Controlled alignment of donor-acceptor columnar liquid crystals and interrogation of optical properties for applications in data storage and encryption

Harper O. W. Wallace

Joseph J. Reczek
Denison University

Department of Chemistry and Biochemistry

May 4, 2020

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage; the copyright notice, the title of the work, and its date appear; and notice is given that copying is by permission of the author. To copy otherwise, to republish, to post on a server, or to redistribute to lists, requires prior specific permission of the author and/or a fee. (Opinions expressed by the author do not necessarily reflect the official policy of Denison University.)

Copyright, Harper O. W. Wallace, 2020

Abstract

Liquid crystals (LCs) have become a central component of modern display technologies for their ability to dynamically self- and re-orient in response to thermal and electric stimuli, and to selectively absorb or transmit linearly polarized light (LPL) depending on their orientation. However, applications in LC displays and optical data storage tend to rely on all-or-none light transmittance behavior rather than incremental gradations of transmittance; furthermore, there has been little work demonstrating the potential utility of LC materials as standalone information storage media. Donor-acceptor columnar liquid crystals (DACLCs), comprised of diaminonaphthalene (DAN) and naphthalenediimide (NDI) aromatics, constitute a particular class of organic, bi-component LCs, one which demonstrates strong dichroic properties when prepared as standalone bulk films. The extent of charge-transfer (CT) absorbance—and thus LPL transmittance—responsible for their dichroic behavior can be moderated by varying the angle of columnar alignment relative to an applied LPL source. Recent work has demonstrated that DACLC self-assembly into extended molecular stacks can be controlled with exceptional precision by inducing a temperature gradient via a laser direct-write technique; this development allows discrete regions of DACLC films to be aligned independently and hence transmit distinct intensities of light when subjected to a uniform orientation of LPL. Here, a quantitative model for DACLC LPL-response behavior is developed (Section I) and applied to high-density optical data storage and encryption in the context of several distinct schemes (Section II); in Section III, higher-order molecular alignment control is pursued through synthetic modification of the NDI aromatic component, toward applications in high-density, opto-magnetic data storage.

Table of Contents

Introduction

Liquid crystals (LCs) are a class of materials that exhibit phase behavior between that characteristic of crystalline solids and of isotropic liquids, which allows them both to flow and diffuse—like liquids—and to maintain orientational and positional order—like crystalline solids.[1,2] This set of thermodynamic properties renders LCs both ordered and dynamic, conferring stimulus-responsive functional qualities that have found ubiquitous application in modern technologies like flat-panel displays and field-effect transistors.[1–5]

Columnar liquid crystals (CLCs) are a particular class of liquid crystals, comprised of disc-shaped ("discotic") molecular components ("mesogens") that are able to spontaneously self-assemble into extended columnar structures via face-oriented π-π stacking.[1–4,6] However, CLCs need not be comprised of a single species of mesogen; on the contrary, columnar stacking can be stabilized by electrostatic interactions between two distinct molecular components with complementary electron density, one contributing an electron-rich aromatic core and the other contributing an electron-poor one.[1,7] This subclass are called donor-acceptor CLCs (DACLCs).

The particular DACLC material set characterized and applied here is comprised of a 1:1 molar ratio of 1,5-diaminonaphthalene (DAN) electron-rich donor and naphthalenediimide (NDI) electron-deficient acceptor organic aromatics. These DACLCs are said to be thermotropic, which means that structured columnar domains can be formed and via temperature control;[1] in particular, when regions of DACLC film are slowly cooled from a melt phase, the donor and acceptor components spontaneously self-assemble in an alternating pattern to form rod-like stacks (Fig. 1a).

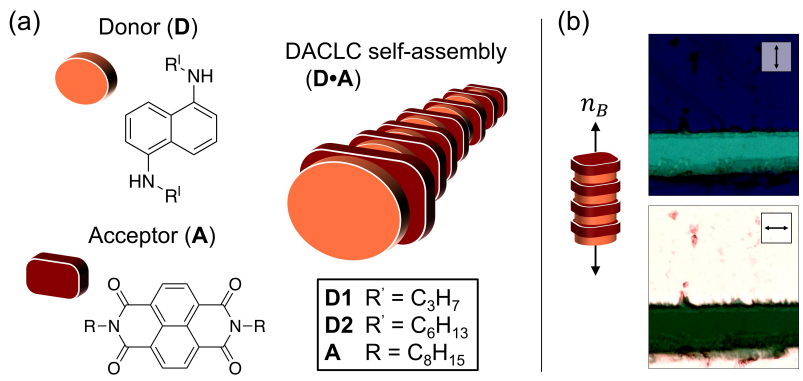


Figure 1. (a) Spontaneous columnar self-assembly of DAN donor and NDI acceptor molecules. (b) DACLC domains aligned with n_B (background) strongly absorb incident linear polarized light (LPL) oriented parallel to n_B (top) and strongly transmit LPL oriented perpendicular to n_B (bottom).

Assembled in this way, DACLCs exhibit charge-transfer (CT) behavior selectively in the direction of columnar alignment, a phenomenon which has bred much interest in the application of similar CLCs to the development of organic semiconductors and microwires for novel organic electronic devices.[1–5,8] In particular, π-π orbital overlap between the aromatic cores of adjacent DACLC components in a columnar stack allows direct electronic excitation from the HOMO of the donor molecule to the LUMO of the acceptor molecule.[4–6] The extent of this energy gap is tunable with the electronics of the organic components, and hence the wavelength of maximum absorbance (λ_{max}) can be controlled by selecting DAN and NDI aromatic components that have been synthetically core-substituted to achieve the desired HOMO/LUMO energies.[4]

Because CT absorption is allowed only in the direction of π-π stacking, incident energy in the form of linearly polarized light (LPL) is selectively absorbed when LPL is oriented in the direction of columnar alignment; for the same reason, electronic excitation and thus CT absorption are disallowed when incident LPL is oriented orthogonal to the direction of columnar alignment, leading to high transmittance.[4,5] Selective absorption of LPL oriented in the direction of columnar alignment gives rise to a property called "dichroism" (Fig. 1b), and the magnitude of this effect ("dichroic ratio" $= \alpha_{\perp}/\alpha_{\parallel}$) exceeds 10 in the CT region in these materials.[3–6]

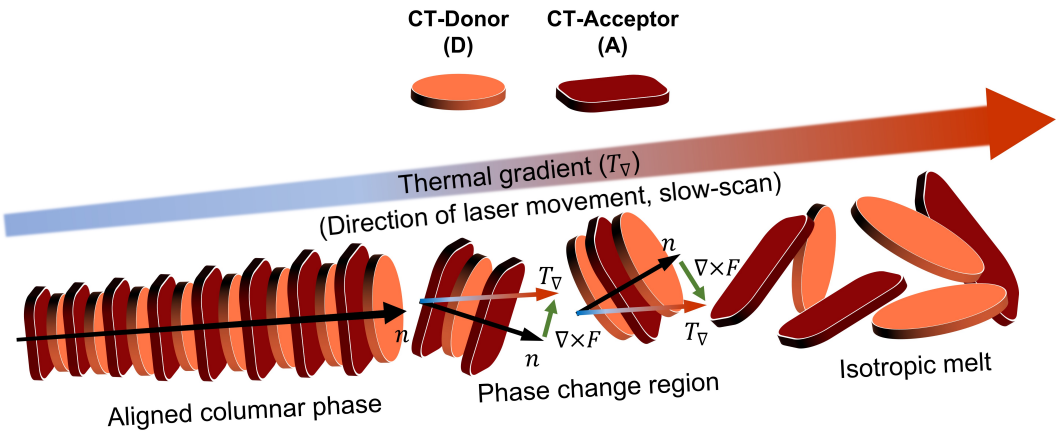


Figure 2. Control of columnar alignment via laser-induced thermal gradient.

Previous work has demonstrated that precise control over local columnar alignment of DACLC films can be achieved using a laser direct-write technique, in which a scanning laser induces a thermal gradient (T_∇) that exerts a torque on the columnar director (n) of a cooling DACLC column to reorient the column in the direction of T_∇ (equivalent to the direction of laser writing) at slow write speeds (0.6–0.8 mm/s) (Figs. 2 and 3).[5] Columnar disalignment (isotropy) can be achieved using the same technique, simply by increasing the write speed (> 5 mm/s) and hence the rate of cooling.[5]

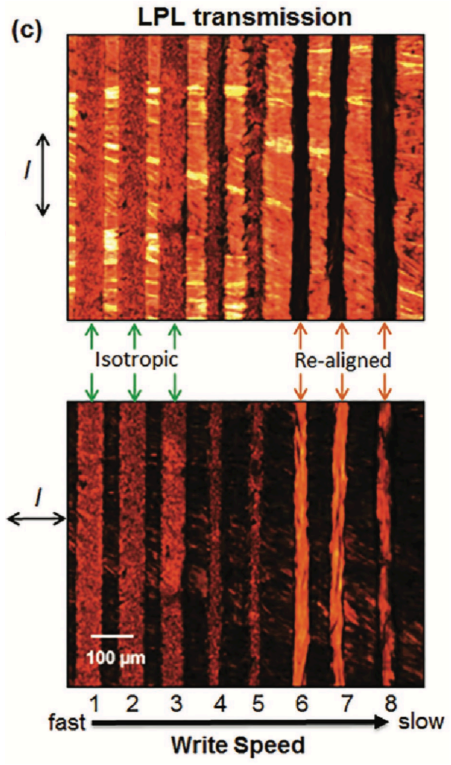


Figure 3. LPL microscopy (LPL-M) images demonstrating DACLC alignment control using the laser direct-write technique described above; fast write speeds (> 5 mm/s) effect columnar isotropy, while slow write speeds (0.6–0.8 mm/s) effect columnar alignment (anisotropy). LPL orientation is indicated beside each image (*I*). (Figure taken from Ref. [5], Fig. 2c).

The columnar alignment control demonstrated in Figure 3 can be achieved in-house using a custom laser-writing setup (Appx. A), and this capability has been used throughout subsequent sections in order to demonstrate the potential utility of these materials in optical information storage and encryption. In particular, laser-aligned DACLC regions were interrogated in order to develop a quantitative model for their characteristic LPL-response behavior (Section I), which was used to illustrate applications of DACLCs in optical data storage and encryption in the context of several distinct schemes (Section II); finally, higher-order molecular control was pursued through synthetic modification of the NDI aromatic component, toward applications in high-density, opto-magnetic data storage (Section III).

I. Characterization of LPL-Response Behavior of Aligned DACLC Films

Introduction

Previous work has demonstrated that DACLCs are highly anisotropic absorbers of LPL.[6] However, existing work has not completely characterized the nature of this response or sought a quantitative model. Quantitative characterization of LPL transmittance through aligned regions of DACLC films offers additional information about the nature of these materials' optical properties and also about individual films themselves.[3] Here, LPL microscopy (LPL-M) was used to generate digital images of laser-aligned DACLC film regions, which were analyzed using custom MATLAB scripts (Appx. E) to derive a mathematical relationship between the orientation of LPL (θ_{LPL}) and the intensity of transmitted light (I_{obs}). This analytical model was used to assert the ability of aligned regions to act as micron-scale linear polarizers, which property was leveraged in Section II as the basis for several data encoding schemes to demonstrate applications of DACLC films to optical data storage and encryption.

Results and Discussion

Regions of DACLC films were aligned using the laser direct-write technique described above, and aligned regions were exposed to angles of LPL (θ_{LPL}) at 5° increments from 0° to 175° (defined here and after by the unit circle). 20x-magnified images of film regions were collected for 36 angles of LPL by LPL-M and subsequently analyzed using custom MATLAB scripts (Appx. E; Fig. 4a). Three test regions of particularly high-fidelity alignment were identified by inspection (Fig. 4b), and the average grayscale intensity over each aligned portion of each LPL-M image were normalized to the transmittance intensity through an isotropic control region (cf. Experimental) and plotted against θ_{LPL}. The results of this analysis demonstrate a clear sinusoidal relationship between θ_{LPL} and transmittance intensity through aligned regions of DACLC films (Fig. 4c).

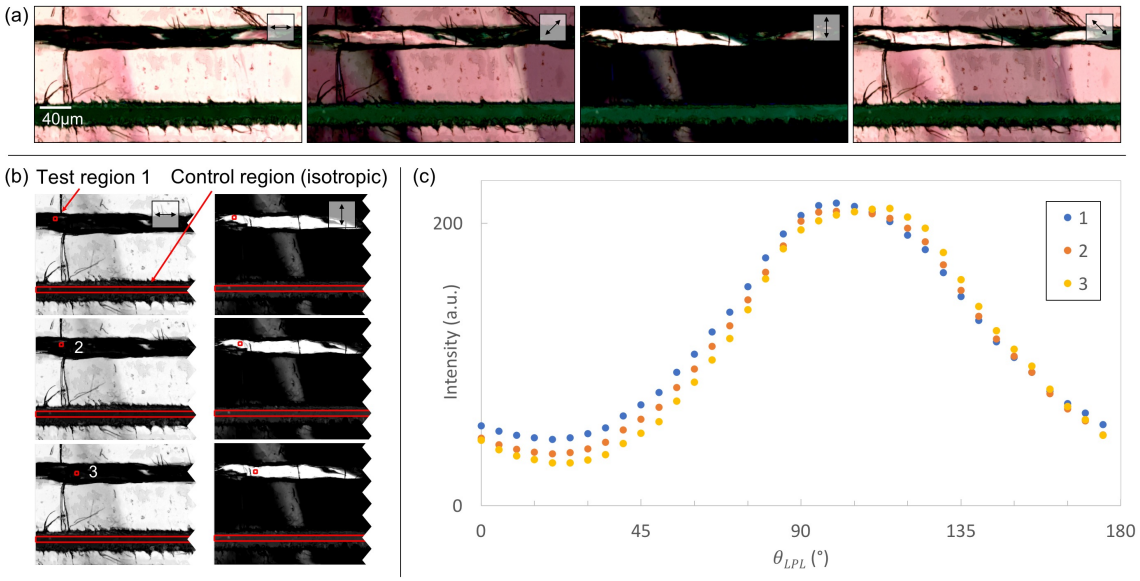


Figure 4. Transmittance intensity varies sinusoidally with the relative angle of LPL to columnar alignment. (a) LPL-M images of a laser direct-write aligned region (top left) of DACLC film, on a bulk-aligned background (middle), and isotropic control region (bottom). θ_{LPL} indicated here and after by the arrow inset in the corner of the row or column. (b) Three (5 μm)^2 test regions within the direct-write aligned portion of the film, and isotropic control, outlined by analysis output. (c) Plot of normalized grayscale intensity versus θ_{LPL} for each of the three test regions in each LPL-M image demonstrates a clear sinusoidal relationship; sinusoidal regressions (not shown) yield $R^2 > 0.99$ for each test region.

The sinusoidal relationship between θ_{LPL} and transmittance intensity is demonstrated more convincingly still by a similar analysis performed on film regions aligned at higher fidelity using the NanoScribe (Fig. 5). In this analysis, portions of the image corresponding to defects in direct-write alignment were detected by a custom algorithm (Appx. E; Fig. 5b) and removed in image pre-processing. Defects were not removed in later analyses because R^2 comparisons of sinusoidal regressions with and without defect removal were not appreciably different (analysis not shown); importantly, this means that even the clear flaws in the alignment of regions in Fig. 5a do not impede grayscale-intensity transmittance analysis. Intensity of LPL transmitted through each aligned region has been normalized to that through the bordering isotropic region (Fig. 5b) before plotting against θ_{LPL} (Fig. 5c).

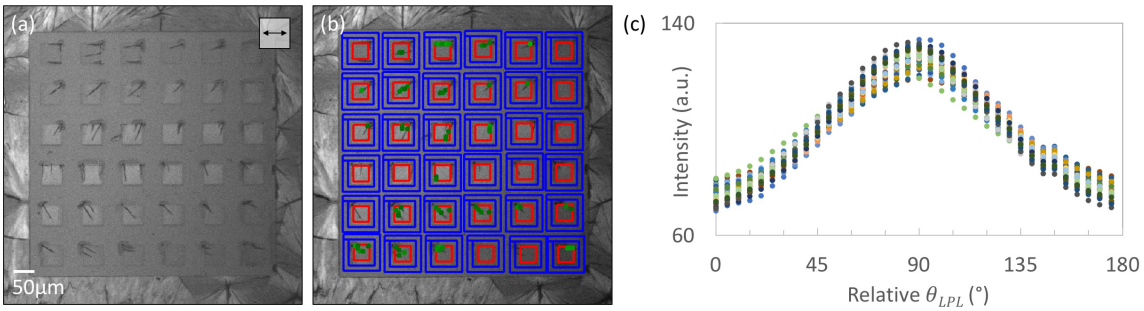


Figure 5. Transmittance intensity varies sinusoidally with the relative angle of LPL to columnar alignment. (a) LPL-M image of 36 (50 µm)^2 regions aligned in 5° increments, and (b) the same image annotated in MATLAB, with aligned regions outlined in red, isotropic borders in blue, and detected defects in green (brightness corresponds to confidence in defect-tagging; Appx. E). (c) Plot of normalized LPL transmittance intensities versus relative θ_{LPL} (to alignment angle) demonstrates a clear sinusoidal relationship; sinusoidal regressions (not shown) yield $R^2 > 0.99$ for each region.

The sinusoidal relationship between θ_{LPL} and transmittance intensity through aligned DACLC regions is mathematically consistent with Malus's law, which describes the intensity of LPL transmittance through a perfect, second linear polarizer; one formalization gives that transmittance intensity, I_{obs}, varies as the square of the cosine of the angle of incident LPL relative to the orientation of the second linear polarizer, $\theta_{LPL,rel}$ (Equation 1).[9,10] In the present analysis, $\theta'_{LPL,rel}$ is taken to be the angle of incident LPL relative to the angle of columnar alignment, and it is phase-shifted by 90° to fit observed results; empirical constants k and b are related to alignment fidelity, material absorptivity, and film thickness (Equation 2).

$$I_{obs} = I_0 \cos^2\left(\theta_{LPL,rel}\right) \tag{1}$$

$$I_{obs} = k\, I_0 \cos^2\left(\theta'_{LPL,rel} + 90°\right) + b \tag{2}$$

An equivalent formalization of Malus's law (Equation 3) describes the transmittance of unpolarized light through two in-line linear polarizers oriented at angles $\theta_{p,1}$ and $\theta_{p,2}$ in the plane of incident light,[9,10] thereby accounting for the absolute orientation of each polarizing optic,

rather than the relative orientation of one to the other. This latter formalization can be used in the present analysis to describe transmittance intensity through an aligned region as a function of the absolute LPL and alignment angles explicitly, where θ_{LPL} (Equation 4) is equivalent to the orientation of the first polarizer, $\theta_{p,1}$ (Equation 3), and the angle of maximum transmittance through the film, θ_{max}, is equivalent to the orientation of the second polarizer, $\theta_{p,2}$ (Equation 4). θ_{max} is related to alignment angle, θ_w (Equation 4), by $\theta_{max} = \theta_w \pm 90°$, a phase adjustment consistent with the one needed in Equation 2.

$$I_{obs} = I_0 \, cos^2\left(\theta_{p,1} - \theta_{p,2}\right) \tag{3}$$

$$I_{obs} = k \, I_0 \, cos^2(\theta_{LPL} - \theta_w + 90°) + b \tag{4}$$

By accounting for θ_{LPL} and θ_w independently in the expression for transmittance intensity, it becomes possible to discern, using transmittance measurements, the absolute angle of columnar alignment, θ_w, in the frame of reference defined by the values for θ_{LPL}. In particular, only three measurements of I_{obs} at unique θ_{LPL} are required to fully determine the form of Equation 4, and hence the value of θ_w, for a given aligned region. This is the done by four-parameter sine curve fitting, or "splining," using straightforward mathematics (Appx. B).

The application of splining to determine θ_w of aligned regions is illustrated in Fig. 6, which depicts three aligned regions in a 36-region film that were analyzed by LPL-M. Transmittance intensities determined from LPL-M images (at $\theta_{LPL} = 0°$, $45°$, and $90°$; Fig. 6b) were used for each of these three regions to determine sinusoidal fit parameters (Fig. 6c). $\theta_{w,fit}$ is derived from the phase shift parameter from the sinusoidal fit, and in all three cases, this value is within 2° of the intended write angle, θ_w. This result points both to high fidelity of alignment and high accuracy of splining analysis, using just three LPL-M images.

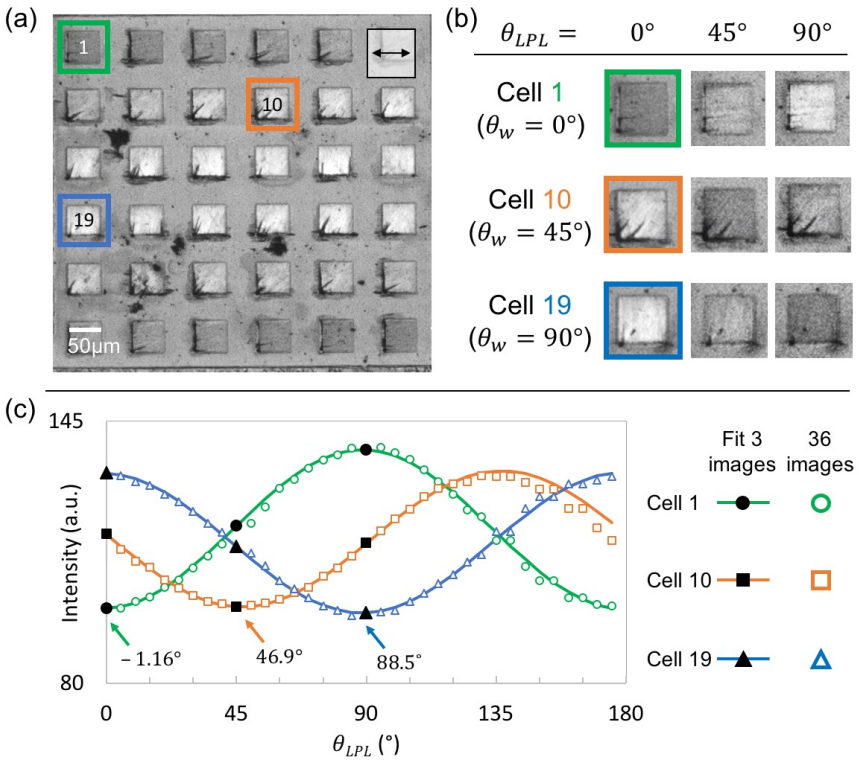


Figure 6. Splining was used to successfully determine θ_w of three (50 μm)^2 aligned DACLC regions. (a) LPL-M image of 36 regions, aligned in 5° increments, 0–175°, in a DACLC film; three of these images (indicated by solid border) were used in splining analysis. (b) Magnified LPL-M images (at indicated θ_{LPL}) of the three regions, aligned at indicated write angles. (c) Plot of normalized transmittance intensity versus θ_{LPL} for each region, from 36 LPL-M images. Three values of θ_{obs} for each region (black) were used to determine sinusoidal fit parameters (solid curves), from which $\theta_{w,fit}$ is derived, indicated in degrees beneath each curve. In each of the three cases, $\theta_{w,fit}$ is within 2° of the intended write angle, θ_w. Note that values for θ_w and $\theta_{w,fit}$ correspond to the θ_{LPL} of minimum light transmittance; this result is consistent with CT-absorption behavior, since maximum absorption occurs when θ_{LPL} is in the direction of θ_w.

