phase while data is being taken is difficult. Often, however, the ability to shift the relative phase between V_i and V_r is needed to maximize the signal. Additionally, sometimes what is needed in a measurement is the phase shift between the reference and input signals. To meet these needs, a cue is taken from dual-phase lock-in amplifiers.[13] These amplifiers use a reference

waveform and a so-called quadrature reference signal, which is just the reference waveform shifted by 90°. Both the in-phase $V_r \cos(\omega_r t)$ and quadrature $V_r \sin(\omega_r t)$ reference waveforms are separately mixed with the signal input as discussed in Sec. III. The result is two mixed signals, the in-phase (I) and quadrature (Q) waveforms. It is the I and Q signals that are sent from the Arduino to the host computer for filtering and display. The phase ambiguity is removed by calculating the magnitude of the mixed signal $R = \sqrt{I^2 + Q^2}$. The phase difference between the input and reference signals is $\tan\phi = Q/I$.

D. Filtering and display

Once the Arduino finishes running through the waveform, it takes the collected data and sends it via USB to a computer running the Processing IDE.[14,15] Processing is a programming language and development environment that was designed to make it easier for the arts community to become software literate. Like the Arduino, it has a vibrant community that has produced numerous tutorials, examples, and books that make learning Processing relatively easy. Processing is also a direct forebear of the Arduino development system and therefore makes it a natural fit with the Arduino side of this project.

In this project, the Processing sketch (Processing-talk for program) initiates serial communications with the Arduino, and once communications are established the Arduino sends the I and Q data to the Processing sketch. The Processing sketch applies a recursive, single-pole, low-pass filter[16] to the array that is to be displayed (I, Q, R, ϕ). A single-pole recursive filter can be written as

$$y[n] = (1 - x_{\text{decay}})x[n] + x_{\text{decay}}y[n-1], \tag{5}$$

where $x_{\text{decay}} = e^{-2\pi f_c}$, y is the output of the filter, x is the input data, and f_c is the time constant of the filter. Mathematically, this filter is identical to a single-pole RC filter in electronics. The stronger the filtering, the better it is for lock-in detection, since inadequate filtering causes the output to oscillate at the reference frequency. Algorithms for calculating coefficients for higher-order filters with faster roll-off are also available.[16] These may be of interest because single-pole filters are limited to attenuations of 6 dB/decade.

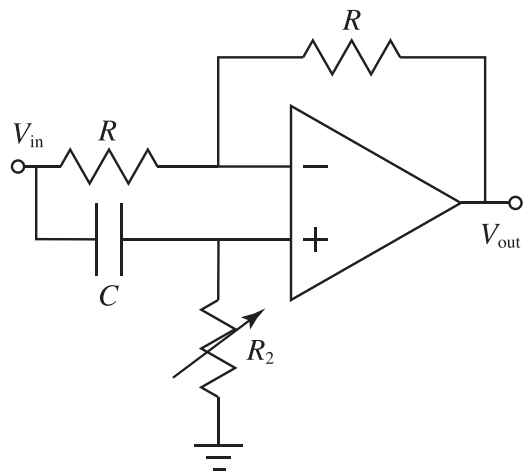

Fig. 4. Phase shifting circuit for testing the Arduino PSD. The input of this circuit is connected to the Arduino output corresponding to the reference signal. For testing, the output of this circuit was fed into the Arduino input used to capture V_s. The op-amp used was a standard 741. Changing R_2 shifts the phase through 180 degrees. The midpoint of the R_2 potentiometer and C are chosen such that $1/R_2C \sim \omega_r$. Sample component values are $R = 10$ $k\Omega$, $R_2 = 10$ $k\Omega$, and $C = 100$ nF.

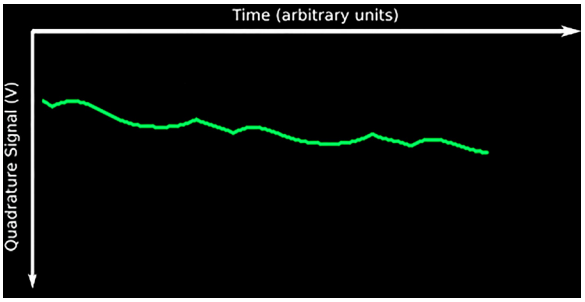

Fig. 5. A screenshot of the Processing output, the displayed axes were overlaid onto the Processing screenshot to aid understanding of this figure. The display can be set to show the phase, magnitude, I, or Q signal as a function of time. The display shows a numerical value of the quantity displayed as well as a graphical display. Here, the in-phase signal of the Arduino PSD is displayed as R_2 of the phase-shift test-circuit is varied. Future plans for this device are to make a more useful display.

V. TESTING

To test this project, an all-pass phase-shifter was built.[17] Figure 4 shows the circuit used to test the Arduino. The voltage gain for this circuit is 1 V/V, so the amplitude of the output is unchanged with respect to the input voltage. The high-pass RC filter at the non-inverting terminal of the op-amp controls the amount of phase-shift at the output. At $\omega_0 = 1/RC$ the phase-shift is 90° and changes by 90°/decade. By varying R_2, the 90° point shifts and moves our operating frequency along the phase plot. To test the Arduino PSD, the reference signal generated by the Arduino is sent to the input of the phase-shifting circuit. The output of the phase-shifting circuit becomes the input signal to the PSD. As mentioned in Sec. IV, the final output of the PSD is a graphical display of either I, Q, R, or ϕ as a function of time. At the time of writing this paper, the display capabilities of this project are very rudimentary. Nevertheless, Fig. 5 shows the I signal as displayed in the Processing sketch as R_2 is varied, demonstrating that the PSD is operating correctly.[18]

VI. CONCLUSIONS

A phase-sensitive detector using an Arduino microcontroller has been described. The main design goal of the project was to make the PSD as self-contained as possible, namely, that an external reference signal was not needed and that any phase corrections were unnecessary. Aside from the Arduino and a computer the only other hardware needed are a resistor and a capacitor. Furthermore, the techniques used for this project can be used in other projects involving using the Arduino microcontroller as a standalone, low-cost, scientific instrument. The "bit-banging" technique is not limited to sine-waves, but can be used to output any synthesized waveform that may be required. More complicated waveforms can be created outside of the Arduino environment and loaded into the EEPROM allowing faster execution and freeing up regular memory.[11]

ACKNOWLEDGMENTS

The author would like to thank Professor Lawrence Nienart of Hartwick College for his advice and encouragement throughout this project. The author would also like to thank the reviewers for helpful suggestions.