In order to characterize the expected accuracy of $\theta_{w,fit}$ from splining, 216 (50 μm)^2 regions (36 in each of six DACLC films) were aligned in 5° increments from 0–175° (as in Fig. 6a) and analyzed both as individual films and as a collective pool (Table 1). As before, transmittance intensities at $\theta_{LPL} = 0°$, 45°, and 90° were used in splining; though any three values of θ_{LPL} can be used to determine $\theta_{w,fit}$ by splining, results were found to be most accurate over all values of θ_w when transmittance intensities are compared at 45° increments (data not shown).

Table 1. Splining values for transmittance intensities give highly accurate estimates for alignment angle, in a sample of 216 (50 μm)^2 aligned regions (36 per film). $\theta_{w,fit}$ is derived from three-point splines with 0°, 45°, and 90°; (*) 4-point least-squares fits with 0°, 30°, 45°, and 90°; (†) 36-point least-squares fits using all LPL-M images. Errors in $\theta_{w,fit}$ are calculated relative to θ_w. Accuracy at, e.g., 10° resolution indicates that $\theta_{w,fit}$ is within 5° of θ_w. Expected percent accuracy is calculated as the area between resolution bounds under the t-distribution with $\bar{\bar{x}}$ and s for a single film or the pool. Note the film-to-film variation in $\theta_{w,fit}$ error and accuracy, and that expected accuracy does not necessarily improve by including additional measurements.

	Error in $\theta_{w,fit}$	**Expected accuracy (%) at resolution (95% C.L.)**				
Film	**(95% C.I.)**	**10°**	**15°**	**20°**	**20° (*)**	**20° (†)**
1	$-1.5° \pm 1.5°$	70.9	88.5	96.2	97.8	96.4
2	$-3.3° \pm 1.4°$	62.8	82.9	93.7	90.3	93.6
3	$-0.6° \pm 0.9°$	92.1	99.0	99.91	99.9	99.9
4	$-0.2° \pm 0.9°$	94.2	99.4	99.96	99.98	—
5	$-3.9° \pm 0.9°$	65.7	91.3	98.8	99.6	99.2
6	$-3.3° \pm 1.9°$	80.1	98.2	99.94	99.9	99.98
Pool	$-2.1° \pm 0.5°$	77.4	93.5	98.7	98.5	98.2

The increase in expected accuracy at lower resolutions follows from the wider allowable margin for error in $\theta_{w,fit}$ at these lower resolutions. However, in the context of data storage, readout resolution determines the number of unique states that can be distinguished from a unit ("bit") of information;[11,12] as such, high-accuracy performance at high resolutions is a critical requirement for high-density data storage applications. Indeed, this tradeoff between solid-state data storage density and readout accuracy is well-established and non-trivial even in state-of-the-art data storage media, though there exist many design and post-readout analytic approaches that can be used to minimize actual error rates;[11,13] these considerations are discussed in greater detail in Section II.

The most significant result of this analysis is the considerable variation in $\theta_{w,fit}$ errors and expected accuracies between region samples from different films (e.g., film 4 at 10° resolution performs at higher accuracy than film 2 in any 20° resolution analysis). This variation primarily

reflects imperfections in laser-alignment, which is important because alignment fidelity has been optimized only enough to demonstrate proof-of-principle; as further work is done to determine write parameters and conditions that optimize alignment fidelity, figures for readout accuracy will doubtless improve.

A pooled error in $\theta_{w,fit}$ of magnitude confidently greater than $1.0°$ at the 95% confidence level is evidence of a modest systematic error resulting from any of several possible sources, including: bias in angles of region alignment, perhaps resulting from how the sample is mounted in the NanoScribe for laser-writing or on the microscope stage for LPL-M imaging (though rotational corrections for grid orientation in the LPL-M images are consistently $\ll 1°$; data not shown); bias in θ_{LPL} recorded during LPL-M, perhaps due to a constant shift of the angle labels used to orient the manually-rotated polarizing optic in the microscope; or bias introduced in splining, perhaps an unanticipated mathematical consequence of the particular values of θ_{LPL} used in analysis. This latter source of systematic error seems likely, since bias in the error in $\theta_{w,fit}$ becomes almost universally positive when four or 36 points are used in analysis (data not shown); if a primary contributor to overall error, systematic error resulting from the choice of values of θ_{LPL} likely reflects underlying error in the transmittance intensities themselves, e.g., due to secondary polarization in LPL-M which is not sufficiently corrected for by normalizing transmittances to those through nearby isotropic control regions. Addressing these sources of error will likely prove critical to improving $\theta_{w,fit}$ accuracy for aligned regions, particularly at 10° resolution and below; after crude correction for systematic error in $\theta_{w,fit}$ by artificially zero-centering the distribution of error terms (i.e., by subtracting $\bar{\bar{\varepsilon}}_{\theta_{w,fit}}$), the pooled expected accuracies increase rather dramatically to 92.3%, 98.2%, and 99.8% at 10°, 15°, and 20° resolution.

Conclusions

Quantitative characterization of the light-response behavior of aligned regions of DACLC films offers a more complete picture of the dichroism that is readily apparent on exposure to θ_{LPL} parallel and orthogonal to the angle of alignment (Fig. 1b). That this light-response behavior can be accurately characterized by Malus's law, making adjustments for scaling and baseline-correction, illustrates that aligned regions effectively behave as independent linear light polarizers. This behavior, coupled with micron-scale control over region alignment via the laser direct-write technique described above, enables the manufacture of complex, rewritable, micron-scale-, multi-region polarizing films, a complex optic which is not traditionally available due to the fabrication constraints of polarized films and the geometric constraints of planar electrodes used to control liquid crystal orientation.[14–16]

The ability to determine sinusoidal fitting parameters that relate θ_{LPL} to transmittance intensity through an aligned DACLC region using the adjusted form of Malus's law (Equation 4) is an important preliminary step to being able to design films that yield pre-defined transmittance intensities on application of a specified θ_{LPL}. Moreover, that the key fitting parameter, $\theta_{w,fit}$, can be accurately derived by a three-point spline demonstrates that written alignment angles can be determined from transmittance intensities without exhaustive data collection; this result has been leveraged in the applications to data storage and encryption discussed in Section II.[17]

The analysis used to describe $\theta_{w,fit}$ readout accuracies will no doubt prove helpful to discerning and addressing the greatest barriers to optimized, high-fidelity laser-writing and high-resolution LPL-M analysis. Further exploration of splining results may seek to derive expressions for film thickness or alignment fidelity in terms of k and b (Equation 4); alignment fidelity analysis would likely rely on something like the von Mises distribution (or the von Mises–Fischer

distribution, if a third dimension of alignment is taken into account),[3,18] though the details are left to future student-researchers.

Experimental

DAN and NDI components were synthesized using published procedures.[4] DACLC mixtures were prepared by weighing out the correct molar ratio of components, and then physically mixing with a spatula prior to melting with a heat gun. The resulting mixture was iteratively corrected using ^{1}H NMR (JOEL 400 MHz) until integration of the respective donor and acceptor peaks gave a ratio of 1.00 ± 0.02. DACLC thin films were prepared by filling a glass cell, comprised of a glass coverslip and microscope slide separated by 20 μm silica beads, via capillary action at 175°C.

Laser-writing tests were performed on samples that had been melted (175°C) and then cooled at 2°C/min to room temperature. Alignment patterns shown in Fig. 4 were written using the custom laser-writing setup housed at Ebaugh Laboratories and detailed in Appx. A; patterns shown in Figs. 5 and after were written into DACLC films using the NanoScribe GmbH Photonic Professional GT 3D printer equipped with a 20x Zeiss EC Epiplan-Neofluar 0.50 NA objective and adjusting the power, scan speed and hatch angle of the scanning beam to control the degree and direction of columnar alignment in each pixel. Isotropic regions were written using a 50 mm/s laser scan speed and 40–60% of maximum laser power. Depending on the size of the pixel/scanning region, aligned areas were written using a 1.5–3.5 mm/s scan speed and 10–12% of maximum laser power, with the resultant polarization direction perpendicular to the hatching direction (thus parallel to the direction of the thermal gradient produced by the laser).

LPL microscopy (LPL-M) was performed using an Olympus BX51TRF microscope and accessories from McCrone Microscopes in transmission mode on a Linkam large area thermal

stage. LPL-M images were captured with a PAXCAM 3 camera, and color images were recorded using a Thorlabs color CCD camera (DCU224C) and Nikon Eclipse TI camera mounted on an inverted stage microscope equipped with a single polarizer. Image analysis was performed using custom scripts written in MATLAB (Natick, MA, USA, MathWorks; Version R2018a, 9.4.0.813654; License Number: STUDENT) and provided in Appx. E.

Both visual inspection and quantitative analysis of LPL-M images demonstrated that LPL transmittance intensity through isotropic film regions varies with θ_{LPL} (Fig. 4a). This observation is not consistent with theory, since columnar disalignment into sufficiently small domains results in CT-absorption being equally favored in all directions in the plane of the film, and hence equivalent transmittance in all directions, at the scale considered here.[4–6] This result is likely due to a tertiary polarization effect (e.g., from polarization bias in the detector camera used in LPL-M) and therefore impacts transmittance intensities measured for aligned regions; as such, it has been corrected in the analyses presented here by normalizing isotropic transmittance intensities across LPL-M images before proceeding with analysis of aligned regions (Appx. E). Because this bias is a source of systematic error that could theoretically be eliminated by modifying the LPL-M optics or analytically characterizing the bias independent of any particular sample film, this correction was justifiably performed using all available LPL-M images, before the any subset of transmittance values was used (as in Fig. 6 and in similar analyses, below) to determine $\theta_{w,fit}$.

II. Application of DACLC LPL-Response to Optical Data Storage and Encryption

Introduction

A growing need for low-cost, high-density data storage media has roused increasing interest in taking advantage of tunable systems with molecule-scale functionality.[11,19–21] In particular, the dynamic stimulus-responsiveness of LCs that has been so widely applied in digital displays has also found utility in optical data storage.[22–26] However, to the best of our knowledge, existing such applications do not function as standalone materials (e.g., rely on one or several external electric fields), and have demonstrated successful optical encoding only using binary (base-2) or ternary (base-3) systems; the former weakness introduces a need for additional appliances—beyond the materials themselves—to maintain information integrity, thereby increasing both cost and effective system size, and the latter precludes the possibility of maximizing storage density by accessing higher-base encoding.[12]

DACLC alignment in standalone films, by contrast, has been shown to persist indefinitely in ambient conditions,[5] and aligned regions demonstrate a characteristic, continuously-variable LPL-response that may be leveraged for near-analog data storage (Section I). Here, these principles, coupled with the ability to achieve high-fidelity alignment control achievable via laser-writing, were used to demonstrate novel, high-base (and hence, theoretically, high-density) schemes for optical information storage and encryption.

As an example, the dependence of LPL transmittance intensity through an aligned region on the angle of LPL applied (Equation 4) means that information inscribed in a film in terms of the intensity of transmitted light through one or many aligned regions can be correctly interpreted only in a particular window of $\theta_{LPL} = \theta_{LPL,key} \pm \varepsilon$. This principle is simulated for an outlined "H" pattern (Fig. 7).

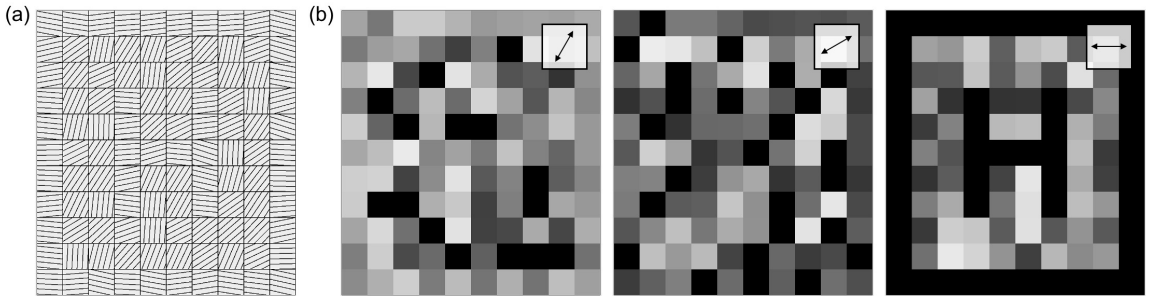


Figure 7. Data encryption principle based on angle-dependent LPL transmittance through aligned regions of DACLC films (custom script, Appx. F). (a) Calculated alignment angles for an outlined "H" pattern with $\theta_{LPL,key} = 0°$. (b) Predicted transmittance intensities at $\theta_{LPL} = 60°$, $30°$, and $0°$, flooring values less than 20% ("selection threshold"; black). Correct readout is only possible for $\theta_{LPL} \approx \theta_{LPL,key}$, with error allowance ε determined by alignment fidelity and transmittance resolution (system-limited), and also by selection threshold and actual write angles (defined).

To demonstrate the application of this principle to optical data storage and encryption, the LPL-response of aligned DACLC regions characterized in Section I was used to optically encode (i) an 8-bit image (converted to grayscale) of the Nintendo character Mario (Fig. 8), (ii) a plain-text message (Fig. 11), and (iii) visually-intelligible strings (Fig. 12) on DACLC films using three distinct schemes. Synthetic-tunable degradation of optical contrast was analyzed to demonstrate another feature of particular utility in applications of DACLCs to data storage and encryption.

Results and Discussion

Transmittance-based information encoding and retrieval scheme (single DACLC film)

Empirical values for k and b for a given DACLC film sample (Equation 4) were used to deconstruct the original grayscale image into the angles of columnar alignment required to yield LPL transmittance, at an intended $\theta_{LPL,key} = 0°$, corresponding to the grayscale value of each pixel (Fig. 8a); these calculations were performed using a custom MATLAB script (Appx. E). When the film is exposed to $\theta_{LPL,key}$, the original image is faithfully represented, but when the film is exposed to other values of θ_{LPL}, the stored information is obscured (Fig. 8b,c).

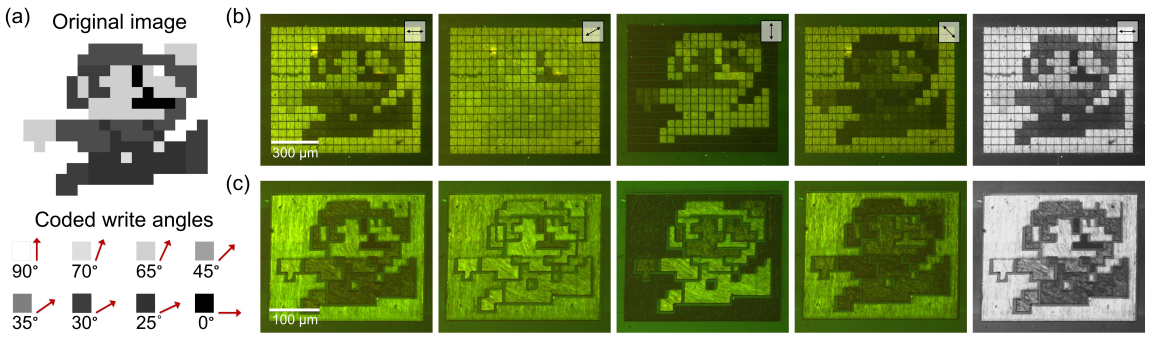


Figure 8. Optical information stored in DACLC films in terms of transmittance intensities is retrievable only on application of $\theta_{LPL,key}$. (a) Original image encoded in terms of transmittance intensities for $\theta_{LPL,key} = 0°$ in a DACLC film as (b) discrete, regular, (50 μm)^2 regions corresponding to pixels and (c) irregular regions with adjacent, like-intensity pixels written continuously. LPL-M images (raw and grayscale-normalized) demonstrate that faithful information readout is possible only where $\theta_{LPL} \approx \theta_{LPL,key}$.

In addition to encoding the original grayscale image directly in pixelated form (Fig. 8b), adjacent pixels in the original image that shared the same intensity were patterned as a continuous region on the DACLC film; this alternative approach can allow for more complex patterning and smaller feature resolution, since aligned regions are able to take any size or shape (Fig. 8c).

Importantly, the information encoded in this example is readily intelligible on visual inspection of the LPL-exposed film, even, to some extent, at values of θ_{LPL} other than the one intended (despite that some images have inverted grayscale levels and others are rendered with poor contrast). This result is not a weakness of the LPL-dependent optical encoding strategy, but rather an artifact of (i) the stored information and the optical output both consisting of equivalent, grayscale values, and (ii) those grayscale values being represented in a direct, one-to-one spatial correspondence. However, were the encoded values not visually meaningful (e.g., in the case of a numeric sequence encoded in terms of transmittance intensities), or had they been pre-processed by reversible scrambling or applying a masking function, or otherwise obscured, the

grayscale information read out on the film's exposure to LPL would be intelligible only at the intended angle of LPL (cf. Fig. 11b,c as an example of pre-processing such that encoded information is not immediately intelligible from grayscale transmittance values, though note that the data in Fig. 11, unlike in Fig. 8, were not encoded in terms of transmittance intensities).

As a solution, information can be encoded in terms of transmittance intensities that are visually meaningless. In the case of information like strings of letters, this can be done by converting characters to numerical values by the American Standard Code for Information Interchange (ASCII) protocol, which values are converted to a particular base ("radix"), defined as the number of distinguishable transmittance intensities used in encoding. N.b. that "base" is used in later discussions of an encoding scheme based on θ_w rather than transmittance intensities; as implied by Fig. 10, these contexts are importantly different, and hence angular and transmittance resolutions are not equivalent, e.g., between angular and transmittance "base-10."

As an example of how string data might be converted to transmittances, consider a system in which intensities can be resolved to 33%; then 0%, 33%, 67%, and 100% is used as the set of possible transmittance values. ASCII codes are converted to base-4, with each digit able to store any of four possible values—0, 1, 2, or 3—which are related to related to transmittance intensities as a proportion of the maximum value (here, 3) and to θ_w for a given θ_{LPL} by Equation 4, assuming ideality ($k = 1$ and $b = 0$). The results of character-to-transmittance conversion are illustrated for the string "HELLO WORLD" (Fig. 9). Though not illustrated, correct transmittance readout would be dependent on application of the correct θ_{LPL}, as in Figs. 7 and 8.

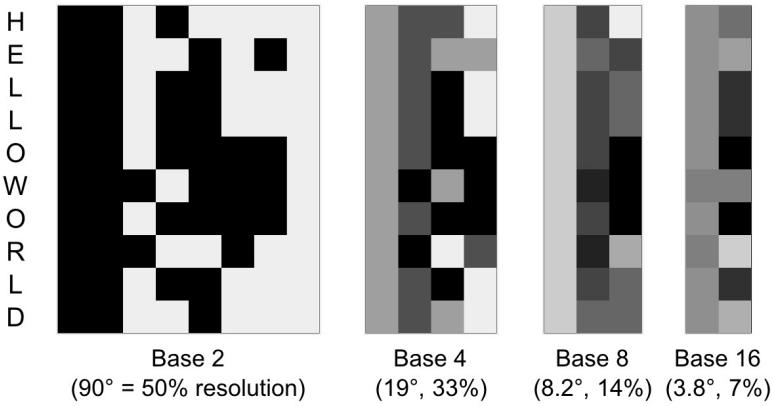


Figure 9. Information can be encoded in terms of transmittance intensities that are not visually meaningful (custom script, Appx. F). Each letter in the string "HELLO WORLD" was encoded in ASCII, which numerical result was then converted to the base indicated and represented graphically as a row of transmittance intensities. Angular resolutions indicate the narrowest margin between possible θ_w, and percent transmittance resolutions indicate the (constant) spacing between intensities, assuming in both cases that $k = 1$ and $b = 0$ (Equation 4).

The tradeoff between resolution and readout accuracy apparent from Table 1 (Section I) is now complicated by an additional interest: storage density. Higher base values allow greater storage density,[11,12] since fewer cells are required to represent each letter (Fig. 9). However, this increase in storage density requires finer resolution of write angles and transmittances—indicated parenthetically beneath each base—as is apparent from the relative closeness of distinct shades in base-16, e.g., compared to base-2. The ability to achieve high resolution of θ_w and transmittances is a practical consideration limited primarily by alignment fidelity in this analysis, though potentially by the optical recording and analysis tools.

With the operating principles well-established for an encryption scheme in which visually meaningless information is interpreted directly from transmittance intensities (as in Fig. 8), it is possible to evaluate the merits and weaknesses of such a scheme, and also of related ones. One seeming weakness is that variation in film thickness between samples and even within a single sample leads to variation in transmittance intensities for the same θ_w on application of a given

θ_{LPL}; in practice, this can be solved by post-hoc computational analysis, e.g., by splining transmittances from LPL-M images to determine θ_w for each region, and then relating θ_w and θ_{LPL} to expected I_{obs} by Equation 4, assuming ideality ($k = 1$ and $b = 0$).

A much more consequential weakness of such a scheme is that data readout accuracy is optimized when raw intensities are quantized to a set equally-spaced values before being interpreted; though the number of quantized values used (i.e., base) determines the extent of spacing between and hence overall resolution, equal spacing ensures that these values can each be resolved to the equal degree of confidence (i.e., it prevents bias for or against particular quantized intensities). However, in the case of encoding based on transmittance intensities, a consequence of this equal spacing is that the sinusoidal relationship between θ_{LPL} and transmittance intensity means that a set of equally-spaced transmittance intensities corresponds to a set of unequally-spaced alignment angles with characteristic, symmetric, and hence exploitable density (Fig. 10a).

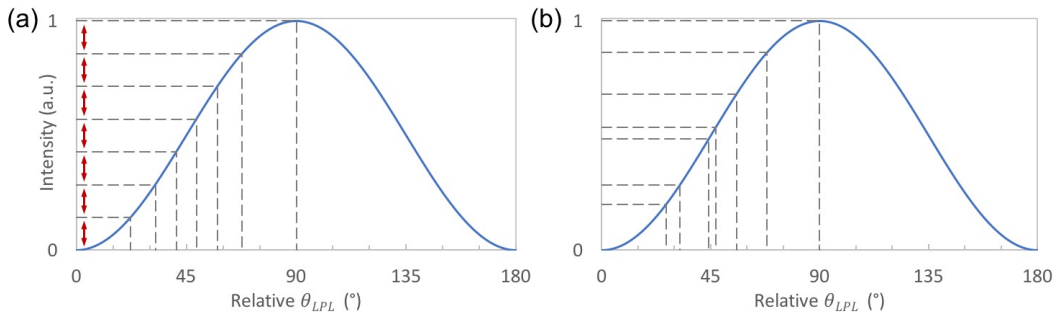


Figure 10. (a) Illustration of the characteristic symmetric, center-dense distribution of alignment angles corresponding to equally-spaced quantized transmittance intensities in base-8. (b) An example set of asymmetric and unequally-spaced such quantized values, as a potential solution at the expense of readout accuracy and bias.

The characteristic distribution of alignment angles corresponding to equally-spaced quantized transmittance intensities makes it possible to determine the frame of reference used

in region alignment, with $\theta_{LPL,key}$ given as a 45° shift from the center of angular density in the distribution of θ_w (equivalent to the value of θ_{LPL} where transmittance intensities are observed to be equally spaced), for modest bases ($4 < b < 20$, given the resolution constraints of this analysis); encoding in higher bases may limit the ability to discern this symmetry, but would require higher-resolution readout. Possible exploitation of equally spaced quantized transmittance intensities or alignment angles seriously undermines the integrity of encryption; as a potential solution, unequally-spaced quantized transmittance values may be used (Fig. 10b) though, as described above, this will reduce readout accuracy and may introduce readout bias.

Another disadvantage of encoding information in terms of transmittance intensities is that it introduces inherent redundancy, since two values of θ_w yield the same transmittance intensity for a given θ_{LPL} (excepting maximum and minimum transmittances). This redundancy may have some practical value, e.g., by complicating cryptographic frequency analysis,[27] but it also limits the number of meaningful values that can be stored in an areal unit to roughly half the number of alignment angles that can be resolvably written to the same unit.

θ_w-based information encoding and retrieval scheme (single DACLC film)

As an alternative to encoding information in terms of transmittance intensities, it is possible to encode information directly in terms of the angle of columnar alignment, in a given frame of reference, θ_{ref} (potentially defined by the values used for θ_{LPL}). Among the advantages of encoding in terms of alignment angles are that (i) values of θ_w between 0°–180° can be distinguished (using transmittance intensities at three values of θ_{LPL}), thereby enabling roughly twice the theoretical storage capacity of transmittance-based encoding, at given alignment resolution; and (ii) without knowledge of θ_{ref}, values of θ_w are known only relatively and are therefore not inherently meaningful without additional information.

To demonstrate the merits of an such an encoding scheme, a message was encoded in terms of alignment angles, which were written to a DACLC film and subsequently read out at high accuracy. Regions were aligned at possible values of θ_w defined as 10° increments from 0°–170° with isotropic as an additional possible state, giving 19 unique data states per data unit. These base-19 "bits" were paired as dyads, each able to store $19^2 = 361$ distinct data states (Fig. 11a). Each distinct dyad was coded to a text character in a cipher with six-to-one redundancy (i.e., six unique dyads correspond to each character), though it is worth noting that the redundant encoding built into the cipher is not an inherent feature of the encoding scheme or the materials but was used in this example only to illustrate incorrect data readout with an incorrect θ_{ref} (Appx. C). Applying the cipher, the characters of the string "ALICE'S_MESSAGE_TO_BOB" were converted to alignment angle dyads, which were used to align (50 μm)^2 regions in 6 x 6 DACLC grid. Values for $\theta_{w,fit}$ were determined from LPL-M images at $\theta_{LPL} = 0°, 45°$, and $90°$ for $\theta_{ref} = 0°$ (correct) and $\theta_{ref} = 90°$ (incorrect), and these calculated alignment angles were interpreted at 10° resolution (base-19) by using cipher as written, and also at an effective 20° resolution (base-10) by granting correct readout of $\theta_{w,fit}$ within 10° of θ_w (Fig. 11b,c).

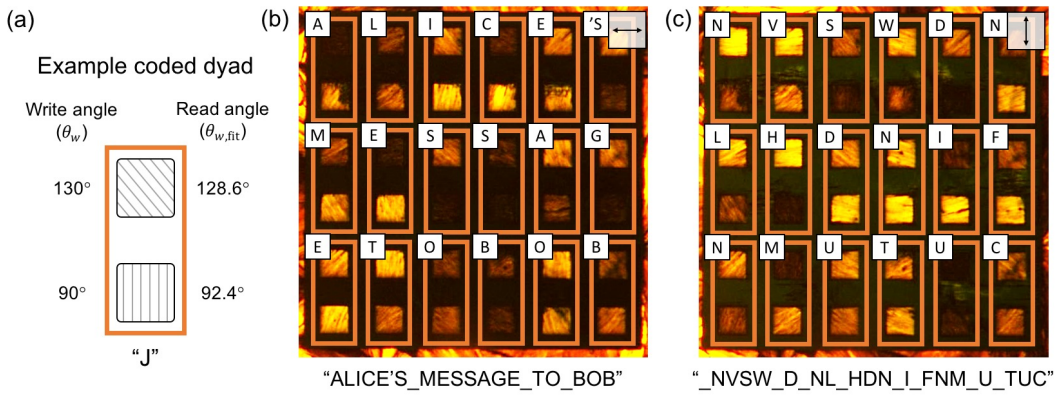


Figure 11. (a) Dyad-based cipher used to encode "ALICE'S_MESSAGE_TO_BOB" into alignment angles. LPL-M images of (50 μm)^2 aligned regions (b) at $\theta_{LPL} = 0°$ and (c) $90°$. (b) Splined transmittance intensities at $\theta_{LPL} = 0°, 45°$, and $90°$ to determine values for $\theta_{w,fit}$ give successful retrieval at 20° resolution (base-10) in the correct frame of reference, $\theta_{ref} = 0°$. (c) In an incorrect frame of reference, $\theta_{ref} = 90°$, these same images give a nonsense output.

In the correct reference frame, $\theta_{ref} = 0°$, the encoded message is correctly interpreted at 20° resolution (base-10; Fig. 11b); however, a single error is observed at 10° resolution (base-19), giving "ALIBE'S_MESSAGE_TO_BOB". This error frequency is consistent with the expected value at 10° resolution (Table 1, Section I) and illustrates the practical tradeoff between data storage density and readout accuracy, though there exist many, sophisticated design and post-readout analytic approaches to reduce this error in practice.[11,13] In an incorrect reference frame, $\theta_{ref} = 90°$, the data are effectively concealed, and the decoded output is a nonsense string.

Importantly, equal spacing of quantized alignment angles, used in this encoding scheme to optimize readout accuracy, cannot be exploited to determine θ_{ref}, as can equal spacing of quantized transmittance intensities to determine $\theta_{LPL,key}$ (cf. Figs. 7 and 9, though note that the spacing described here is not depicted explicitly).

Transmittance-based information encryption and retrieval scheme (two-film overlay)

Since aligned regions of DACLC films behave as linear light polarizers, it is also possible to design encryption schemes that replace the linear polarizer used to generate a single orientation of LPL (applied uniformly to all aligned regions in a DACLC film) with a second film that has been aligned perpendicular to the original θ_{LPL}. Furthermore, it is possible to instead align this second film regionally, in such a way that corresponds to the regional alignment of the first film; when the films are overlaid, transmittance intensity of unpolarized light through a pair of overlaid regions can be expressed in terms of the angles of alignment of the corresponding regions, defined in the same frame of reference (Equation 5).

$$I_{obs} = k' \, I_0 \, cos^2\left(\theta_{w,1} - \theta_{w,2}\right) + b' \tag{5}$$

In this way, film regions can be aligned such that they reveal a particular transmittance intensity only on overlay with a second film, and hence DACLC films can be used to implement encryption schemes that take advantage of overlaid polarizers.[28] To illustrate this application, (50 μm)^2 regions of a "mask" film were aligned arbitrarily. A "key" film was designed with corresponding regions that were aligned relative to the mask to reveal a particular transmittance pattern on exposure to unpolarized light; i.e., a key region was aligned parallel to the corresponding mask region ($\theta_{w,1} \approx \theta_{w,2}$) for maximum transmittance, or orthogonal to it ($\theta_{w,1} \approx \theta_{w,2} + 90°$) for minimum transmittance. A second key film was designed relative to the same mask in order to demonstrate the scheme's versatility (Fig. 12). Importantly, since the encoded information is stored in the film-pair system in virtue of a relationship between the two films, this information is not accessible from either film independently (Fig. 12b); only on mask-key overlay is the intended transmittance pattern observed (Fig. 12c).

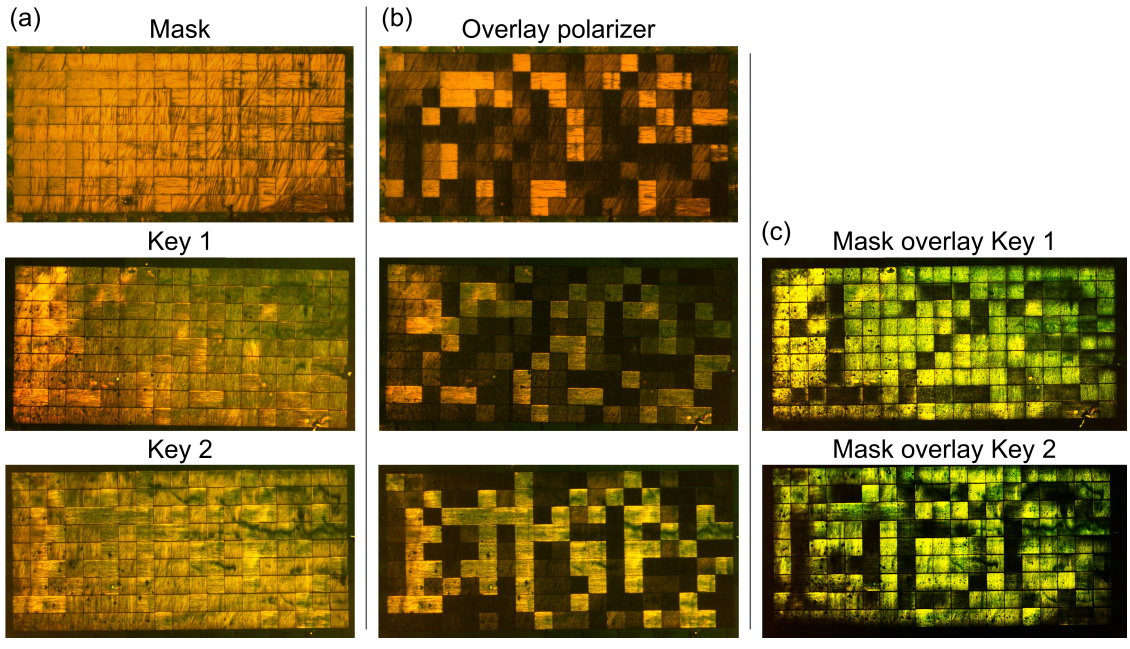


Figure 12. (a) Microscope images (unpolarized light) of a DACLC "mask" film of (50 μm)^2 regions aligned at arbitrary θ_w, and two corresponding "key" films with regions aligned relative to the mask to achieve a particular transmittance pattern on (c) mask-key overlay (unpolarized light). (b) LPL-M images of mask and keys independently ($\theta_{LPL} = 90°$).

Importantly, uniform exposure of either the mask or key to any angle of LPL does not reveal the encoded information (Fig. 12b). To be sure, the resemblance of some portions of the LPL-exposed key films to the corresponding mask-key overlay patterns (Fig. 12b,c) is an artifact of the small number of distinct alignment angles used in this example. Since regions on both films are aligned only at $\theta_{w,1}, \theta_{w,2} \in [0°, 45°, 90°, 135°]$, relative to either common axis to the two films (required to be in alignment for the patterning used here to correspond on overlay), there are four possible transmittance intensities both on application of $\theta_{LPL} = 90°$ and through any pair of corresponding regions; further, these two sets of possible transmittance intensities are approximately identical, since $\theta_{LPL} = 90°$ happens to correspond to $\theta_{w,1} = 90°$, e.g.. Since only two transmittance intensities are seen in the output (Fig. 12c), it is expected that transmittance intensities on application of $\theta_{LPL} = 90°$ (or $0°$) should correspond to the intensities on overlay roughly 1/4 of the time; this effect is clearly small enough that the encoded information is unintelligible from any LPL-M image of either mask or key alone (Fig. 12b), but it is noticeable. Importantly, as the number of possible values for θ_w increases—either to increase the number of distinct transmittance intensities on mask-key overlay (e.g., addition of $\theta_w = 60°$ to allow a 0°–60° pair), or alternatively to allow for additional distinct frames of reference between corresponding aligned regions, with the same number of transmittance intensities (e.g., addition of $\theta_w = 15°$ and $60°$ to allow a 15°–60° pair)—this effect is expected to become negligible.

The encryption scheme illustrated here is fundamentally a pad-cipher, which is unbreakable without both "pad" (analogous to mask) and key, so long as a particular pad corresponds only to a single key ("one-time pad");[29] as such, the versatility demonstrated by designing a second key for use with the same mask comes at the cost of encryption integrity. It is also worth addressing that in this example, as in the example illustrated in Fig. 8, information is encoded in terms of equally-spaced, quantized transmittance intensities; but unlike the scheme

in Fig. 8, this encryption scheme is not exploitable in the ways detailed in Fig. 10, because each pair of corresponding regions is effectively aligned in its own frame of reference, independent of the other regions.

Tunable degradation of dichroism via synthetic modification of DAN component

A final feature of DACLC dichroism that renders the materials well-suited to applications in data storage and encryption is the dependence of alignment lifetime on the molecular components used in the mixture. As a result, the persistence of CT absorption in aligned DACLC films at room temperature can vary depending on mesogen substituent side-chains.[6,30,31]

To demonstrate this, laser-written samples of **A•D1** (DAN-propyl) and **A•D2** (DAN-hexyl) were compared at 1, 2, and 10 days after writing; where the contrast in LPL transmittance through differently-aligned regions persisted reliably at 10 days (and far beyond, though not shown) in the sample of **A•D1**, this contrast was lost in a matter of days in the sample of **A•D2** (Fig. 13).

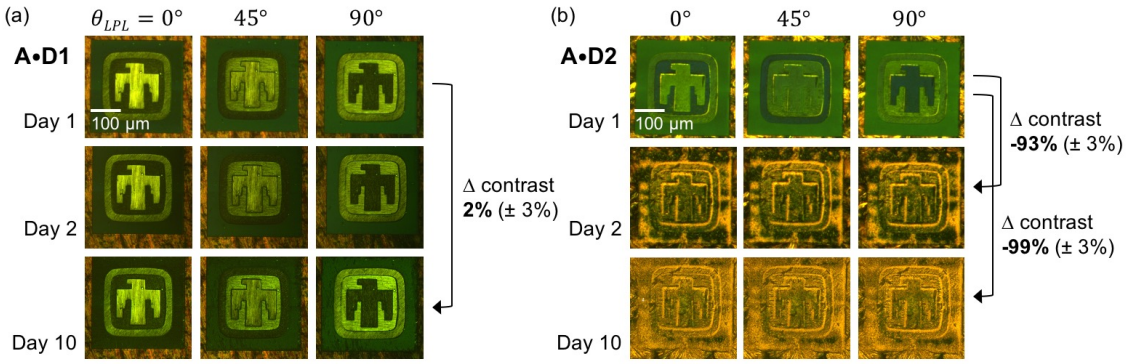


Figure 13. Persistence of information encoded in DACLC films is synthetically tunable via modification to DAN component substituents. (a) Optical contrast persisted indefinitely in a sample of **A•D1** but (b) fadeed significantly in just two days in a sample of **A•D2**. Contrast analysis performed using a custom MATLAB script based on definitions reported in Ref. [32] (Appx. E).

The tunable persistence of DACLC dichroism achieved by synthetic design enables a predictable and controllable life-span for information encoded in DACLC films and hence invites

the possibility to build autonomous self-destruction after specified time window into the storage medium itself. This is yet another design feature that may be leveraged in the application of DACLCs to optical data storage and encryption.

Conclusions

The characteristic LPL-response of aligned regions of DACLC films described in Section I allows the design of sophisticated schemes for optical data storage, retrieval, and encryption. Encoded information can be interpreted at application-specific resolution parameters to optimize either data density or readout accuracy, as required by the application, and the ability to chemically control the lifespan of encoded information affords additional utility of these materials in applications to optical data storage and encryption.

The ability of aligned DACLC films to represent more than two possible meaningful states in the same areal unit of substrate opens the door to novel data storage schemes and affords particular advantages to storage density over conventional binary media. However, it is necessary to acknowledge that the dividends to storage density are not limitless: a binary data storage system consisting of n discrete storage units is able to store 2^n distinct information stages; by contrast, a base-10 system (requiring 36° resolution, if information is encoded in terms of alignment angles) is able to store $10^n = \left(2^{log_2 10}\right)^n = 2^{n\ log_2 10}$ distinct states, which is equivalent to the number of states stored in a binary system of $n\ log_2 10$ units.[12] More generally, a binary system requires $log_2 x$ times as many units to store the same amount of information as a base-x system; assuming that the area occupied by a single unit is equivalent between both cases, this means that the base-10 system requires only about 30% of the area as the binary system. However, the logarithmic relationship between base-value and required discrete units means diminishing returns: angular encoding even at 1° resolution (i.e., base-360) still requires

12% of the physical area as a traditional binary system; and although this represents a theoretical improvement to storage density, it tends to become negligible with increasing investment in optimizing the technology.

As such, the models for optical data storage presented here are more likely to be become competitive with existing magnetic models not by increasing resolution of angular alignment, but rather by minimizing the size of aligned regions.[33] As an example, a DACLC film able to store y times as many binary units as the same area of a magnetic competitor is able to store $2^{ny}/2^{n} = 2^{ny-n} = 2^{n(y-1)}$ times as much information;[12] if the DACLC film is encoded in base-x, rather than binary, then it is able to store $2^{n(y\,log_2 x-1)}$ times as much information, which is not just considerable even for modest y (for meaningful values of n), but also importantly increases exponentially with the size of the medium (i.e., the number of discrete data units, n).

Though the areal storage units described in this analysis measure between tens and thousands of μm^2 (e.g., Fig. 8b,c), these values are used only to demonstrate the highest-fidelity alignment achievable given the constraints of the existing laser-alignment setup. Importantly, these areal storage densities do not actually reflect the lower limit on optical data storage density in DACLC films and would surely be maximized by industrial optimization; however, it is worth acknowledging that current areal data storage density in magnetic media, on the order of 1 TB in^{-2} = 1 bit (10^{-4} μm^2)$^{-1}$,[34–36] is much greater than the density demonstrated here.

Areal storage density in DACLC films could be optimized by minimizing the size of aligned regions, or alternatively by introducing an additional, discernable degree of freedom in columnar alignment, as considered in Section III.

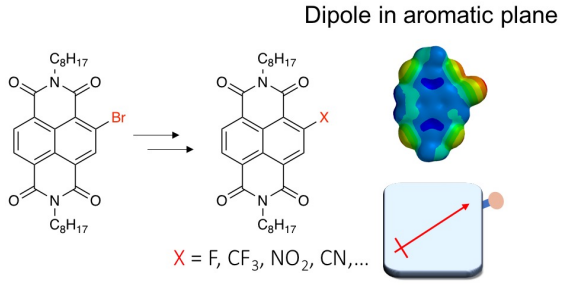
Cf. Experimental in Section I

III. Higher-Order DACLC Alignment Control via Synthetic Modification of NDI Acceptor

Introduction

In recent years, core-substituted naphthalenediimides (cNDIs) have become the focus of considerable research for their potential applications in a variety of fields and to a number of organic technologies including catalysis and biomedicine, field-effect transistors, photovoltaics, and colorimetric sensors.[37–39] However, the extent of this research has generally been limited to the synthesis and application of di- and tetra-substituted cNDI derivatives, which have no magnetic dipole in the aromatic plane and therefore cannot be aligned by an electric field.

Unlike core di- and tetra-substituted NDIs, core mono-substituted NDIs (cmNDIs) do have a dipole in the aromatic plane and are therefore field-responsive (Scheme 1).



Scheme 1. Concept for cmNDIs with dipole in the aromatic plane.

Access to field-responsive cmNDIs invites the possibility for higher-order molecular control of DACLC alignment in the rotational orientation of NDI components in the plane normal to the direction of columnar stacking. In order to induce detectable rotational anisotropy, an external magnetic field, B_0, must be applied. Though the required field strength depends on the nature of the molecular components and is therefore unknown for a DACLC system, alignment of molecules with comparable dipoles has been achieved in the presence of fields of 4,000–5,000 gauss (0.4–0.5 T) in other systems;[40,41] fields of this strength can be achieved using an array of

commercially-available N52-grade NdFeB magnets (Fig. 14c).[40] In this way, the synthesis of cmNDIs may introduce an additional, controllable degree of freedom in the orientation of DACLCs, one which may be levered as a complement to columnar alignment toward higher-order opto-magnetic data storage.

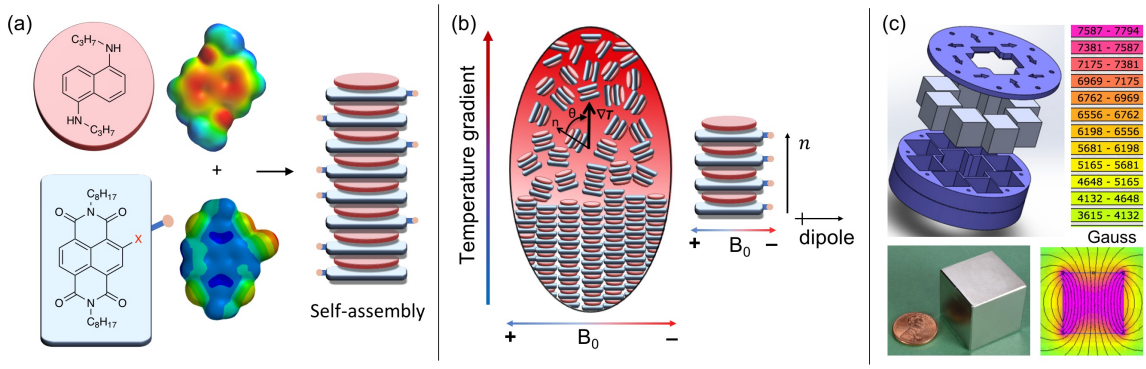
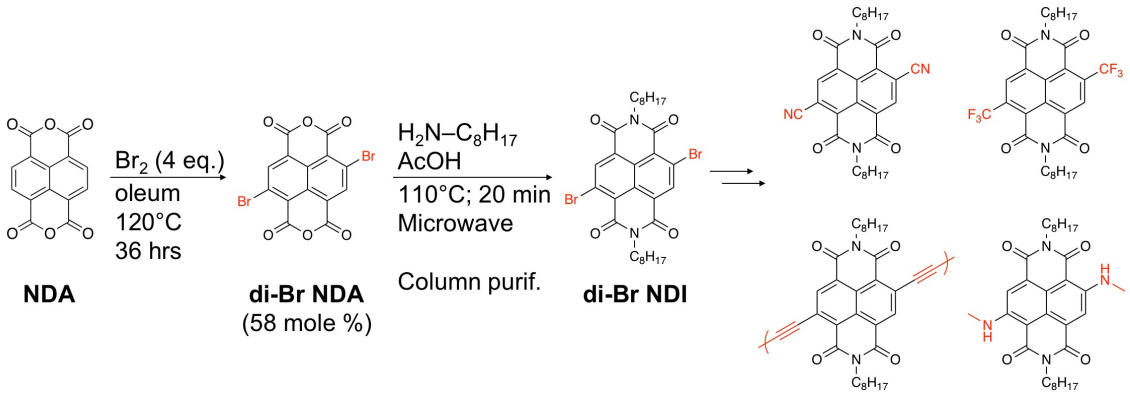


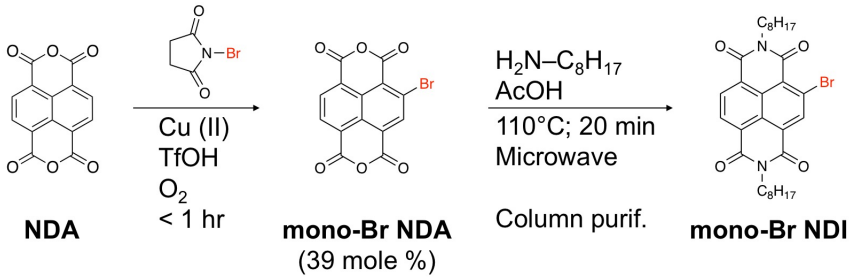
Figure 14. Concept for higher-order molecular control of DACLC materials, achieved via NDI core mono-substitution. (a) Meaningfully non-uniform rotational orientation of cmNDI components in DACLC columnar stacking, in virtue of cmNDI dipoles. (b) Alignment of cmNDI dipoles in the presence of an external magnetic field, B_0, for potential applications in high-density data storage. (c) Magnet array, built from commercially-available N52-grade NdFeB magnets, expected to be capable of generating a sufficiently large field to induce rotational anisotropy. (Upper-left image of (c) taken from Ref. [40], Fig. S7).