[a]Electronic mail: schultzk@harwick.edu. URL: www.HartwickChaosLab.github.io

[1]W. Yang, "Teaching phase-sensitive demodulation for signal conditioning to undergraduate students," Am. J. Phys. **78**, 909–915 (2010).
[2]K. G. Libbrecht, E. D. Black, and C. M. Hirata, "A basic lock-in amplifier experiment for the undergraduate laboratory," Am. J. Phys. **71**, 1208–1213 (2003).
[3]R. Wolfson, "The lock-in amplifier: A student experiment," Am. J. Phys. **59**, 569–572 (1991).
[4]P. Horowitz and W. Hill, *The Art of Electronics*, 2nd ed. (Cambridge U.P., Boston, 1989).
[5]M. González, G. Santiago, V. Slezak, and A. Peuriot, "Simple synchronic detection at audio frequencies through a PC sound card," Rev. Sci. Instrum. **78**, 055108-1–4 (2007).
[6]Up-to-date Arduino and Processing code can be found on my github page, which can be forked and modified. <github.com/HartwickChaosLab/Arduino-Phase-Sensitive-Detector>.
[7]Arduino Project Web Site; <http://www.arduino.cc>.
[8]M. Margolis, *Arduino Cookbook*, 2nd ed. (O'Reilly Media, Sebastopol, 2011).
[9]ATmega Data Sheet <http://www.atmel.com/devices/atmega328.aspx> (accessed October 1, 2015).
[10]P. C. D. Hobbs, *Building Electro-Optical Systems: Making It All Work* (John Wiley & Sons, New York, 2000).
[11]J. Thompson, "Advanced Arduino sound synthesis," Make **35**, 80–88 (2014); available at http://makezine.com/projects/make-35/advanced-arduino-sound-synthesis/.
[12]Atmel white paper:AVR20, "Characterization and Calibration of the ADC on an AVR," available at <http://www.atmel.com/images/doc2559.pdf> (accessed October 1, 2015).
[13]M. Meade, "Advances in lock-in amplifiers," J. Phys. E. **15**, 395–405 (1982).
[14]Processing Project Web Site; <http:www.processing.org>.
[15]C. Reas and B. Fry, *Processing: A Programming Handbook for Visual Designers and Artists*, 2nd ed. (MIT Press, Cambridge, 2014).
[16]S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing* (California Tech. Pub., San Diego, 1997).
[17]S. Franco, *Design with Operational Amplifiers and Analog Integrated Circuits*, 3rd ed. (McGraw Hill, Boston, 2002).
[18]See supplementary material at http://dx.doi.org/10.1119/1.4953341 for the Processing and Arduino code as described in this article can be found here, as well as a short tutorial on how to use the code. Up-to-date and modifiable code can be found by going to Ref. 6.

Complex artificial halos for the classroom

Markus Selmke[a] and Sarah Selmke
Universität Leipzig, 04103 Leipzig, Germany

(Received 16 November 2015; accepted 23 May 2016)

Halos represent a common and imposing atmospheric optics phenomenon whose displays are caused by tiny air-borne ice crystals. Their variety stems from a certain set of orientation classes to which these crystals belong. We present a robust and inexpensive device, made of modular components, that allows for the replication of most of these orientation classes in the laboratory. Under the illumination of light, the corresponding artificial halo counterparts emerge. The mechanical realization of this device allows a thorough understanding and demonstration of these beautiful atmospheric optics phenomena.
[http://dx.doi.org/10.1119/1.4953342]

I. INTRODUCTION

In nature, halos can occur all year around due to innumerable refracting and reflecting ice crystals, present in either cold air or in high (and therefore cold) cirrus clouds. Depending on the specific weather conditions, the crystals causing these phenomena typically are either hexagonal columns, hexagonal plates, or both. While many additional shapes and forms can occur, these two suffice to account for most halos. A rich variety of halos exists, comprising arcs, spots, and rings, many of which are vividly colored. The origin of this variety is a set of different classes of stable ice crystal orientations. Still, there is a further source of the wide range of halos as each class of orientations allows different paths for light to travel through these crystals as they take on all orientations allowed in such an orientation class. The mechanism of most halo types can be explained by fake caustics and may be understood with similar concepts as the rainbow. A recent article[1] in this journal gave an introduction to the topic, while details on the nomenclature and classification schemes can be found on Cowley's webpage[2] and in several well-illustrated books[3–5] or articles.[6,7] So far, direct demonstrations of the phenomena's varied displays have remained difficult due to the aforementioned complexity.

II. A MODULAR DEVICE

Using only a single crystal, several experimental works since Bravais[8] have demonstrated the feasibility of constructing dedicated electromechanical devices that can replicate the orientation classes associated with all known regular crystal halos,[1,9–12] except for the Lowitz orientations.[13] While impressive chemical approaches also exist,[14] specifically for the circular halos,[9,15] macroscopic demonstrations have the clear advantage of a better visualization of the