Core di-substituted NDIs can be synthesized via di-brominated NDA intermediates through traditional halogenation methods (Scheme 2).



Scheme 2. Core di-substituted NDI synthesis via core core-substituted Br-NDA intermediates.

However, the same halogenation methods that are effective in synthesis of di- and tetra-substituted NDIs have reaction kinetics that give only low yields of mono-substituted species (< 10%[42,43]). As a result, synthesis of cmNDI derivatives at appreciable yield requires alternative halogenation methods. Recent work[44] has demonstrated successful synthesis of mono-substituted Br-NDA intermediates (Scheme 3).



Scheme 3. Optimized reaction scheme, reported in Ref. [44], for selective synthesis of mono-Br NDI, toward cmNDIs.

These new protocols were carried out in order to generate a stock of mono-Br NDI that was subsequently used in synthesis routes[45,46] that have been used to demonstrate efficient core di-functionalization. These procedures were subtly modified to the address the unique synthetic challenges involved in core mono-functionalization, but are ultimately demonstrated to be effective in the synthesis of core-mono-substituted NDI derivatives, toward potential applications in field-responsive DACLCs.

Synthesis of Core Di-Functionalized NDI Aromatics

In order to first verify the efficacy of synthetic routes for core di-functionalization, di-Br NDA was selectively synthesized by traditional halogenation methods and reacted to form di-Br NDI by the same microwave synthesis step illustrated in Scheme 3;[43] purification of di-Br NDI species is demonstrated by ^{1}H NMR in Fig. 15a.

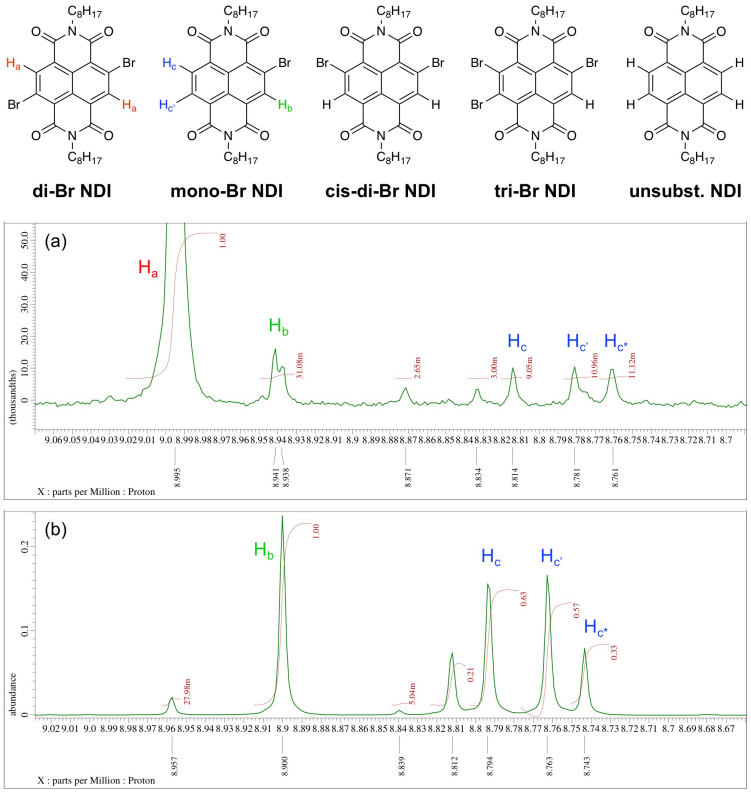
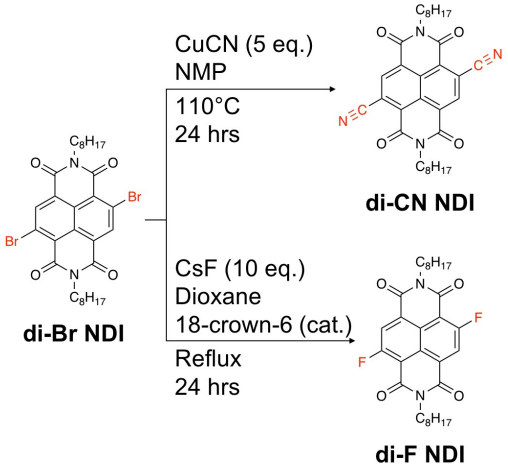


Figure 15. ^{1}H NMR spectra demonstrating selective synthesis and purification of (a) predominantly (> 90%) di-substituted and (b) predominantly mono-substituted Br-NDI.

Purified di-Br NDI was then used as starting material in synthesis routes reported in the literature for core di-substitution with cyano[45] and fluorine[46] functional groups (Scheme 4).



Scheme 4. General reaction schemes for NDI core di-substitution with cyano and fluorine functional groups, reported in Refs. [45] and [46], respectively.

A protocol reported by Ref. [45] was used to achieve core di-substitution with cyano functional groups using di-substituted Br-NDI starting material (Scheme 4). Unlike in the reported procedure, activated molecular sieves were used to remove water from the reaction, and product was precipitated in 1 M H_2SO_4. The efficacy of this modified procedure for di-substitution was verified by ^{1}H and ^{13}C NMR (Fig. 16).

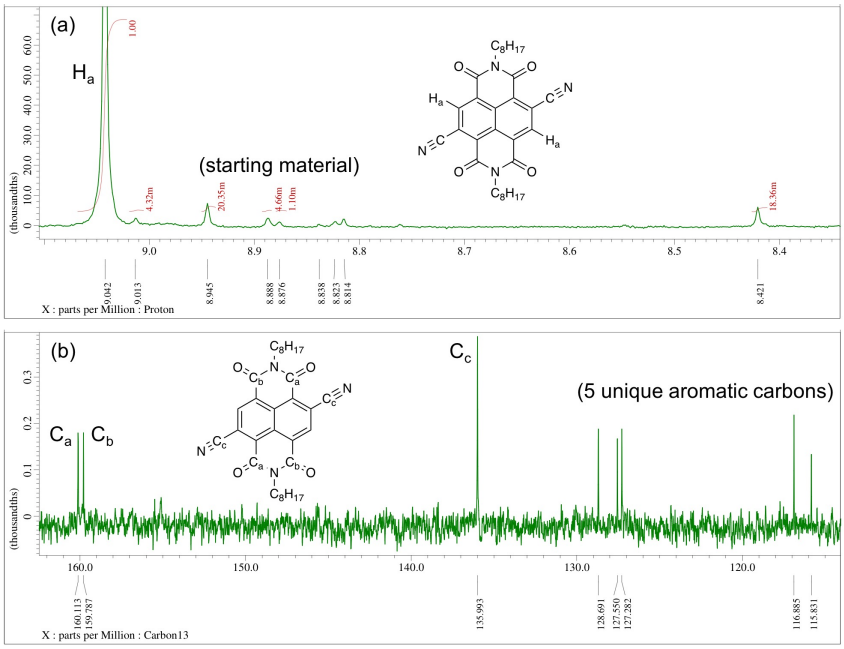


Figure 16. (a) ^{1}H and (b) ^{13}C NMR spectra of di-CN NDI. Structures verified by mass spectrometry.

Similarly, a protocol reported by Ref. [46] was used to react predominantly di-substituted Br-NDIs to demonstrate core-substitution with fluorine (Scheme 4); the results of one such reaction were verified by ^{1}H and ^{19}F NMR (Fig. 17).

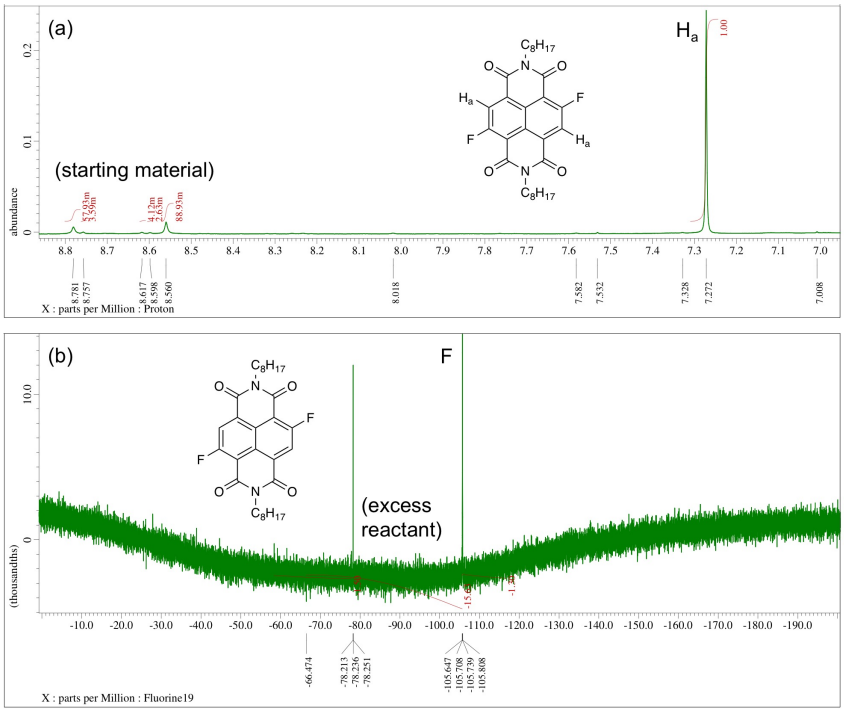


Figure 17. (a) ^{1}H and (b) ^{19}F NMR spectra of di-F NDI. Structure verified by IR spectroscopy.

Synthesis of Core Mono-Functionalized NDI Aromatics

Having established the efficacy of two literature protocols in the synthesis of core di-functionalized NDI derivatives, the protocols were carried out in the same way as before, but using mono-Br NDI as starting material, in order to achieve core mono-functionalization. Though the results of attempts to synthesize mono-F NDI were indeterminate, mono-CN NDI was successfully isolated.

Procedures reported in Ref. [44] were used to selectively synthesize mono-substituted Br-NDA, which was reacted to form mono-substituted Br-NDI by microwave synthesis (Scheme 3); results were verified by ^{1}H NMR (Fig. 15b). The modified procedure from Ref. [45] described above (Scheme 4) was used again with mono-Br NDI starting material in order to synthesize core mono-CN NDI; results were confirmed by ^{1}H and ^{13}C NMR (Fig. 18), MS, and IR (data not shown).

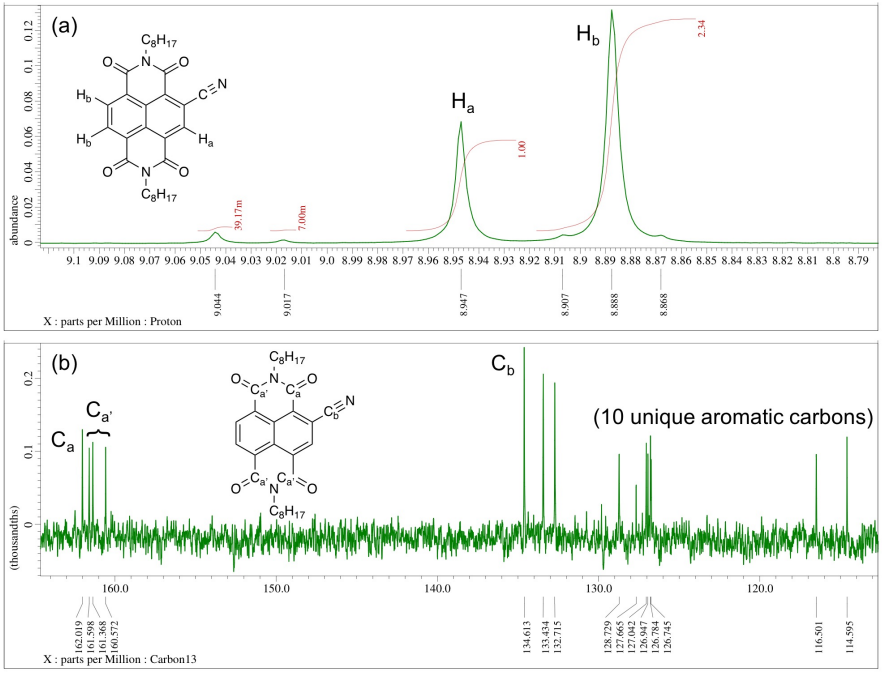


Figure 18. (a) ^{1}H and (b) ^{13}C NMR spectra of mono-CN NDI. Structure verified by MS and IR.

Conclusions

Successful synthesis of the core mono-CN NDI derivative demonstrated here represents a significant step toward the realization of field-responsive DACLCs and the application of DACLC materials to high-density opto-magnetic data storage. Future work will surely incorporate this and other core mono-functionalized NDI derivatives into DACLC mixtures and design methods to control and probe this additional degree of freedom in molecular alignment.

Experimental

All reagents and solvents were obtained from Sigma-Aldrich or Fischer Scientific and used as purchased without additional purification. ^{1}H and ^{13}C NMR spectra were collected using a JOEL 400 MHz spectrometer at 25°C. The magnet array illustrated in Fig. 14c was assembled with four 1" cube N52-grade NdFeB magnets (K&J Magnetics, BX0X0X0-N52) in a frame that was 3D-printed from a design published in Ref. [40].

References

[1] Reczek J.J., Naphthalenediimide in Modular Columnar Liquid Crystals: Key Component of Donor–Acceptor Columnar Liquid Crystals. In *Naphthalenediimide and Its Congeners*, Cambridge: Royal Society of Chemistry, **2017**, 90–115.

[2] Bisoyi H.K., Li Q., *Prog. Mater. Sci. 104*, **2019**, 1–52. (2019).

[3] Charlet E., Grelet E., *Phys. Rev. E 78*(4), **2008**, 041707.

[4] Leight K.R., Esarey B.E., Murray A.E., Reczek J.J., *Chem. Mater. 24*(17), **2012**, 3318–28.

[5] Van Winkle M., Scrymgeour D.A., Kaehr B., Reczek J.J., *Adv. Mater. 30*(20), **2018**, 1706787.

[6] Gray Bé A., Tran C., Sechrist R., Reczek J.J., *Org. Lett. 17*(19), **2015**, 4834–7.

[7] Martinez C.R., Iverson B.L., *Chem. Sci. 3*(7), **2012**, 2191–201.

[8] Wang J.Y., Yan J., Ding L., Ma Y., Pei J., *Adv. Funct. Mater. 19*(11), **2009**, 1746–52.

[9] Damian I., *Buletinul Stiintific al Universitatii Politehnica din Timisoara 49*(63), **2004**, 107–11.

[10] Krása J., Jiřička J. Lokajíček M., *Phys. Rev. E 48*(4), **1993**, 3184–6.

[11] Sampson A., Nelson J., Strauss K., & Ceze L., *ACM T. Comput. Sys. 32*(3), **2014**, 1–23.

[12] Shannon C.E., *Bell Sys. Tech. J. 27*(3), **1948**, 379–423.

[13] Gao H., Bradley, J.R., *Spat. Stat. 31*, **2019**, 100357.

[14] Yeh P., Gu C., *Optics of Liquid Crystal Displays (2nd ed.)*, Sussex: John Wiley & Sons, **2009**.

[15] Lueder E., *Liquid Crystal Displays: Addressing Schemes and Electro-Optical Effects*, Sussex: John Wiley & Sons, **2010**.

[16] Ozaki R., Ozaki M., Yoshino K., *Crystals 5*(3), **2015**, 394–404.

[17] Alegria F.C., *Measurement 42*(5), **2009**, 748–56.

[18] Marquez J.P., *Int. J. Solids Struct.* 43(21), **2006**, 6413–23.

[19] Dhar L., *ACS Cent. Sci. 5*(5), **2019**, 753–4.

[20] Cafferty B.J., Ten A.S., Fink M.J., Morey S., Preston D.J., Mrksich M., Whitesides G.M., *ACS Cent. Sci. 5*(5), **2019**, 911–6.

[21] Ceze L., Nivala J., Strauss K., *Nat. Rev. Genet. 20*(8), **2019**, 456–466.

[22] Jang J.S., Shin D.H., *Opt. Lett. 26*(22), **2001**, 1797–9.

[23] Reményi J., Várhegyi P., Domján L., Koppa P., Lõrincz E., *Appl. Opt. 42*(17), **2003**, 3428–34.

[24] John R., Joseph J., Singh K., *Opt. Rev. 12*(3), **2005**, 155–60.

[25] Tu H.Y., Cheng C.J., Chen M.L., *J. Opt. A 6*(6), **2004**, 524.

[26] Matoba O., Javidi B., *Appl. Opt. 43*(14), **2004**, 2915–9.

[27] Eskicioglu A., Litwin L., *IEEE Potentials 20*(1), **2001**, 36–8.

[28] Mogensen P.C., Glückstad J.A., *Opt. Commun. 173*(1–6), **2000**, 177–83.

[29] Shannon C.E., *Bell Sys. Tech. J. 28*(4), **1949**, 656–715.

[30] Alvey P.M., Reczek J.J., Lynch, V., Iverson B.L., *J. Org. Chem. 75*(22), **2010**, 7682–90.

[31] Reczek J.J., Villazor K.R., Lynch V., Swager T.M., Iverson B.L., *J. Am. Chem. Soc. 128*(44), **2006**, 7995–8002.

[32] Vanderheiden G., Reid L.G., Cooper M., Caldwell B., Contrast ratio, *Web Content Accessibility Guidelines (WCAG) 2.0, WWW Consortium (W3C)*, **2008**.

[33] Hou X., Chen H., Zhang Z., Wang S., Zhou, P., *Adv. Electron. Mater. 5*(9), **2019**, 1800944.

[34] Weller D., Parker G., Mosendz O., Lyberatos A., Mitin D., Safonova N.Y., Albrecht M., *J. Vac. Sci. Technol. B 34*(6), **2016**, 060801.

[35] Hono K., Takahashi Y., Ju G., Thiele J., Ajan A., Yang X., Ruiz R., Wan L., *MRS Bull. 43*(2), **2018**, 93–9.

[36] Kief M.T., Victora R.H., *MRS Bull. 43*(2), **2018**, 87–92.

[37] Kobaisi M.A., Bhosale S.V., Latham K., Raynor A.M., Bhosale S.V., *Chem. Rev. 116*(19), **2016**, 11685–796.

[38] Suraru S.L., Wüerthner F., *Angew. Chem Int. Ed. 53*(29), **2014**, 7428–48.

[39] Suseela Y.V., Regioselective bromiantion of naphthalenetetracarboxylic dianhydride and synthesizing its derivatives for DNA binding studies, *Doctoral dissertation, Jawaharlal Nehru Centre for Advanced Scientific Research*, **2014**.

[40] Guillamat P., Ignes-Mullol J., Sagues F., *PNAS 113*(20), **2016**, 5498–502.

[41] Shin J., Kang M., Tsai T., Leal C., Bruann P.V., Cahill D.G., *ACS Macro Lett. 5*(8), **2016**, 955–60.

[42] Jones B.A., Facchetti A., Marks T.J., Wasielewski M.R., *Chem. Mater. 19*(11), **2007**, 2703–5.

[43] Thompson A.C., Grimm H.M., Gray Bé A., McKnight K.J., Reczek J.J., *Synth. Comm. 45*(9), **2015**, 1127–36.

[44] Santana C.G., Zeng J.G., Triboni E.R., Reczek J.J., Investigation of selective and efficient mono-bromination of 1,4,5,8- naphthalene dianhydride by acid-mediated copper C–H activation. (Pending review)

[45] Schmidt S.B., Biskup T., Jiao X., McNeill C.R., Sommer M., *J. Mater. Chem. C 7*(15), **2019**, 4466–74.

[46] Yuan Z., Ma Y., Geßner T., Li M., Chen L., Eustachi M., Weitz R.T., Li C., Müllen K., *Org. Lett. 18*(3), **2016**, 456–9.

[47] Händel P., *Proceedings of the IEEE Nordic Signal Processing Symposium*, Sweden, **2000**.

Acknowledgements

Dr. Joe Reczek

Dr. Bryan Kaehr

Dr. Kyle Tsai

Dr. Jordan Fantini

Dr. Jordan Katz

Dr. Tim Troyer

Madeline Van Winkle

Carrie Dietz

Gaby Pleitez Gomez

Dave Burdick

Phil Waite

Cathy Romei

Denison University Research Fund

Denison University Department of Chemistry & Biochemistry

Sandia National Labs

Appendix A. Laser-alignment setup, laser module control, and optimized alignment settings

The custom laser-alignment setup housed in Ebaugh Laboratories at Denison University is diagrammed in Fig. A1 and described at length below.

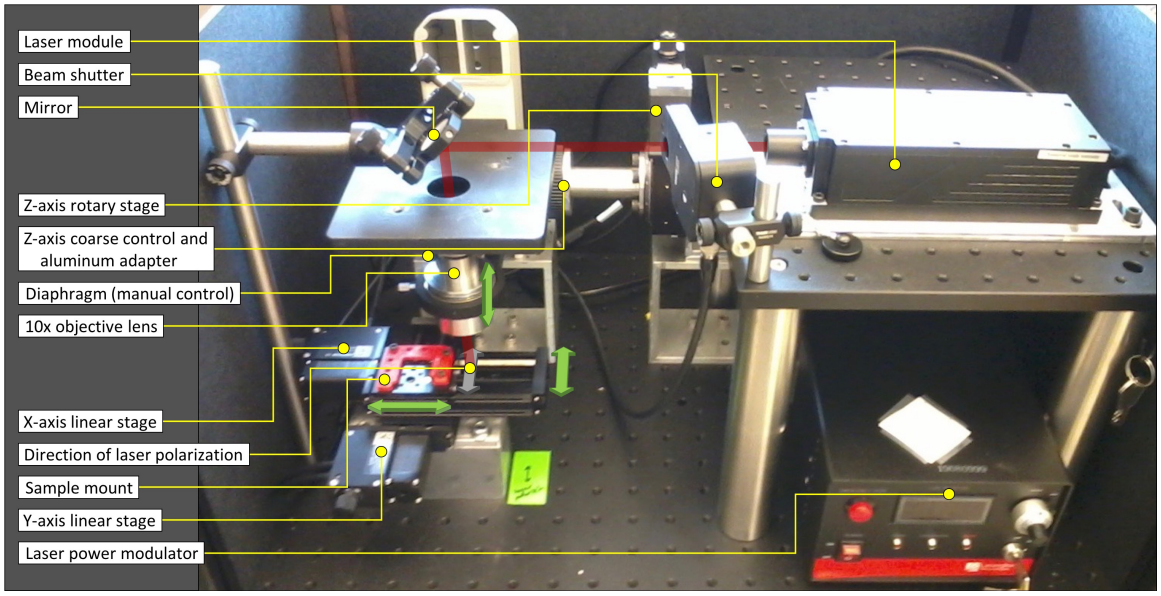


Figure A1. Annotated diagram of in-house laser writing setup. Green arrows denote axes of motion, programmatically controlled using motorized linear and rotary stages.

The beam emitted from a 450 mW, 671 nm laser module (Laserglow Technologies, Toronto, CA) is projected horizontally, strikes a mirror, and is reflected vertically down. The beam can be interrupted by a shutter (SH05, Thorlabs, Newton, NJ) mounted between the laser module and the mirror; the shutter is controlled programmatically by modulating the current delivered to it by a Keithley 2400 SourceMeter (Tektronix, Beaverton, OR; set to 0.4 A (source) at 21.0 V (compliance) to close). A 10x microscope objective is suspended vertically in-line with the beam, in the optic holder of a modified student microscope, above two motorized linear stages (T-LSM050A, Zaber Technologies, Vancouver, CA) that move the film sample under the beam. The modified microscope's coarse control axle is joined to a motorized rotary stage (X-RSW60A, Zaber

Technologies, Vancouver, CA) via an aluminum adaptor shaft, in order to allow for programmatic control of lens height (z) above the film sample, and hence beam focus.

The two linear stages are mounted one on top of the other in the plane of the ground and oriented along different positional axes such that the upper stage's platform has full range of motion in two dimensions; they are positioned beneath the microscope objective such that this range of motion allows the focused beam to strike any point in much of the central area of the upper platform. A DACLC film sample prepared on a 1-inch microscope slide is mounted on the upper of the two motorized linear stages, in a custom, 3D-printed sample mount.

The goals of this research have required an extensive overhaul of the existing in-house laser-writing setup in order to dramatically expand its write capabilities. Among the major changes to this setup that were made over the course of this project were: the addition of an adjustable-height platform (the body of a student microscope stage) to suspend a focusing optic at known height (z) above the sample; attachment of a motorized rotary stage to the coarse control axle of this platform, via an aluminum adaptor shaft, in order to achieve precise programmatic control of z and hence beam focus; integration and analysis of a 450mW, 671nm laser module (Fig. A4); and incorporation of a beam shutter, programmatically controlled using the Keithley 2400 SourceMeter.

As part of this work, I have designed a Python library to facilitate programmatic control of the motorized stages and beam shutter, thereby making complex laser-write patterning possible in-house. The technical details of this library are available in Appx. D, but one key feature is the ability to call any of fourteen pre-defined commands to align or disalign regions of the film sample along standard paths (e.g., parallel lines or circle), by simply specify parameters like start and end position and write speed (Fig. A2).

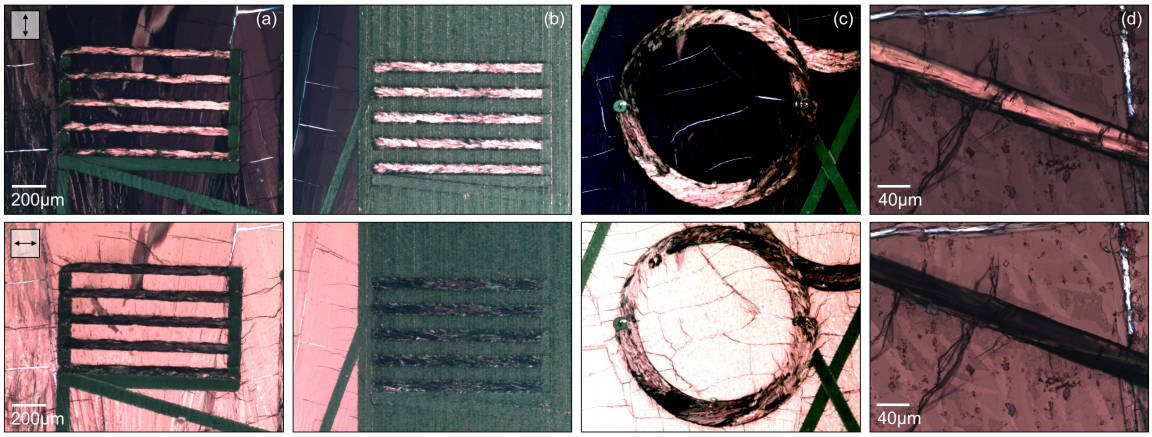


Figure A2. Parametrized write commands allow facile alignment patterning. LPL-M images (angle of LPL indicated by the arrow at the upper left of the row) of: (a) over an orthogonally-aligned film background, (b) over an isotropic background, (c) in a circle, and (d) at micron-scale. (a,b) written by write_parallel_lines_gap(…), (c) by write_circle(…), and (d) by write_line(…) (Appx. D).

This custom library is also able to model write paths graphically and predict runtimes before writing (Fig. A3), both of which features greatly improve the facility, efficiency, and economy of laser-alignment experiments. Very many more important details of this library are available in the README.md and in code annotations, all available in Appx. D.

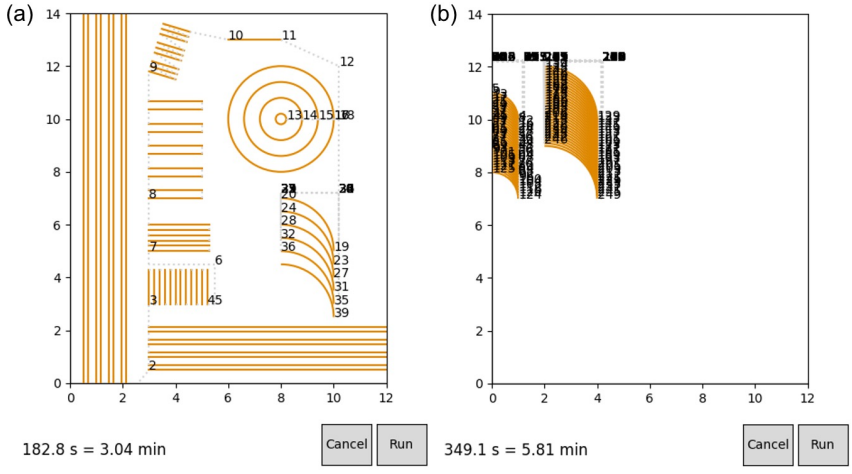


Figure A3. Sample outputs of path modeling and runtime prediction. Path (a) is comprised of various elementary parametrized write-commands built into the library; (b) traces overlapping quarter-turns at two different radii, for an alignment pattern with potential utility in applications of DACLC films as variable-index of refraction and waveguide media (not investigated).[3]

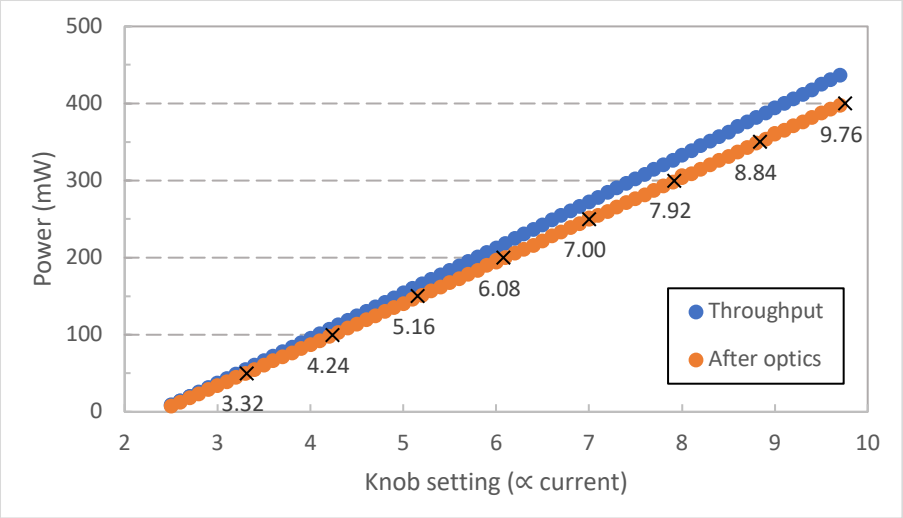


Figure A4. Laser power before and after optics is related to power-module knob setting using Thorlabs S302C power meter. Knob settings for 50 mW increments are labeled for reference.

Determining the parameters for optimized laser-alignment using the in-house laser setup is an ongoing challenge. The complex balance between sample thickness, beam spot size and focal distance, laser power, and write speed proves to be an elusive one. The two approaches I have tried to implement have been (i) to painstakingly enumerate incremental adjustments of each of these co-variates so as to quasi-span the multi-dimensional space and thereby derive an explicit relationship between variables, and (ii) to attempt uninformed combinations and pray. Only the latter approach bore fruit (Table A1).

Table A1. Optimized laser-alignment parameters, as of 28 Feb 2020. Stage height (z) measured arbitrarily from the surface of the upper linear stage to the upper surface of the z-control stage platform (just below the mirror); conversions to meaningful values used by the rotary stage are done automatically in execute_commands.py (Appx. D). (* = still imperfect.)

Write condition	**Stage height, z (mm)**	**Speed, s (mm/s)**	**After-optics laser power, P (mW)**
Align over aligned background (wide)	147.0	0.4–0.8	350
(narrow)	144.8	0.6	120
Align over isotropic background*	144.0	0.6–0.8	210
Isotropic (wide)	145.0	8.0	150

Appendix B. Four-parameter sine curve fitting ("splining")

Equation 4 relating I_{obs} to θ_{LPL} and θ_w can be re-expressed as Equation B1 using a double-angle trigonometric identity for cos^2 and the identity that $cosx = sin(x + 90°)$:

$$I_{obs} = \frac{1}{2}\, k\, I_0\, sin(2(\theta_{LPL} - \theta_w + 90°) + 90°) + \frac{1}{2}\, k\, I_0 + b \tag{B1}$$

This equation is of the same form as Equation B2, where $x = \theta_{LPL}$.

$$y = asin(\omega x + \varphi) + C \tag{B2}$$

, which can be re-expressed using an angle-sum identity for sin:

$$= a(sin\varphi\ cos\omega x + cos\varphi\ sin\omega x) + C \tag{B3}$$

$$= A\ cos\omega x + B\ sin\omega x + C \quad , \quad A = a\ sin\varphi \text{ and } B = a\ cos\varphi \tag{B4}$$

It follows from Equation B1 that $\omega = 2$, but this is also consistent with dichroic behavior of aligned regions being periodic with $\mathrm{T} = 180°$. The assumption that $\omega = 2$ allows a single, determinate solution (i.e., eliminates aliasing) for the remaining unknown parameters a, φ, and C, with only three measurements of I_{obs} at unique θ_{LPL}.

For multiple measurements of y at multiple values of x, Equation B4 gives:[47]

$$\vec{y} = \mathrm{D}(A \quad B \quad C)^T \quad , \quad \mathrm{D} = \begin{pmatrix} cos\omega x_1 & sin\omega x_1 & 1 \\ cos\omega x_2 & sin\omega x_2 & 1 \\ \vdots & \vdots & \vdots \\ cos\omega x_n & sin\omega x_n & 1 \end{pmatrix} \tag{B5}$$

The least-squares solution to this system of linear equations is given by Equation B6, for $n > 3$, and the exact solution by Equation B7, for $n = 3$.[47]

$$(A \quad \widehat{B} \quad C)^T = (\mathrm{D}^T\mathrm{D})^{-1}\mathrm{D}^T\vec{y} \tag{B6}$$

$$(A \quad B \quad C)^T = \mathrm{D}^{-1}\vec{y} \tag{B7}$$

where parameters a and φ are given, in both cases, by:[47]

$$a = (A^2 + B^2)^{1/2} \tag{B8}$$

$$\varphi = tan^{-1}\frac{A}{B} \tag{B9}$$

Substituting into Equation B1,

$$\theta_w \approx \theta_{w,fit} = \frac{1}{2}(270° - \varphi) \tag{B10}$$

Appendix C. Dyad cipher and readout accuracy (cf. Fig. 11)

Table C1. Dyad cipher. Note that each character value corresponds redundantly to six dyads, and that each of six dyads corresponds uniquely to a single character value. Dyads used in the example message, "ALICE'S_MESSAGE_TO_BOB" (Fig. 11b), are highlighted. Redundancies built into the cipher were not leveraged to obscure any erroneous readout.

Character	Dyad 1	Dyad 2	Dyad 3	Dyad 4	Dyad 5	Dyad 6
A	20, 140	10, 140	0, 60	40, 130	100, 10	110, 20
B	10, 90	140, 50	70, 10	80, 0	90, 10	170, 70
C	50, 140	160, 70	90, 130	120, 10	10, 110	170, 140
D	60, 140	50, 80	70, 50	160, 50	130, 90	150, 100
E	50, 70	140, 140	50, 90	80, 120	10, 100	40, 30
F	10, 130	20, 30	10, 30	140, 80	160, 140	90, 120
G	140, 120	60, 120	150, 0	120, 110	80, 150	130, 10
H	10, 150	80, 70	50, 10	160, 120	90, 70	160, 150
I	170, 80	130, 80	170, 170	170, 20	110, 40	40, 0
J	50, 0	80, 10	60, 100	100, 90	170, 120	60, 30
K	70, 0	130, 60	90, 20	160, 170	10, 160	0, 50
L	120, 30	100, 130	20, 40	120, 100	60, 110	20, 100
M	30, 0	80, 30	0, 20	50, 50	90, 30	170, 130
N	70, 90	140, 160	30, 130	0, 150	70, 130	70, 80
O	80, 80	150, 50	150, 140	120, 80	0, 10	90, 110
P	30, 140	170, 30	60, 60	0, 100	50, 100	20, 170
Q	100, 30	110, 170	40, 40	150, 90	110, 120	130, 0
R	30, 50	120, 0	120, 20	60, 40	110, 150	140, 10
S	120, 170	160, 90	150, 40	140, 170	160, 0	130, 110
T	40, 10	40, 50	30, 170	30, 70	30, 150	120, 120
U	70, 20	140, 130	120, 150	130, 70	50, 160	70, 70
V	110, 130	100, 150	120, 40	50, 120	20, 10	130, 30
W	110, 60	70, 160	40, 80	90, 90	140, 60	100, 60
X	80, 140	110, 30	40, 160	20, 50	80, 170	170, 150
Y	80, 60	0, 160	130, 40	120, 60	70, 40	170, 100
Z	110, 140	160, 110	40, 70	150, 130	0, 130	100, 120
_A	130, 120	140, 20	80, 110	20, 90	160, 60	80, 100

_B	10, 170	170, 110	60, 70	40, 90	170, 160	30, 160
_C	150, 20	30, 10	150, 120	160, 100	30, 80	130, 50
_D	100, 160	100, 170	70, 170	170, 50	140, 0	160, 160
_E	50, 130	50, 20	110, 50	60, 80	130, 170	0, 70
_F	50, 60	90, 60	160, 10	130, 130	70, 110	40, 100
_G	30, 40	110, 110	120, 140	50, 150	120, 160	150, 150
_H	40, 60	150, 160	70, 150	110, 100	140, 30	100, 70
_I	0, 40	130, 20	20, 110	90, 160	120, 130	90, 140
_J	150, 10	50, 170	20, 160	30, 30	10, 20	170, 40
_K	10, 60	140, 40	20, 150	140, 150	160, 130	100, 40
_L	60, 0	20, 0	150, 80	30, 90	10, 120	20, 120
_M	100, 0	30, 120	40, 110	70, 120	110, 90	40, 170
_N	90, 150	10, 80	150, 110	100, 20	60, 10	30, 60
_O	40, 150	20, 130	40, 140	10, 70	20, 60	60, 150
_P	100, 110	150, 30	50, 30	100, 80	80, 160	10, 0
_Q	50, 110	110, 10	20, 70	140, 110	70, 30	0, 140
_R	0, 0	60, 160	170, 10	110, 0	170, 0	90, 100
_S	110, 160	10, 40	70, 140	150, 170	60, 170	70, 100
_T	70, 60	160, 20	100, 50	80, 40	120, 70	20, 80
_U	140, 100	60, 50	0, 30	0, 170	40, 120	120, 50
_V	130, 100	40, 20	100, 100	10, 50	50, 40	110, 70
_W	90, 40	80, 90	20, 20	90, 50	0, 120	0, 80
_X	30, 110	10, 10	30, 20	60, 130	130, 160	0, 90
_Y	0, 110	170, 60	160, 80	120, 90	140, 70	60, 90
_Z	110, 80	140, 90	160, 40	30, 100	150, 60	80, 130
'S	100, 140	60, 20	90, 170	170, 90	130, 150	150, 70
'T	90, 80	130, 140	80, 20	160, 30	80, 50	90, 0

Table C2. Written angles and readout accuracy using base-10 and base-19 encoding schemes. Using a base-10 encoding scheme (such that exact readout angle is rounded to the nearest 20° increment; in this case, because the encoding scheme is defined in 10° increments, readout is considered to be "correct" if it falls within 10° of the actual written angle), readout of the encoded message is perfectly accurate. Using a base-19 encoding scheme (such that exact readout angle is rounded to the nearest 10° increment), readout is near-perfect, with one error in 36 readings (highlighted). Redundancies built into the cipher (Table C1) were not leveraged to obscure any erroneous readout.

Written Angle (°)

0	20	130	160	50	60
60	40	80	70	90	20
30	10	140	160	110	130
120	100	170	0	20	10
50	80	150	30	90	140
70	40	140	160	110	50

Angle Readout (°)

0.3	18.3	127.0	165.6	48.1	58.6
59.4	41.0	79.0	68.0	88.1	18.1
26.3	8.2	141.5	161.5	109.8	129.1
117.3	98.5	174.5	179.8	17.9	8.2
46.9	77.5	149.0	26.5	91.2	143.5
72.4	39.4	140.3	164.8	106.8	47.8

Appendix D. Compiled Python scripts used for programmatic stage control

```
README.md

AUTHOR:     Harper O. W. Wallace
DATE:       12 Apr 2020

________

DESCRIPTION OF STAGES SETUP:

This library executes move commands for Zaber Technologies rotary (1 x
X-RSW60A) and linear (2 x T-LSM050A) stages, all using binary encoding,
and it supervises voltage and current control of Keithley 2400
SourceMeter to open and close a Thorlabs SH05 beam shutter.

The two linear stages are mounted in the plane of the ground (one atop
the other) and oriented orthogonally such that, from the perspective of
the upper stage's platform, one stage controls the "x-coordinate of
position," and the other controls the "y-coordinate." Film samples
prepared on 1-inch microscope slides can be mounted on the upper stage
using a 3D-printed slide-holder (slide_holder.stl) that I designed to be
compatible with Zaber T-LSM050A stages.

The upper stage's platform has a range of motion such that the laser
beam emitted from a 671nm, 450mW Laserglow laser module can strike any
point in much of the central area of the upper platform, and therefore
any point on a mounted film sample. The beam is projected horizontally
above the stages, strikes a mirror and reflects vertically down, and is
focused through a 10x microscope objective; it can be interrupted
programmatically by means of a Thorlabs SH05 beam shutter (mounted
between source and mirror), which is operated by modulating current from
the Keithley 2400 SourceMeter (cf. keithley_handler.py). The 10x
microscope objective is suspended in the beam path (vertically; after
reflection) and above the film sample, in the optic holder of a modified
student microscope whose coarse control is connected to the rotary stage
to allow for programmatic control of the distance between lens and
sample (corresponds to "z" in execute_commands.py).

"Writing" (used below and in .py files) entails moving a thin film of
thermoresponsive materials (donor-acceptor columnar liquid crystals) on
the stages beneath the focused beam to control their columnar alignment.
For this particular application, particular considerations must be made
to speed, power, and focal distance, but for the purposes of this
overview, it is sufficient to say that the path that the beam follows
corresponds to the pattern written to the sample.


________

execute_commands.py:

Parametrized move commands allow the user to write a single line, write
parallel lines, write part of a circle, write a full circle, wipe a
region, outline a region, and home all stages merely by specifying
relevant parameters (e.g., circle center, radius, and write speed); many
of these write functions are parametrized in several different ways in
order to simplify writing slight variations on the same type of pattern
(e.g., horizontal lines that span the entire sample region, v. vertical
lines that span a specified sub-region, v. diagonal lines that are
separated by a specified perpendicular distance). These commands and
their arguments are detailed in execute_commands.py.

```

```
62.
63.  coordinates.py:
64.
65.  A Cartesian coordinate system keeps track of sample boundaries and write
66.  command positions within them. This system confers particular advantages
67.  in allowing the user to keep track of previously-written and new
68.  patterns in a systematic way, and also in conserving area on sample
69.  films in order to maximize their usefulness between wipes.
70.
71.  The coordinate system is defined by two critical parameters (1–2) and
72.  one very helpful one (3):
73.  1. **GLOBAL_O** (*V2*) = "global origin," or the x- and y- stage
74.     positions when the laser beam is focused on the bottom-left corner of
75.     a square sample film
76.  2. **TR** (*V2*) = "top right," etc.
77.  3. **LOCAL_O** (*V2*) = "local origin," effectively a pointer vector
78.     from the origin, which allows the coordinate system to be sub-defined
79.     for a given write command, without needing to redefine GLOBAL_O or TR.
80.
81.  Although GLOBAL_O stands for "global origin," TR is actually another
82.  global origin and is often the more critical of the two to determine
83.  accurately, since it is ends up defining the apparent "bottom left"
84.  origin position relative to which moves are made, once the coordinate
85.  system is inverted to account for inversion on viewing under a microscope.
86.
87.  The user can define values for GLOBAL_O and TR using
88.  POSITION_GETTER_MODE (in execute_commands.py), which allows her to take
89.  manual control of the stages (using control knobs on the stages
90.  themselves) to adjust their position until they reach a desired position
91.  on the sample (e.g., GLOBAL_O or TR). At this point, the user can press
92.  the 'p' key on the keyboard to print the position of the stages to the
93.  console, so they may be recorded, e.g., to define GLOBAL_O and TR for a
94.  new sample (press 'esc' once these values have been recorded).
95.
96.  Units are millimeters, but neither class of Zaber stages uses this unit
97.  explicitly; instead, they use microsteps (which I call "data" in
98.  conversion methods, because those values are what ends up getting sent
99.  to the devices). Conversion factors between distance and speed are not
100. intuitive, and they are different for the rotary and linear stages.
101. Understanding how these units are interconverted is key to resolving
102. issues where the stages do not behave as expected, which can happen.
103. Consult [1] for a detailed explanation of Zaber's units and conversions.
104.
105. This coordinate system will be more useful if it is specified for EACH
106. film sample. To facilitate maintaining these boundary positions for
107. different samples, I have designed (and installed, in our setup in
108. Ebaugh Labs at Denison University) a 3D-printable 1-inch glass
109. microscope slide holder that can be mounted to the Zaber T-LSM050A stage
110. so that this coordinate system is approximately (~0.01mm) conserved on
111. removal and replacement of the sample (cf. slide_holder.stl). Values for
112. GLOBAL_O and TR should be stored in sample-specific .txt files (cf.
113. samples/_example.txt), which are read by execute_commands.py
114.
115.
116. mapping_handler.py
117.
118. A new and experimental feature of this library is move mapping, which
119. theoretically allows the user to simulate how write commands will turn
120. out in order to correct any errors before running them on a sample; as a
121. sort of extension of the coordinate system itself, this feature too is
122. meant to help conserve space on the film and maximize usefulness between
```

```
writes. Move mapping will be particularly advantageous for writing
multi-step patterns that will be necessary to generate complex
diffraction gratings for Fourier projection and holographic images. As I
explain in more detail in the header of mapping_handler.py, rendering
hasn't been rigorously tested yet; but it verifying predicted paths for
different commands should be straightforward, and a procedure is
outlined in the header of mapping_handler.py.

mapping_handler.py consults the .txt data file corresponding to the
relevant film sample to gather historical write data and defined global
origins. These film sample files must be made available for each sample
(at the path specified in execute_commands.py), and in the format
specified in samples/_example.txt and samples/_template.txt. Notes on
how film sample files are used are offered in execute_commands.py, but
here are some notes on syntax:
- Film sample files are interpreted in such a way that is compatible
  with the actual Python syntax for calling these commands. This allows
  the user to copy and paste commands between sample .txt files and the
  manual command-calling section of execute_commands.py (rather than
  having to reformat from some other input format). It makes using Move
  Mapping much more convenient (if only because you have to be familiar
  with only one command-calling syntax), but you should note that
  parsing out varied-type (e.g., int, float, V2, arrays) arguments from
  a string in a text file is nontrivial: it is critical that you use
  exactly the format that I describe in samples/_template.txt, and even
  so it is also possible that there are some text-parsing bugs that you
  will need to resolve.
- The interpreter will respect Python-syntax line comments (i.e., "#
  ..."), so I'd recommend commenting in information about laser power so
  you can keep track of it.
- Film sample data files use a special symbol ("## ...") to mark section
  divisions between already-written commands, new commands, and
  references; this symbol should not appear elsewhere than those three
  places (cf. samples/_template.txt).

Limitations of mapping_handler.py:
- If you want to modify a command method (e.g., write_line(...)), then
  you'll have to make sure that the corresponding section in
  mapping_handler.py is updated; if you want to add a new command
  method, then you'll need to define one in mapping_handler.py for it to
  be rendered.
- It assumes that your frame of reference for writing is based on the
  microscope image, which is an inversion of how patterns are actually
  written to the sample (i.e., assumes INVERT = True, cf.
  execute_commands.py). This means that if you change the value of
  INVERT in execute_commands.py, then the preview rendered by
  mapping_handler.py will be inverted.
- It hasn't been designed to shift shift historical information if you
  redefine a sample's GLOBAL_O or TR.


________

EXTREMELY IMPORTANT NOTES:
1. DO NOT LET THE OBJECTIVE LENS HIT THE SAMPLE HOLDER SCREWS:
   Although it is possible to programmatically set a minimum height for
   the rotary stage to alleviate concerns like this, unfortunately the
   screws are sufficiently elevated above the sample that the focal
   distance would probably be too large to write at modest power, if you
   were to try that (rather than doing it that way, I have set the
   minimum position such that the aluminum jacket for the objective lens
```

```
    will not collide with the screws, so that the linear stages should
    not be damaged in a collision). Therefore it is EXTREMELY IMPORTANT
    that you move the stages such that the objective lens is someplace
    within the sample region, THEN lower the objective lens to the
    correct focal distance. This is done automatically in the
    parametrized move commands defined below, but if you define any more,
    or if you call move_to(...), then you must consider stage height.

 2. DO NOT MELT THE BEAM SHUTTER BY LEAVING THE LASER BEAM ON IT TOO
    LONG:
    If the laser spot is on the shutter leaves for too long, they will
    deform and the diaphragm will no longer be able to open and close.
    For the ThorLabs SH05 beam shutter we are using, about ten seconds is
    safe given the wavelength, spot size, and highest throughput of our
    laser (according to a ThorLabs Application Technician). This can be
    an inconvenience when you would like to move around the sample film
    without dragging an isotropic line behind you, e.g., and so it means
    that you may have to find creative ways to organize writing to
    minimize the time that the laser needs to be effectively "off";
    alternatively, you may find that raising the objective lens
    sufficiently high will diffuse the beam such that you can move the
    sample without aligning/disaligning; as a last resort, you may need
    to TURN THE LASER OFF MANUALLY in certain circumstances, rather than
    relying on the beam shutter.

 3. DO NOT MOVE THE ROTARY STAGE AT TOO GREAT A SPEED OR ACCELERATION:
    Stage accelerations are defined in execute_commands.py
    (define_operating_constants), and I strongly recommend that you not
    alter them. The primary reason that you should not increase the
    rotary stage acceleration is that it'd be a huge problem if you bent
    or snapped the pin that runs through the coarse control of the
    microscope stage and which the rotary stage is rotating to control
    stage height; if this pin twists too abruptly (and I have no idea
    what the damage threshold is), you're probably going to need to build
    a new z-control stage and a new control adapter for the rotary stage,
    and you may do direct damage to the rotary stage and/or the object
    lens.
    It's also important not to let the rotary stage move too fast: though
    it is possible with the linear stages, incorrect moves of the rotary
    stage are much more likely to cause actual damage (to the objective
    lens, to the sample, to the linear stages, and to itself). I've set
    an upper bound on speed so that the rotary stage moves slowly; this
    allows you to manually stop it (by pressing in once on the manual
    control knob) if it looks like it's heading for dangerous territory.

 4. YOU CAN INTERRUPT PROGRAMMED STAGE MOVES BY PRESSING SHIFT-CTRL-C
    I strongly recommend playing around with sending commands and
    interrupting them with Shift-Ctrl-C so you know. One notable quirk is
    that, if you interrupt writing a circle, the most recent move_vel
    command stands, and so one or both of the linear stages may just keep
    going until it is fully extended; obviously, this is a big concern if
    the tip of the objective lens is below the level of the sample holder
    screws.

 5. PUTTING A Nd MAGNET ARRAY DIRECTLY ON THE LINEAR STAGES INTERFERES
    WITH THEIR OPERATION
    They sort of spazzed out there, for a while. If you are interested in
    aligning in a strong B field, then make sure that you have the magnet
    array sufficiently far away from the stages (e.g., mounted above,
    with a several-inch spacer).

```

```
245.
246. TROUBLESHOOTING:
247. - In Zaber Console, if pressing the "Stop" button does not update a
248.   stage's current position, it's probably because disable_auto_reply
249.   (used in binarydevice.py to selectively silence/allow responses) is
250.   still on. Still in Console, click on the device, then Settings, and
251.   reset "Device Mode" such that bit 0 = 0 (i.e., round down to the
252.   nearest even number).
253. - ModuleNotFoundError probably means that it has not been installed.
254.   Instructions for installing pip:
255.       https://pip.pypa.io/en/stable/installing/.
256.   Once it's installed, call:
257.       [sudo] pip install [the name of the module].
258. - You must multiple V2 objects by integers, rather than integers by V2
259.   objects (cf. note in coordinates.py).
260. - If you turn the manual control knob on the rotary stage and it doesn't
261.   move, it's in Displacement Mode. To put it back in Velocity Mode, push
262.   in the control knob and hold it for a few seconds until the light
263.   blinks.
264. - If a serial connection cannot be made to the stages, make sure you've
265.   plugged in the USB to the port specified in execute_commands.py
266.   (define_operating_constants), or change the specified port to match
267.   where it's actually plugged in.
268.
269.
270. NOTES ON VERSION HISTORY
271. - Older versions of these Python libraries are available in this repo,
272.   but unfortunately they weren't saved as meaningful commits (i.e., I
273.   didn't use GitHub) but rather as ~random snapshots from when I got
274.   nervous about not having multiple copies. You may find it helpful to
275.   consult these versions to resolve issues, but note that I can't
276.   guarantee that the older versions were saved in fully functional
277.   states (e.g., maybe I was in the middle of adding new functionality
278.   when I saved them, I just can't remember, and now I can't test them).
279. - Control over Zaber stages alone is possible using old C# scripts (also
280.   available in this repo) that work with Zaber Console. Those files
281.   obviously have very different syntax, but more importantly they
282.   interact with Zaber stages quite differently; they may still be
283.   useful, though.
284.
285.
286. REFERENCE:
287. [1] https://www.zaber.com/protocol-manual?protocol=Binary
288. [2] https://www.zaber.com/support/docs/api/core-python/0.9/binary
289.     .html#binary-module
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git
PROGRAM:    execute_commands.py
AUTHOR:     Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Executes move commands for Zaber Technologies rotary (1 x X-RSW60A) and
linear (2 x T-LSM050A) stages, all using binary encoding, and supervises
voltage and current control of Keithley 2400 SourceMeter to open and
close a Thorlabs SH05 beam shutter.

Communication with Zaber stages is achieved using a heavily modified
version of Zaber's Serial Library[1], and relies on zaber/serial/*. Cf.
zaber/serial/binaryserial.py for more details. The short of it is: I
would recommend not changing these files.

Communication and control over the Keithley is achieved using a module
published by T. Max Roberts[2], and relies on keithley_handler.py.

PLEASE CONSULT README.md FOR AN OVERVIEW OF THE DESIGN OF THIS LIBRARY,
AND ALSO FOR PRACTICAL, TROUBLESHOOTING, AND REFERENCE INFORMATION.


LINKS:
[1] https://www.zaber.com/support/docs/api/core-python/0.6/ (available
      in this repo, with modifications, in zaber/serial/)
[2] https://github.com/maxroberts/Keithley-2400-SourceMeter-Python-Int
      erface/blob/master/keithley_serial.py (link is broken, but the
      code is available in this repo in keithley_handler.py)
"""


# PARAMETERS FOR TESTING CONDITIONS
# Modify Keithley/stage connections for particular testing conditions
#   e.g., don't bother moving the rotary if you're testing movement in
#   the plane DUMMY_CONNECTIONS effectively overrides CONNECT_ROTARY and
#   CONNECT_KEITHLEY e.g., D_C == True, these devices won't connect even
#   if C_R or C_K == True.
# MAC_TESTING determines what modules are imported (OS limitations)

DUMMY_CONNECTIONS = True
CONNECT_ROTARY = True
CONNECT_KEITHLEY = True
MAC_TESTING = True


import sys, time, math, re
from mapping_handler import MappingHandler
from coordinates import V2

# these modules are not available on mac
if not MAC_TESTING:
    import winsound
    import keyboard

if not DUMMY_CONNECTIONS:
    import keithley_handler as kc
    from zaber.serial import BinarySerial, BinaryDevice, BinaryCommand, CommandTy
pe

```

```
61.
62.  # POSITION_GETTER_MODE
63.  # Allows you to move stages manually and press 'p' (on your keyboard) to
64.  #   print out stage positions to the console. It's useful for defining
65.  #   corners of sample regions: run execute_commands.py, navigate to the
66.  #   sample origin using linear stage manual control knobs, and press 'p'
67.  #   on the keyboard to print out their position. Press 'esc' to exit
68.  #   POSITION_GETTER_MODE.
69.  # MOVE_MAPPING
70.  # Allows you to visualize how commands will run before you put the
71.  #   sample under the laser. True: then commands are given in a file that
72.  #   corresponds to the sample of interest, which file keeps track of
73.  #   historical read-write information so that the same commands can be
74.  #   executed again; False: then commands must be must bespecified
75.  #   directly in this file.
76.
77.  POSITION_GETTER_MODE = False
78.  MOVE_MAPPING = True
79.
80.
81.  # SAMPLE_NAME
82.  # To access the relevant sample's .txt data file (at SAMPLES_PATH +
83.  #   SAMPLE_NAME + ".txt") to define origins (GLOBAL_O and TR) for manual
84.  #   command-calling. This file also stores historical write information
85.  #   and is where new commands ought to be entered if (a) you are
86.  #   interested in using the MOVE_MAPPING feature, or (b) you'd like to
87.  #   keep a record of the writes you've made to this sample.
88.
89.  SAMPLES_PATH = "/Users/harperwallace/Dropbox/GitHub/howw-stage-
     controls/samples/"
90.  SAMPLE_NAME = "_example"
91.  #SAMPLE_NAME = "HW 2020-01-23 A"
92.  #SAMPLE_NAME = "HW 2020-01-23 B"
93.  #SAMPLE_NAME = "HW_1_19_1"
94.  #SAMPLE_NAME = "HW_1_19_2"
95.
96.
97.  def main():
98.
99.      try:
100.         define_operating_constants()
101.
102.         # Doesn't do anything if !CONNECT_KEITHLEY or fake connections
103.         setup_keithley()
104.
105.         # Be sure to call setup_stages() before moving/setting objective height
106.         # Note that there's some give in the rotation of the pin through
107.     #   the microscope coarse control
108.         setup_stages()
109.
110.         if POSITION_GETTER_MODE:
111.
112.             # Press 'p' to print current position to console
113.             keyboard.add_hotkey('p', print_position)
114.             # Press 'esc' once you're finished with position-getting
115.             keyboard.wait('esc')
116.
117.
118.         elif MOVE_MAPPING:
119.
```

```
            mh = MappingHandler(SAMPLES_PATH, SAMPLE_NAME, CONNECT_KEITHLEY, DEFAULT_HOME_SPEED)
            mh.draw_map()

            if mh.continue_to_run:
                write_mapped_commands(mh.new_cmds_array)
                mh.update_sample_history()

        else:
            # MANUAL COMMAND-CALLING
            # EXTREMELY IMPORTANT NOTES:
            # 1. DON'T LET THE OBJECTIVE LENS HIT THE SAMPLE HOLDER SCREWS.
            # 2. DON'T MELT THE BEAM SHUTTER BY LEAVING THE LASER ON IT TOO LONG.
            # (details in README.md)

            # Extract GLOBAL_O and TR from the sample's corresponding .txt file.
            # Note that if GLOBAL_O appears in a line below TR for some
            #   reason, then GLOBAL_O won't be read (i.e., don't change the
            #   order of GLOBAL_O and TR from the template file).
            try:
                log_file = open(SAMPLES_PATH + SAMPLE_NAME + ".txt", "r")
                for line in log_file.readlines():
                    if len(line.rstrip()) > 0 and line[0] != "#":
                        stripped_line = line.split("#")[0].rstrip()
                        if stripped_line.find("=") != -1 and stripped_line.find("=") < stripped_line.find("("):
                            val = stripped_line.split("=",1)[0].rstrip()
                            args = list(map(float, re.sub("[^0-9.,]","", stripped_line.split("=",1)[1].replace("V2","").replace("V3","")).split(",")))
                            if val == "GLOBAL_O":
                                GLOBAL_O = V2((args[0], args[1]))
                            elif val == "TR":
                                TR = V2((args[0], args[1]))
                                REGION_SIZE = GLOBAL_O - TR
                                break
            except FileNotFoundError:
                print("SAMPLE FILE NOT FOUND. USING GENERIC ORIGINS.")

        # Move writing region without redefining global origins
            LOCAL_O = V2((0, 0))

            ## BEGIN MANUAL COMMANDS:









            ## END MANUAL COMMANDS

            """ REFERENCE:
            write_parallel_lines_gap(z, start, end, gap, speed, num_lines):
            write_parallel_lines_vertical_continuous(z, start, end, gap, speed):
            write_parallel_lines_horizontal_continuous(z, start, end, gap, speed):
            write_parallel_lines_vertical_region_tall(z, speeds, gap, inter_speed_gap_factor = 0.2):
```

```
            write_parallel_lines_horizontal_region_wide(z, speeds, gap, inter_spe
    ed_gap_factor = 0.2):
            write_parallel_lines_horizontal_const_height(z, x_width, speeds, gap,
     inter_speed_gap_factor = 0.2):
            write_parallel_lines_delta_s(z, start, end, gap, speed, delta_speed,
    num_lines_per_speed, num_speeds):
            write_line(start, end, speed):
            write_circle(center, radius, speed):
            write_part_circle(center, radius, start_deg, end_deg, speed):
            outline_region(z, speed = None):
            wipe_region(z, gap = 0.08, speed = None):
            move_to(point, ground_speed = None, laser_on = False):
            home_all():
            """

        if not MAC_TESTING:
            # Make a little beep noise so I know it's all finished
            winsound.Beep(1000, 500)

    # SHIFT-CTRL-C TO INTERRUPT, THEN CLEAN UP
    except KeyboardInterrupt:
        print("--terminated--")
    except:
        print("--unexpected error--")
        raise
    finally:
        clean_up()


# Sends new commands in the sample's data file (after mapping by
#   MappingHandler and user confirmation) to stages for writing.
# NOTE: Definitions and re-definitions of LOCAL_O are already taken into
#   account; the position arguments passed to these commands are already
#   adjusted appropriately, so there is no need to adjust that value
#   between writes in this method.
def write_mapped_commands(cmds):

    LOCAL_O = V2((0,0))

    for cmd_args in cmds:
        if cmd_args[0] == "write_parallel_lines_vertical_continuous":
            write_parallel_lines_vertical_continuous(*cmd_args[1:])
        elif cmd_args[0] == "write_parallel_lines_horizontal_continuous":
            write_parallel_lines_horizontal_continuous(*cmd_args[1:])
        elif cmd_args[0] == "write_parallel_lines_vertical_region_tall":
            write_parallel_lines_vertical_region_tall(*cmd_args[1:])
        elif cmd_args[0] == "write_parallel_lines_horizontal_region_wide":
            write_parallel_lines_horizontal_region_wide(*cmd_args[1:])
        elif cmd_args[0] == "write_parallel_lines_horizontal_const_height":
            write_parallel_lines_horizontal_const_height(*cmd_args[1:])
        elif cmd_args[0] == "write_parallel_lines_gap":
            write_parallel_lines_gap(*cmd_args[1:])
        elif cmd_args[0] == "write_parallel_lines_delta_s":
            write_parallel_lines_delta_s(*cmd_args[1:])
        elif cmd_args[0] == "write_line":
            write_line(*cmd_args[1:])
        elif cmd_args[0] == "write_circle":
            write_circle(*cmd_args[1:])
        elif cmd_args[0] == "write_part_circle":
            write_part_circle(*cmd_args[1:])
        elif cmd_args[0] == "outline_region":
```

```
            outline_region(*cmd_args[1:])
        elif cmd_args[0] == "wipe_region":
            wipe_region(*cmd_args[1:])
        elif cmd_args[0] == "move_to":
            move_to(*cmd_args[1:])


# WRITE PARALLEL LINES: VERTICAL, CONTINUOUS
# Writes vertical parallel lines in an egyptian pattern: |-|_|-|_|
# NOTE: unlike other parallel_lines functions, start and end define
#   REGION BOUNDARIES rather than ENDPOINTS OF THE FIRST LINE. This
#   means that vertical, parallel lines will be written in order to fill
#   a rectangle with bottom-left corner = start + LOCAL_O, top-right
#   corner = end + LOCAL_O.
# NOTE: doesn't take num_lines as an argument.
# e.g., write_parallel_lines_vertical_continuous(
#                145.0, V2((0,0)), V2((2.3,1.3)), 0.2, 5)
def write_parallel_lines_vertical_continuous(z, start, end, gap, speed):

    if DUMMY_CONNECTIONS:
        return

    shift = V2((gap, 0))
    move_to(start)

    z_rotary.move_abs(mm2rotdata(z), await_reply = True)

    if CONNECT_KEITHLEY:
        kh.set_output_on()

    num_lines = int(abs((end - start).x)/float(gap))

    for i in range(0, num_lines, 2):

        if i > 0:
            move_to(start + shift*i, speed)
        move_to(V2((start.x, end.y)) + shift*i, speed)
        move_to(V2((start.x, end.y)) + shift*(i + 1), speed)
        move_to(start + shift*(i + 1), speed)

    if CONNECT_KEITHLEY:
        kh.set_output_off()

    # PRINT OUT WHERE TO MANUALLY SET LOCAL_O NEXT
    if not MOVE_MAPPING:
        print(LOCAL_O + V2((0, abs((end - start).y) + gap)))
        print("OR")
        print(LOCAL_O + V2(((num_lines + 1)*gap, 0)))


# WRITE PARALLEL LINES: HORIZONTAL, CONTINUOUS
# Writes horizontal parallel lines in a down->up egyptian pattern
# NOTE: doesn't take num_lines as an argument.
# NOTE: unlike other parallel_lines functions, start and end define
#   REGION BOUNDARIES rather than ENDPOINTS OF THE FIRST LINE.
# e.g., write_parallel_lines_horizontal_continuous(
#                145.0, V2((0,0)), V2((2.3,1.3)), 0.2, 5)
def write_parallel_lines_horizontal_continuous(z, start, end, gap, speed):

    if DUMMY_CONNECTIONS:
        return
```

```
291.
292.     shift = V2((0, gap))
293.     move_to(start)
294.
295.     z_rotary.move_abs(mm2rotdata(z), await_reply = True)
296.
297.     if CONNECT_KEITHLEY:
298.         kh.set_output_on()
299.
300.     num_lines = int(abs((end - start).y)/float(gap))
301.
302.     for i in range(0, num_lines, 2):
303.         if i > 0:
304.             move_to(start + shift*i, speed)
305.         move_to(V2((end.x, start.y)) + shift*i, speed)
306.         move_to(V2((end.x, start.y)) + shift*(i + 1), speed)
307.         move_to(start + shift*(i + 1), speed)
308.
309.     if CONNECT_KEITHLEY:
310.         kh.set_output_off()
311.
312.     # PRINT OUT WHERE TO SET LOCAL_O NEXT
313.     if not MOVE_MAPPING:
314.         print(LOCAL_O + V2((0, (num_lines + 1)*gap)))
315.         print("OR")
316.         print(LOCAL_O + V2((abs((end - start).x) + gap, 0)))
317.
318.
319. # WRITE PARALLEL LINES: VERTICAL, REGION-TALL
320. # Writes vertical lines that extend from LOCAL_O.y to LOCAL_O.y +
321. #   REGION_SIZE.y, with additional padding of 1 mm on each side. Two
322. #   such lines are written at each speed (mm/s) enumerated in speeds.
323. #   Inter-speed gap factor inflates the gap between same-speed line
324. #   pairs, so lines written at different speeds  can be more easily
325. #   distinguished and counted under the microscope. The laser stage
326. #   height adjusts according to z, *after* the linear stages move to the
327. #   starting position.
328. # e.g., write_parallel_lines_vertical_region_tall(
329. #               145.0, [8, 7, 6, 5], 0.6)
330. def write_parallel_lines_vertical_region_tall(z, speeds, gap, inter_speed_gap_fac
     tor = 0.2):
331.
332.     if DUMMY_CONNECTIONS:
333.         return
334.
335.     z_rotary.move_abs(mm2rotdata(z), await_reply = True)
336.
337.     for i in range(len(speeds)):
338.
339.         start = V2((i*(2 + inter_speed_gap_factor)*gap, -1))
340.         end = V2((i*(2 + inter_speed_gap_factor)*gap, REGION_SIZE.y + 1))
341.
342.         speed = speeds[i]
343.         print(speed)
344.
345.         if i == 0:
346.             move_to(start)
347.             z_rotary.move_abs(mm2rotdata(z), await_reply = True)
348.
349.         write_line(start, end, speed)
350.         write_line(end + V2((gap, 0)), start + V2((gap, 0)), speed)
```

```
351.
352.     if not MOVE_MAPPING:
353.         print(LOCAL_O + V2(((len(speeds)*(2 + inter_speed_gap_factor) + 1)*gap, 0
    )))
354.
355.
356. # HORIZONTAL, REGION-WIDE LINES
357. # Writes horizontal lines that extend from LOCAL_O.x to LOCAL_O.x +
358. #   REGION_SIZE.x, with additional padding of 1 mm on each side. Two
359. #   such lines are written at each speed (mm/s) enumerated in speeds.
360. #   Inter-speed gap factor inflates the gap between same-speed line
361. #   pairs, so lines written at different speeds  can be more easily
362. #   distinguished and counted under the microscope. The laser stage
363. #   height adjusts according to z, *after* the linear stages move to the
364. #   starting position.
365. # e.g., write_parallel_lines_vertical_region_tall(
366. #                145.0, [1, 0.8, 0.6, 0.4, 0.2], 0.6)
367. def write_parallel_lines_horizontal_region_wide(z, speeds, gap, inter_speed_gap_f
    actor = 0.2):
368.
369.     if DUMMY_CONNECTIONS:
370.         return
371.
372.     for i in range(len(speeds)):
373.
374.         start = V2((-1, i*(2 + inter_speed_gap_factor)*gap))
375.         end = V2((REGION_SIZE.x + 1, i*(2 + inter_speed_gap_factor)*gap))
376.
377.         speed = speeds[i]
378.         print(speed)
379.
380.         if i == 0:
381.             move_to(start)
382.             z_rotary.move_abs(mm2rotdata(z), await_reply = True)
383.
384.         write_line(start, end, speed)
385.         write_line(end + V2((0, gap)), start + V2((0, gap)), speed)
386.
387.     if not MOVE_MAPPING:
388.         print(LOCAL_O + V2((0, (len(speeds)*(2 + inter_speed_gap_factor) + 1)*gap
    )))
389.
390.
391. # WRITE HORIZONTAL LINES AT CONSTANT HEIGHT
392. # Writes horizontal lines that extend from LOCAL_O.x to LOCAL_O.x +
393. #   REGION_SIZE.x, with additional padding of 1 mm on each side. Two
394. #   such lines are written at each speed (mm/s) enumerated in speeds.
395. #   Inter-speed gap factor inflates the gap between same-speed line
396. #   pairs, so lines written at different speeds  can be more easily
397. #   distinguished and counted under the microscope. The laser stage
398. #   height adjusts according to z, *after* the linear stages move to the
399. #   starting position.
400. # e.g., write_parallel_lines_horizontal_const_height(
401. #                145.0, [8, 7, 6, 5], 0.6)
402. def write_parallel_lines_horizontal_const_height(z, x_width, speeds, gap, inter_s
    peed_gap_factor = 0.2):
403.
404.     if DUMMY_CONNECTIONS:
405.         return
406.
407.     for i in range(len(speeds)):
```

```
408.
409.         start = V2((0, i*(2 + inter_speed_gap_factor)*gap))
410.         end = V2((x_width, i*(2 + inter_speed_gap_factor)*gap))
411.
412.         speed = speeds[i]
413.         print(speed)
414.
415.         if i == 0:
416.             move_to(start)
417.             z_rotary.move_abs(mm2rotdata(z), await_reply = True)
418.
419.         write_line(start, end, speed)
420.         write_line(end + V2((0, gap)), start + V2((0, gap)), speed)
421.
422.     if not MOVE_MAPPING:
423.         print(LOCAL_O + V2((0, (len(speeds)*(2 + inter_speed_gap_factor) + 1)*gap
    )))
424.         print("OR")
425.         print(LOCAL_O + V2((x_width + gap, 0)))
426.
427.
428. # WRITE PARALLEL LINES WITH A CONSTANT GAP
429. # Write parallel lines with spaces: | | | |
430. # If gap is positive, then parallel lines are written along the axis
431. #   clockwise-perpendicular to the vector, (end - start). This means
432. #   that if the lines are vertical, and if start is below end, lines are
433. #   written left to right;  but if the lines are horizontal, and if
434. #   start is to the left of end, then lines are written top to bottom.
435. #   This all is mapped out by MappingHandler, but you should consider it
436. #   when designing parallel_lines_gap moves.
437. # NOTE: start and end define ENDPOINTS OF THE FIRST LINE. Additional
438. #   lines will be written parallal, and approximately to the right,
439. #   depending on the exact angle between start and end, at a
440. #   perpendicular distance of gap.
441. # e.g., write_parallel_lines_gap(
442. #               145.0, V2((0.0,0.3)), V2((1,0)), -0.2, 1, 5)
443. def write_parallel_lines_gap(z, start, end, gap, speed, num_lines):
444.
445.     if DUMMY_CONNECTIONS:
446.         return
447.
448.     direction = end - start
449.
450.     # in order to write //-  as opposed to -// (perpendicular_cntclk)
451.     shift = direction.unit.perpendicular_clk * gap
452.
453.     for i in range(num_lines):
454.
455.         if i == 0:
456.             move_to(start)
457.             z_rotary.move_abs(mm2rotdata(z), await_reply = True)
458.
459.         write_line(start + shift*i, end + shift*i, speed)
460.
461.
462. # WRITE PARALLEL LINES WITH A SPEED INCREMENT
463. # Writes parallel lines at different speeds, starting at speed and
464. #   increasing by delta_speed for every num_lines_per_speed lines
465. #   written at that temporary speed. num_speeds determines how many
466. #   times this outer loop iterates, and hence how many total lines are
467. #   written = num_lines_per_speed * num_speeds
```

```
# If gap is positive, then parallel lines are written along the axis
#   clockwise-perpendicular to the vector, (end - start). This means
#   that if the lines are vertical, and if start is below end, lines are
#   written left to right;  but if the lines are horizontal, and if
#   start is to the left of end, then lines are written top to bottom.
#   This all is mapped out by MappingHandler, but you should consider it
#   when designing write_parallel_lines_delta_s moves.
# NOTE: start and end define ENDPOINTS OF THE FIRST LINE. Additional
#   lines will be written parallel, and approximately to the right,
#   depending on the exact angle between start and end, at a
#   perpendicular distance of gap.
# e.g., write_parallel_lines_delta_s(
#                145.0, V2((0.0,0.3)), V2((1,0)), -0.2, 1, 1, 3, 3)
def write_parallel_lines_delta_s(z, start, end, gap_dist, speed, delta_speed, num_lines_per_speed, num_speeds):

    if DUMMY_CONNECTIONS:
        return

    direction = end - start

    # in order to write //-  as opposed to -// (perpendicular_cntclk)
    shift = direction.unit.perpendicular_clk * gap_dist

    for i in range(num_speeds):
        spacer = shift * (num_lines_per_speed + (0.7 if num_lines_per_speed > 1 else 0)) * i
        write_parallel_lines_gap(z, start + spacer, end + spacer, gap_dist, speed + i*delta_speed, num_lines_per_speed)


# WRITE LINE
# Writes a single line from start to end, at the given speed
def write_line(start, end, speed):

    if DUMMY_CONNECTIONS:
        return

    move_to(start)
    move_to(end, ground_speed = speed, laser_on = True)

# WRITE CIRCLE
# Just calls write_part_circle for start = 0, end = 360
def write_circle(center, radius, speed):

    if DUMMY_CONNECTIONS:
        return

    write_part_circle(center, radius, 0, 360, speed)


# WRITE PART CIRCLE
# Writes arcs by calling move_vel (i.e., move at constant speed) on the
#   x- and y- stages at ~differential time increments (defined in
#   milliseconds by DELTA_T). This is far and away the most complicated
#   move function, not least because of the idiosyncrasies of how the
#   Zaber stages handle commands, which makes it necessary to keep track
#   of time programmatically (rather than waiting for the device to
#   respond that it's finished). The key point is that circles can be
#   finicky.
"""
```

```
526. [start], [end] = deg
527. """
528. def write_part_circle(center, radius, start_deg, end_deg, speed):
529. 
530.     if DUMMY_CONNECTIONS:
531.         return
532. 
533.     invert_factor = 1 if INVERT_COORDINATES else -1
534. 
535.     # f = circle frequency; (mm/s) / (1000 * mm) = 1 / ms
536.     f = speed / (1000 * radius)
537.     # T = period = time it takes to do a whole circle (ms)
538.     T = 1000 * 2*math.pi * radius / speed
539. 
540.     # no need for invert_factor--handled in move_to
541.     start_pos = center + V2(start_deg)*radius
542.     move_to(start_pos)
543. 
544.     x_linear.disable_auto_reply()
545.     y_linear.disable_auto_reply()
546. 
547.     start_time = time.perf_counter()
548. 
549.     for t in range(int(start_deg * T / 360) + DELTA_T, int(end_deg * T / 360), DE
    LTA_T):
550. 
551.         delta = (V2(180/math.pi * f*t) - V2(180/math.pi * f*(t - DELTA_T))) * rad
    ius
552.         velocity = delta.unit * speed
553. 
554.         # await_reply set to None here because move_vel can't be
555.         # interrupted by disable_auto_reply (will get busy error
556.         # response = [_, 255, 255]);  None value just skips setting
557.         # auto_reply status
558. 
559.         x_linear.move_vel(invert_factor * linspeed2lindata(velocity.x), await_rep
    ly = None)
560.         y_linear.move_vel(invert_factor * linspeed2lindata(velocity.y), await_rep
    ly = None)
561. 
562.         while time.perf_counter() - start_time < 0.001*t:
563.             time.sleep(0.001)
564. 
565.     x_linear.stop()
566.     y_linear.stop()
567. 
568. 
569. # MOVE TO
570. # Moves to the specified point (V2) at a given ground_speed (mm/s),
571. #   which does not include the speed of changing z. As written, move_to
572. #   only moves in the plane, and z-stage controls must be passed
573. #   separately. However, it may become useful in the future to integrate
574. #   in-plane and z-axis controls so they are handled by the same
575. #   methods; cf. V3 (a three-dimensioned vector object) in older
576. #   versions of this file and of coordinates module (__v3/
577. #   execute_commands.py and /coordinates.py) for a head start on this.
578. # NOTE: A key feature of move_to is that it is able to handle
579. #   simultaneous moves by calculating the time that each stage is
580. #   expected to take, and awaiting a reply only from the slower-moving
581. #   stage. Though this design is not as useful for simultaneous moves
582. #   only in the x-y plane (where ground_speed is separated into x- and
```

```
583. #   y- components such that both linear stages should take the same
584. #   amount of time to complete a given move), it is very useful if
585. #   z-axis commands are handled together with x-y commands, since the
586. #   rotary stage moves slowly (cf. EXTREMELY IMPORTANT NOTE #3 in
587. #   README.md). As above, cf. older versions of this file in (__v3/)
588. #   for a head start at achieving this.
589. # NOTE: This makes use of the method time_to_move (below) to predict
590. #   move times. Cf. the notes on that method.
591. """
592. [point] = V2, in mm
593. [ground_speed] = mm/s; if value == None, => DEFAULT_HOMING_SPEED
594. """
595. def move_to(point, ground_speed = None, laser_on = False):
596.
597.     if DUMMY_CONNECTIONS:
598.         return
599.
600.     if CONNECT_KEITHLEY:
601.         if not laser_on:
602.             kh.set_output_off()
603.         else:
604.             kh.set_output_on()
605.
606.     ground_speed = DEFAULT_HOME_SPEED if ground_speed is None else ground_speed
607.
608.
609.     # point data as a position (i.e., given invert, FsOR)
610.     global_point = local2globalmm(point)
611.     global_point_data = mm2lindata(global_point)
612.
613.     dist = global_point - current_position()
614.     dist_data = mm2lindata(dist)
615.
616.     veloc = abs(dist.unit) * ground_speed
617.
618.     x_time = time_to_move(dist.x, veloc.x, LIN_STAGE_ACCELERATION)
619.     y_time = time_to_move(dist.y, veloc.y, LIN_STAGE_ACCELERATION)
620.
621.     times = [x_time, y_time]
622.     last_to_move = times.index(max(times))
623.
624.     if abs(dist_data.x) > 0:
625.         x_linear.set_target_speed(linspeed2lindata(veloc.x), await_reply = True)

626.     if abs(dist_data.y) > 0:
627.         y_linear.set_target_speed(linspeed2lindata(veloc.y), await_reply = True)

628.
629.     # this is super ugly but it has to go like this procedurally, such that the
630.     #   command "await_reply"-
    ing (i.e., the slowest move) is the last one called
631.     if last_to_move == 0:
632.         if abs(dist_data.y) > 0:
633.             # don't await reply if x-axis will be slowest (necessary for
634.         #   simultaneous moves)
635.             y_linear.move_abs(global_point_data.y)
636.         if abs(dist_data.x) > 0:
637.             x_linear.move_abs(global_point_data.x, await_reply = True)
638.     elif last_to_move == 1:
639.         if abs(dist_data.x) > 0:
640.             # don't await reply if y-axis will be slowest
```

```
641.             x_linear.move_abs(global_point_data.x)
642.         if abs(dist_data.y) > 0:
643.             y_linear.move_abs(global_point_data.y, await_reply = True)
644.
645.
646.     if CONNECT_KEITHLEY and laser_on:
647.         kh.set_output_off()
648.
649.
650. # OUTLINE THE GLOBAL REGION
651. # Draws an outline around the REGION_SIZE (ignoring LOCAL_O) in order to
652. #   test GLOBAL_O and TR boundaries, e.g., as determined in
653. #   POSITION_GETTER_MODE.
654. """
655. z = stage height;  [z] = mm
656. [speed] = mm/s; if value == None, => DEFAULT_HOMING_SPEED
657. """
658. def outline_region(z, speed = None):
659.
660.     if DUMMY_CONNECTIONS:
661.         return
662.
663.     move_to(V2((-0.1, -0.1)) + LOCAL_O * (-1 if INVERT_COORDINATES else 1))
664.     z_rotary.move_abs(mm2rotdata(z), await_reply = True)
665.
666.     if CONNECT_KEITHLEY:
667.         kh.set_output_on()
668.
669.     move_to(V2((-0.1, REGION_SIZE.y + 0.1)) + LOCAL_O * (-
    1 if INVERT_COORDINATES else 1), speed)
670.     move_to(REGION_SIZE + V2((0.1, 0.1)) + LOCAL_O * (-
    1 if INVERT_COORDINATES else 1), speed)
671.     move_to(V2((REGION_SIZE.x + 0.1, -0.1)) + LOCAL_O * (-
    1 if INVERT_COORDINATES else 1), speed)
672.     move_to(V2((-0.1, -0.1)) + LOCAL_O * (-
    1 if INVERT_COORDINATES else 1), speed)
673.
674.     if CONNECT_KEITHLEY:
675.         kh.set_output_off()
676.
677.
678. # WIPE THE REGION
679. # Writes parallel lines that are separated by gap to be sufficiently
680. #   small such that write line edges overlap slightly (depending on
681. #   power and focal distance), in order to achieve bulk isotropy.
682. def wipe_region(z, gap = 0.08, speed = None):
683.
684.     if DUMMY_CONNECTIONS:
685.         return
686.
687.     move_to(V2((-1, 0)) + LOCAL_O * (-1 if INVERT_COORDINATES else 1))
688.     z_rotary.move_abs(mm2rotdata(z), await_reply = True)
689.
690.     if CONNECT_KEITHLEY:
691.         kh.set_output_on()
692.
693.     for i in range(int(REGION_SIZE.y / (2*gap)) + 1):
694.         if i > 0:
695.             move_to(V2((-1, gap * 2*i))                    + LOCAL_O * (-
    1 if INVERT_COORDINATES else 1), speed)
```

```
696.         move_to(V2((REGION_SIZE.x + 1, gap * 2*i))         + LOCAL_O * (-
     1 if INVERT_COORDINATES else 1), speed)
697.         move_to(V2((REGION_SIZE.x + 1, gap * (2*i + 1)))   + LOCAL_O * (-
     1 if INVERT_COORDINATES else 1), speed)
698.         move_to(V2((-1, gap * (2*i + 1)))                  + LOCAL_O * (-
     1 if INVERT_COORDINATES else 1), speed)
699.
700.     if CONNECT_KEITHLEY:
701.         kh.set_output_off()
702.
703.
704. # HOME ALL
705. # Sends all stages back to their 'home' positions (x = y = z = 0)
706. def home_all():
707.
708.     if DUMMY_CONNECTIONS:
709.         return
710.
711.     # await_reply must = True for all homing commands, otherwise
712.     #   there'll be 'busy' errors because the succeeding commands will
713.     #   be called before homing is complete
714.
715.     if CONNECT_ROTARY:
716.         z_rotary.home(await_reply = True)
717.
718.     x_linear.home(await_reply = True)
719.     y_linear.home(await_reply = True)
720.
721.
722. """ CONVERSIONS """
723. # NOTE: ALWAYS int-cast when converting real units to data.
724. # NOTE: NEVER int-cast when converting data to a real unit.
725.
726. # Converts distance (or V2) in mm to linear stage data
727. # Returns an int, or V2 of ints
728. def mm2lindata(mm):
729.     if type(mm) is V2:
730.         return (mm * DATA_PER_MM).round()
731.     return (int)(mm * DATA_PER_MM)
732.
733. # Converts local position (V2, in mm) to global position (V2, in mm, OUT
734. #   OF CONTEXT of defined coordinate system), considering inversion, and
735. #   global and local frames of reference
736. # Returns V2 (mm)
737. def local2globalmm(local_mm):
738.     if (INVERT_COORDINATES):
739.         return TR + LOCAL_O + local_mm
740.     return GLOBAL_O - LOCAL_O - local_mm
741.
742. # Converts linear speed (mm/s) to linear stage speed data (mstep/s)
743. # Returns an int
744. def linspeed2lindata(speed):
745.     return (int)(speed * DATA_PER_MM_SPEED)
746.
747. # Converts linear stage distance data (mstep, or V2 of msteps) to
748. #   distance in mm, or V2 of mm
749. def lindata2mm(data):
750.     return data / DATA_PER_MM
751.
752. # Converts an arbitrary measurement of focal distance ~ laser height
753. #   (mm) to degree-related rotary stage data (cf. B_EMPIR in
```

```
754. #   define_operating_constants, below)
755. # Returns an int
756. def mm2rotdata(mm):
757.     return deg2rotdata(B_EMPIR - DEG_PER_MM * mm)
758. 
759. # Converts rotary stage data to an arbitrary measurement of focal
760. #   distance ~ laser height (mm)
761. # Returns a float
762. def rotdata2mm(data):
763.     return (-rotdata2deg(data) + B_EMPIR) / DEG_PER_MM
764. 
765. # Converts rotation angle (degrees) to rotary stage data
766. # Returns an int
767. def deg2rotdata(deg):
768.     return (int)(deg * DATA_PER_DEG)
769. 
770. # Converts rotary stage data to rotation angle angle (degrees)
771. # Returns an float
772. def rotdata2deg(data):
773.     return data / DATA_PER_DEG
774. 
775. # Converts rotary stage speed (mm/s) to data
776. # Returns an int
777. def linspeed2rotdata(speed):
778.     return degspeed2rotdata(speed * DEG_PER_MM)
779. 
780. # Converts rotary stage speed (deg/s) to data
781. # Returns an int
782. def degspeed2rotdata(speed):
783.     return (int)(speed * DATA_PER_DEG_SPEED)
784. 
785. 
786. # TIME TO MOVE
787. #   Calculates the expected time it should take to make a given move,
788. #   based on distance, speed, and acceleration. This works well with x-
789. #   and y- stages, and although it could be made to work with the rotary
790. #   stage, you'd need to work in some additional conversion factors. Cf.
791. #   notes on units in README.md.
792. """
793. [dist] = mm
794. [speed] = mm/s
795. [accel] = mm/s^2
796. """
797. def time_to_move(dist, speed, accel):
798.     if mm2lindata(dist) == 0 or linspeed2lindata(speed) == 0:
799.         return 0
800. 
801.     dist = abs(dist)
802. 
803.     dist_always_accel = speed**2 / (2 * accel)
804.     time_always_accel = (2 * dist_always_accel / accel)**0.5
805. 
806.     if dist < dist_always_accel:
807.         return time_always_accel
808.     return time_always_accel + (dist - dist_always_accel) / speed
809. 
810. 
811. # CURRENT POSITION
812. # Polls x- and y- stages for their positions and returns a V2 object OUT
813. #   OF the context of the defined coordinate system (i.e., not
814. #   considering GLOBAL_O, autc.). It's used by print_position (for
```

```
#   POSITION_GETTER_MODE) and also by time_to_move (to calculate the
#   distance to a target position).
def current_position():
    if DUMMY_CONNECTIONS:
        return V2((0,0))
    return V2((lindata2mm(x_linear.get_position()), lindata2mm(y_linear.get_posit
ion())))

def print_position():
    # For some reason you have to poll position several times before
    #   it's properly updated, but this may be a TIME delay rather than
    #   a frequency delay...  more testing will tell.
    for i in range(15):
        current_position()
    print(current_position())


""" CONNECTIONS AND SETUP """

# SETUP KEITHLEY
# (Unless DUMMY_CONNECTIONS or !CONNECT_KEITHLEY):
# Opens a serial connection to the Keithley 2400 SourceMeter and sets
#   operating values that enable control of the connected Thorlabs SH05
#   beam shutter
def setup_keithley():
    global kh

    if DUMMY_CONNECTIONS or not CONNECT_KEITHLEY:
        return

    kh = kc.KeithleyHandler()

    kh.set_source_current(0.4)
    kh.set_voltage_compliance(21.0)
    kh.set_output_on()

# SETUP STAGES
# (Unless DUMMY_CONNECTIONS):
# Opens serial connections to Zaber stages and sets default operating
#   parameters, like target speed, acceleration, and max position, all
#   defined in define_operating_constants (below)
def setup_stages():
    global serial_conn, x_linear, y_linear, z_rotary

    if DUMMY_CONNECTIONS:
        return

    serial_conn = BinarySerial(STAGES_PORT, timeout = None)

    if CONNECT_ROTARY:
        z_rotary = BinaryDevice(serial_conn, 1)
        z_rotary.set_home_speed(degspeed2rotdata(DEFAULT_ROT_SPEED))
        z_rotary.set_target_speed(degspeed2rotdata(DEFAULT_ROT_SPEED))
        z_rotary.set_acceleration(ROT_STAGE_ACCELERATION)

        z_rotary.set_min_position(deg2rotdata(ROTARY_MIN_ANGLE))
        z_rotary.set_max_position(deg2rotdata(ROTARY_MAX_ANGLE))


    x_linear = BinaryDevice(serial_conn, 2)
    y_linear = BinaryDevice(serial_conn, 3)
```

```
875.
876.      x_linear.set_home_speed(linspeed2lindata(DEFAULT_HOME_SPEED))
877.      y_linear.set_home_speed(linspeed2lindata(DEFAULT_HOME_SPEED))
878.
879.      x_linear.set_target_speed(linspeed2lindata(DEFAULT_HOME_SPEED))
880.      y_linear.set_target_speed(linspeed2lindata(DEFAULT_HOME_SPEED))
881.
882.      x_linear.set_acceleration(LIN_STAGE_ACCELERATION)
883.      y_linear.set_acceleration(LIN_STAGE_ACCELERATION)
884.
885.      x_linear.disable_manual_move_tracking()
886.      y_linear.disable_manual_move_tracking()
887.
888.
889.      home_all()
890.
891.
892.  # CLEAN UP
893.  # (Unless !CONNECT_KEITHLEY):
894.  # Turns off output of Keithley 2400 SourceMeter.
895.  # (Unless DUMMY_CONNECTIONS):
896.  # Homes stages and resets target speeds to default (in order to speed up
897.  #   manual moves after programmed slower ones), then closes the serial
898.  #   connection to the Zaber stages.
899.  def clean_up():
900.      print("cleaning up")
901.
902.      if DUMMY_CONNECTIONS:
903.          print("nothing to clean up: DUMMY_CONNECTIONS = True")
904.          return
905.
906.      if CONNECT_KEITHLEY:
907.          kh.set_output_off()
908.
909.      home_all()
910.
911.      x_linear.set_target_speed(linspeed2lindata(DEFAULT_HOME_SPEED))
912.      y_linear.set_target_speed(linspeed2lindata(DEFAULT_HOME_SPEED))
913.
914.      x_linear.enable_auto_reply()
915.      y_linear.enable_auto_reply()
916.
917.      serial_conn.close()
918.
919.      print("finished")
920.
921.
922.  # DEFINE OPERATING CONSTANTS
923.  # Sets global values, e.g., default speeds and accelerations, conversion
924.  #   factors, and inversion parameter.
925.  def define_operating_constants():
926.
927.      """          ZABER CONTROL          """
928.      global GLOBAL_O, TR, LOCAL_O, REGION_SIZE, \
929.             STAGES_PORT, MM_PER_MSTEP, DATA_PER_MM, DATA_PER_MM_SPEED, DATA_PER_DE
      G, DATA_PER_DEG_SPEED, DEG_PER_MM, \
930.             DELTA_T, DEFAULT_HOME_SPEED, DEFAULT_ROT_SPEED, LIN_STAGE_ACCELERATION
      , ROT_STAGE_ACCELERATION, \
931.             ROTARY_MIN_ANGLE, ROTARY_MAX_ANGLE, B_EMPIR, INVERT_COORDINATES
932.
933.
```

```
    GLOBAL_O = V2((35.6, 39))       # ORIGIN = LASER FOCUSED ON BOTTOM LEFT CORNER
    TR = V2((21.6, 25))         # ...TOP RIGHT CORNER
    LOCAL_O = V2((0, 0))
    REGION_SIZE = GLOBAL_O - TR

    STAGES_PORT = "COM3"                # serial port to Zaber stages
    DATA_PER_MM = 1000 / 0.047625       # conversion from mm to data
    DATA_PER_MM_SPEED = 2240            # conversion from mm/s to data (speed)
    DATA_PER_DEG = 12800 / 3            # conversion from degrees to data
    DATA_PER_DEG_SPEED = 6990           # conversion from deg/s to data (speed)
    DEG_PER_MM = 9.2597                 # (deg/mm) empirical conversion, mm to degrees
    DELTA_T = 24                        # (ms) SET LOWER WHEN WRITING FASTER

    DEFAULT_HOME_SPEED = 5              # (mm/s)
    DEFAULT_ROT_SPEED = 15              # (deg/s)
    LIN_STAGE_ACCELERATION = 2000       # (data/s^2) ?

    # BE CAREFUL ABOUT INCREASING THIS VALUE! Cf. EXTREMELY IMPORTANT
    #   NOTES #3 in README.rm.
    ROT_STAGE_ACCELERATION = 40         # (data/s^2) ?

    ROTARY_MIN_ANGLE = 0.0          # (deg) min position of rotary stage = upper bound on laser height
    ROTARY_MAX_ANGLE = 113.5            # (deg) max postition of rotary stage = min. allowable laser height
    B_EMPIR = 1414.9692                 # (deg) position of rotary stage at 0 mm

    # True if writing to a sample to be viewed in microscope (since microscope inverts image)
    INVERT_COORDINATES = True;


if __name__ == '__main__':
    main()
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git
PROGRAM:    mapping_handler.py
AUTHOR:     Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Mapping predicts how moves will track before actual laser-writing, so
paths can be verified before a sample is altered, so the time it will
take to write these paths can be estimated before writing, and also so
that the user can keep track of the paths they've written to a sample in
the past (path renderings can be saved as .jpgs using the "Save" button
at the bottom of the gui).

I have not had the opportunity to test this portion of the library
together with execute_commands.py on the actual setup (since labwork and
University operations were suspended in response to COVID-19), which
means that paths predicted here may vary from those that are actually
followed; in particular, I am uncertain about the angular direction the
stages move when tracing a circle. Thankfully, verifying these predicted
paths should only entail writing a series of commands to an actual
sample and comparing the written pattern to the predicted one; if there
are discrepancies, I'd recommend updating the code (below) that renders
the prediction, rather than the code that actually writes to the samples.
"""


import numpy as np
import math
import re
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import matplotlib.lines as mlines
from matplotlib.widgets import Button

from matplotlib.patches import Rectangle, Circle, Arc
from coordinates import V2

from datetime import datetime
from pytz import timezone



class MappingHandler(object):

    def __init__(self, path_prefix, sample_name, connect_keithley, default_speed)
:

        self.path_prefix = path_prefix
        self.sample_name = sample_name

        # If True, assumes that the beam shutter will close for moves
        #   between (and hence won't render those moves, but will
        #   consider and the paths between moves anyway, as gray,
        #   dashed lines)

        self.connect_keithley = connect_keithley

        self.default_speed = default_speed

        self.TR = V2((0, 0))
```

```
61.           self.GLOBAL_O = V2((0, 0))
62.           self.REGION_SIZE = V2((0, 0))
63.           self.local_o = V2((0, 0))
64.           self.curr_pos = V2((0, 0))
65.           self.new_file_text = ""
66.           self.new_cmds_array = []
67.           self.continue_to_run = False
68.
69.       def draw_map(self):
70.           fix, ax = plt.subplots(nrows=1, ncols=1, figsize=(5.5,6))
71.           plt.subplots_adjust(bottom=0.2)
72.
73.           axbcancel = plt.axes([0.7, 0.05, 0.1, 0.075])
74.           bcancel = Button(axbcancel, 'Cancel')
75.           bcancel.on_clicked(self.cancel)
76.
77.           axbrun = plt.axes([0.81, 0.05, 0.1, 0.075])
78.           brun = Button(axbrun, 'Run')
79.           brun.on_clicked(self.run)
80.
81.           try:
82.               total_time = 0
83.               curr_command = 0
84.               in_new_cmds = False
85.
86.               log_file = open(self.path_prefix + self.sample_name + ".txt", "r")
87.
88.               for line in log_file.readlines():
89.
90.                   if len(line.rstrip()) > 0 and line[0] != "#":
91.
92.                       step_time = 0
93.                       stripped_line = line.split("#")[0].rstrip()
94.
95.                       # setting GLOBAL_O, TR, or LOCAL_O
96.                       if stripped_line.find("=") != -
    1 and stripped_line.find("=") < stripped_line.find("("):
97.
98.                           val = stripped_line.split("=",1)[0].rstrip()

99.                           args = list(map(float, re.sub("[^0-
    9.,]","", stripped_line.split("=",1)[1].replace("V2","").replace("V3","")).split(
    ",")))
100.
101.                          if val == "GLOBAL_O":
102.                              self.GLOBAL_O = V2((args[0], args[1]))
103.                          elif val == "TR":
104.                              self.TR = V2((args[0], args[1]))
105.                              self.REGION_SIZE = self.GLOBAL_O - self.TR
106.                              ax.add_patch(Rectangle((0, 0), self.REGION_SIZE.x, se
    lf.REGION_SIZE.y, fill=None, alpha=1))
107.                              plt.xlim(left = - 0.05 * self.REGION_SIZE.x, right =
    1.05 * self.REGION_SIZE.x)
108.                              plt.ylim(bottom = - 0.05 * self.REGION_SIZE.y, top =
    1.05 * self.REGION_SIZE.y)
109.                              ax.xaxis.set_ticks(np.arange(0,self.REGION_SIZE.x,2))

110.                              ax.yaxis.set_ticks(np.arange(0,self.REGION_SIZE.y,2))

111.                              ax.set_aspect('equal', adjustable='box')
112.                          elif val == "LOCAL_O":
```

```
                            self.local_o = V2((args[0], args[1]))

                        self.new_file_text += line

                    # calling command
                    else:

                        curr_command += 1

                        cmd = stripped_line.split("(",1)[0]

                        # separates out arguments, omitting lists (e.g., "speeds" array in write_parallel_lines_vertical_region_tall)

                        args_str = stripped_line.split("(",1)[1].replace("V2","").replace("V3","")
                        array_arg = re.search("\[.*?\]", args_str)

                        if array_arg:
                            speeds = list(map(float, re.sub("\[|\]", "", array_arg.group(0)).split(",")))
                            args_str = re.sub("\[.*?\]", "{}".format(len(speeds)), args_str)
                        args = list(map(float, re.sub("[^0-9.,-]", "", args_str).split(",")))


                        # write_parallel_lines_vertical_continuous(z, start, end, gap, speed)
                        if cmd == "write_parallel_lines_vertical_continuous":
                            start = self.local_o + V2((args[1], args[2]))
                            end = self.local_o + V2((args[3], args[4]))
                            gap = args[5]
                            shift = V2((gap, 0))
                            speed = args[6]
                            ax.annotate(curr_command, xy=(start.x, start.y))

                            num_lines = int(abs((end - start).x)/float(shift.x))

                            for i in range(0, num_lines, 2):
                                step_time += self.render_write_line(ax, start + shift*i, V2((start.x, end.y)) + shift*i, speed, in_new_cmds)
                                step_time += self.render_write_line(ax, V2((start.x, end.y)) + shift*(i + 1), start + shift*(i + 1), speed, in_new_cmds)

                            if in_new_cmds:
                                self.new_cmds_array.append((cmd, args[0], start, end, gap, speed))


                        # write_parallel_lines_horizontal_continuous(z, start, end, gap, speed)
                        elif cmd == "write_parallel_lines_horizontal_continuous":

                            start = self.local_o + V2((args[1], args[2]))
                            end = self.local_o + V2((args[3], args[4]))
                            gap = args[5]
                            speed = args[6]
                            shift = V2((0, gap))
                            ax.annotate(curr_command, xy=(start.x, start.y))
```

```
162.
163.                             num_lines = int(abs((end - start).y)/float(shift.y))
164.                             for i in range(0, num_lines, 2):
165.                                 step_time += self.render_write_line(ax, start + s
    hift*i, V2((end.x, start.y)) + shift*i, speed, in_new_cmds)
166.                                 step_time += self.render_write_line(ax, V2((end.x
    , start.y)) + shift*(i + 1), start + shift*(i + 1), speed, in_new_cmds)
167.
168.                             if in_new_cmds:
169.                                 self.new_cmds_array.append((cmd, args[0], start,
    end, gap, speed))
170.
171.
172.                         # write_parallel_lines_vertical_region_tall(z, speeds, ga
    p, inter_speed_gap_factor = 0.2)
173.                         elif cmd == "write_parallel_lines_vertical_region_tall":
174.
175.                             start = self.local_o + V2((0, -1))
176.                             end = self.local_o + V2((0, self.REGION_SIZE.y + 1))
177.                             gap = args[2]
178.                             inter_speed_gap_factor = 0.2 if len(args) < 4 else ar
    gs[3]
179.                             shift = V2(((2 + inter_speed_gap_factor)*gap, 0))
180.
181.                             ax.annotate(curr_command, xy=(start.x, start.y))
182.                             for i in range(len(speeds)):
183.                                 speed = speeds[i]
184.
185.                                 step_time += self.render_write_line(ax, start + s
    hift*i, end + shift*i, speed, in_new_cmds)
186.                                 step_time += self.render_write_line(ax, end + shi
    ft*i + V2((gap, 0)), start + shift*i + V2((gap, 0)), speed, in_new_cmds)
187.
188.                             if in_new_cmds:
189.                                 self.new_cmds_array.append((cmd, args[0], speeds,
     gap, inter_speed_gap_factor))
190.
191.
192.                         # write_parallel_lines_horizontal_region_wide(z, speeds,
    gap, inter_speed_gap_factor = 0.2)
193.                         elif cmd == "write_parallel_lines_horizontal_region_wide"
    :
194.
195.                             start = self.local_o + V2((-1, 0))
196.                             end = self.local_o + V2((self.REGION_SIZE.x + 1, 0))
197.                             gap = args[2]
198.                             inter_speed_gap_factor = 0.2 if len(args) < 4 else ar
    gs[3]
199.                             shift = V2((0, (2 + inter_speed_gap_factor)*gap))
200.
201.                             ax.annotate(curr_command, xy=(start.x, start.y))
202.                             for i in range(len(speeds)):
203.                                 speed = speeds[i]
204.
205.                                 step_time += self.render_write_line(ax, start + s
    hift*i, end + shift*i, speed, in_new_cmds)
```

```
206.                                     step_time += self.render_write_line(ax, end + shi
    ft*i + V2((0, gap)), start + shift*i + V2((0, gap)), speed, in_new_cmds)
207.
208.                                 if in_new_cmds:
209.                                     self.new_cmds_array.append((cmd, args[0], speeds,
     gap, inter_speed_gap_factor))
210.
211.
212.                             # write_parallel_lines_horizontal_const_height(z, x_width
    , speeds, gap, inter_speed_gap_factor = 0.2)
213.                             elif cmd == "write_parallel_lines_horizontal_const_height
    ":
214.
215.                                 x_width = args[1]
216.                                 start = self.local_o
217.                                 end = self.local_o + V2((x_width, 0))
218.                                 gap = args[3]
219.                                 inter_speed_gap_factor = 0.2 if len(args) < 5 else ar
    gs[4]
220.                                 shift = V2((0, (2 + inter_speed_gap_factor)*gap))
221.
222.                                 ax.annotate(curr_command, xy=(start.x, start.y))
223.                                 for i in range(len(speeds)):
224.                                     speed = speeds[i]
225.
226.                                     step_time += self.render_write_line(ax, start + s
    hift*i, end + shift*i, speed, in_new_cmds)
227.                                     step_time += self.render_write_line(ax, end + shi
    ft*i + V2((0, gap)), start + shift*i + V2((0, gap)), speed, in_new_cmds)
228.
229.                                 if in_new_cmds:
230.                                     self.new_cmds_array.append((cmd, args[0], x_width
    , speeds, gap, inter_speed_gap_factor))
231.
232.
233.                             # write_parallel_lines_gap(z, start, end, gap, speed, num
    _lines)
234.                             elif cmd == "write_parallel_lines_gap":
235.
236.                                 start = self.local_o + V2((args[1], args[2]))
237.                                 end = self.local_o + V2((args[3], args[4]))
238.                                 gap = args[5]
239.                                 speed = args[6]
240.                                 num_lines = int(args[7])
241.                                 ax.annotate(curr_command, xy=(start.x, start.y))
242.
243.                                 direction = end - start
244.                                 shift = direction.unit.perpendicular_clk * gap
245.                                 for i in range(num_lines):
246.                                     step_time += self.render_write_line(ax, start + s
    hift*i, end + shift*i, speed, in_new_cmds)
247.
248.                                 if in_new_cmds:
249.                                     self.new_cmds_array.append((cmd, args[0], start,
    end, gap, speed, num_lines))
250.
251.                             # write_parallel_lines_delta_s(z, start, end, gap_dist, s
    peed, delta_speed, num_lines_per_speed, num_speeds)
252.                             elif cmd == "write_parallel_lines_delta_s":
253.
254.                                 start = self.local_o + V2((args[1], args[2]))
```

```
                                end = self.local_o + V2((args[3], args[4]))
                                gap_dist = args[5]
                                speed = args[6]
                                delta_speed = args[7]
                                num_lines_per_speed = int(args[8])
                                num_speeds = int(args[9])
                                ax.annotate(curr_command, xy=(start.x, start.y))

                                direction = end - start
                                shift = direction.unit.perpendicular_clk * gap_dist
                                inter_speed_spacer = shift * (num_lines_per_speed + (0.7 if num_lines_per_speed > 1 else 0))
                                for i in range(num_speeds):

                                    for j in range(num_lines_per_speed):
                                        step_time += self.render_write_line(ax, start + inter_speed_spacer*i + shift*j, end + inter_speed_spacer*i + shift*j, speed + i*delta_speed, in_new_cmds)

                                if in_new_cmds:
                                    self.new_cmds_array.append((cmd, args[0], start, end, gap_dist, speed, delta_speed, num_lines_per_speed, num_speeds))


                            # write_line(start, end, speed)
                            elif cmd == "write_line":

                                start = self.local_o + V2((args[0], args[1]))
                                end = self.local_o + V2((args[2], args[3]))
                                speed = args[4]

                                ax.annotate(curr_command, xy=(start.x, start.y))
                                step_time += self.render_write_line(ax, start, end, speed, in_new_cmds)

                                if in_new_cmds:
                                    self.new_cmds_array.append((cmd, start, end, speed))


                            # write_circle(center, radius, speed)
                            elif cmd == "write_circle":

                                center = self.local_o + V2((args[0], args[1]))
                                radius = args[2]
                                speed = args[3]
                                color = '#DF8800' if in_new_cmds else '#149E27'

                                start = V2((center.x + radius, center.y))
                                ax.annotate(curr_command, xy=(start.x, start.y)) #args[-1][2:], xy=(start.x, start.y))
                                ax.add_patch(Circle((center.x, center.y), radius = radius, fill=None, alpha=1, color=color, linewidth=1.5))

                                if in_new_cmds:
                                    step_time += self.render_move_to_start(ax, start) + 2*np.pi*radius / speed
                                    self.new_cmds_array.append((cmd, center, radius, speed))

                                self.curr_pos.setXY(start)
```

```
306.
307.
308.                             # write_part_circle(center, radius, start_deg, end_deg, s
    peed)
309.                             elif cmd == "write_part_circle":
310.
311.                                 center = self.local_o + V2((args[0], args[1]))
312.                                 radius = args[2]
313.                                 start_deg = args[3]
314.                                 end_deg = args[4]
315.                                 speed = args[5]
316.                                 color = '#DF8800' if in_new_cmds else '#149E27'
317.
318.                                 start = center + V2(start_deg)*radius
319.                                 ax.annotate(curr_command, xy=(start.x, start.y))
    #args[-1][2:], xy=(start.x, start.y))
320.                                 ax.add_patch(Arc((center.x, center.y), 2*radius, 2*ra
    dius, 0, start_deg, end_deg, fill=None, alpha=1, color=color, linewidth=1.5))
321.
322.                                 if in_new_cmds:
323.                                     step_time += self.render_move_to_start(ax, start)
     + 2*np.pi*(start_deg - end_deg)*radius / (360*speed)
324.                                     self.new_cmds_array.append((cmd, center, radius,
    start_deg, end_deg, speed))
325.
326.                                 self.curr_pos.setXY(center + V2(end_deg)*radius)
327.
328.
329.                             # outline_region(z, speed = None)
330.                             # DON'T CORRECT FOR LOCAL_O
331.                             elif cmd == "outline_region":
332.
333.                                 speed = self.default_speed if len(args) < 2 else args
    [1]
334.
335.                                 ax.annotate(curr_command, xy=(-0.1, -0.1))
336.                                 step_time += self.render_write_line(ax, V2((-0.1,-
    0.1)), V2((-0.1, self.REGION_SIZE.y + 0.1)), speed, in_new_cmds)
337.                                 step_time += self.render_write_line(ax, V2((-
    0.1, self.REGION_SIZE.y + 0.1)), self.REGION_SIZE + V2((0.1, 0.1)), speed, in_new
    _cmds)
338.                                 step_time += self.render_write_line(ax, self.REGION_S
    IZE + V2((0.1, 0.1)), V2((self.REGION_SIZE.x + 0.1, -0.1)), speed, in_new_cmds)
339.                                 step_time += self.render_write_line(ax, V2((self.REGI
    ON_SIZE.x + 0.1, -0.1)), V2((-0.1,-0.1)), speed, in_new_cmds)
340.
341.                                 if in_new_cmds:
342.                                     self.new_cmds_array.append((cmd, args[0], speed))

343.
344.
345.                             # wipe_region(z, gap = 0.08, speed = None)
346.                             # DON'T CORRECT FOR LOCAL_O
347.                             elif cmd == "wipe_region":
348.
349.                                 gap = args[1]
350.                                 speed = self.default_speed if len(args) < 3 else args
    [2]
351.
352.                                 for i in range(int(self.REGION_SIZE.y / (2*gap)) + 1)
    :
```

```
                                    step_time += self.render_write_line(ax, V2((-
1, gap * 2*i)), V2((self.REGION_SIZE.x + 1, gap * 2*i)), speed, in_new_cmds)
                                    step_time += self.render_write_line(ax, V2((self.
REGION_SIZE.x + 1, gap * (2*i + 1))), V2((-
1, gap * (2*i + 1))), speed, in_new_cmds)

                                if in_new_cmds:
                                    self.new_cmds_array.append((cmd, args[0], gap, sp
eed))


                            # move_to(point, ground_speed = None, laser_on = False)
                            elif cmd == "move_to":

                                if in_new_cmds:
                                    point = self.local_o + V2((args[0], args[1]))
                                    speed = self.default_speed if len(args) < 3 else
 args[2]

                                    ax.annotate(curr_command, xy=(self.curr_pos.x, se
lf.curr_pos.y))
                                    step_time += self.render_move_to_start(ax, point,
 speed)
                                    self.new_cmds_array.append((cmd, args[0], gap, sp
eed))


                            if in_new_cmds:
                                total_time += step_time
                                self.new_file_text += line.rstrip() + "\t\t# [{}]\n".
format(curr_command)
                            else:
                                self.new_file_text += line

                    # reset local origin for new commands
                    # NOTE: don't change the ## NEW header in the template sample fil
e
                    elif line[0:6] == "## NEW":

                        in_new_cmds = True

                        self.new_file_text = self.new_file_text.rstrip() + "\n\n# " +
 datetime.now(timezone("US/Eastern")).strftime("%Y-%m-%d %H:%M:%S") + "\n"
                        self.new_file_text += "LOCAL_O = V2((0, 0))\n"
                        self.local_o = V2((0, 0))
                        self.curr_pos = self.TR*(-1)

                        #curr_power = 0

                    elif line[0:6] == "## REF":

                        in_new_cmds = False
                        self.new_file_text += "\n\n\n\n## NEW COMMANDS\n\n\n\n\n\n" +
 line

                    # include newlines and comments
                    elif line[0] == "#" or not in_new_cmds:
                        self.new_file_text += line


            log_file.close()
```

```
402.
403.             plt.text(-
     110.0, 3.0, "{} s = {} min".format(int(10*total_time)/10.0, int(100*total_time/60
     .0)/100.0), fontsize=12)
404.             plt.show()
405.
406.         except FileNotFoundError:
407.             print("{}{} not found.".format(self.path_prefix, self.sample_name))
408.
409.
410.     def run(self, event):
411.         plt.close()
412.         self.continue_to_run = True
413.
414.
415.     def cancel(self, event):
416.         plt.close()
417.         self.continue_to_run = False
418.
419.
420.     # returns the duration of the write
421.     def render_write_line(self, ax, start, end, speed, is_new):
422.
423.         t = 0
424.         color = '#DF8800' if is_new else '#149E27'        # i.e., orange if new;
      else green
425.
426.         if is_new:
427.             t += self.render_move_to_start(ax, start)
428.
429.         l = mlines.Line2D([start.x, end.x], [start.y, end.y], color=color)
430.         ax.add_line(l)
431.         self.curr_pos.setXY(end)
432.         return t + (end - start.asV2()).magnitude / speed
433.
434.     def render_move_to_start(self, ax, start, speed = None):
435.
436.         if speed == None:
437.             speed = self.default_speed
438.
439.         if start != self.curr_pos.asV2():       # then at least calculate the tim
     e to move to start
440.             if (not self.connect_keithley):          # then map move to start
441.                 l = mlines.Line2D([self.curr_pos.x, start.x], [self.curr_pos.y, s
     tart.y], linestyle=':', color='lightgray')
442.                 ax.add_line(l)
443.             dist = (start - self.curr_pos.asV2()).magnitude
444.             self.curr_pos.setXY(start)
445.             return dist / speed
446.         return 0
447.
448.     def update_sample_history(self):
449.         log_file = open(self.path_prefix + self.sample_name + ".txt", "w")
450.         log_file.write(self.new_file_text)
451.         log_file.close()
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git
PROGRAM:    keithley_handler.py
AUTHOR:     T. Max Roberts
DATE:       --
FROM:       https://github.com/maxroberts/Keithley-2400-SourceMeter-
               Python-Interface/blob/master/keithley_serial.py

DESCRIPTION:
The keithley object that will manage the transmission of data and commands.
"""


#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time
import visa
import numpy as np


class KeithleyHandler:

    def __init__(self, addr = 24):
        try:
            rm = visa.ResourceManager()
            self.keith = rm.open_resource('GPIB0::%i::INSTR' %(addr))
            self.run_start_up_commands()
        except:
            print("Something went wrong...")
            sys.exit(0)

    def run_start_up_commands(self):
        for com in start_up_commands:
            self.send_command(com)
            time.sleep(.01)

    def send_command(self, command):
        response = self.keith.write(command)
        if response != '':
            return response
        else:
            return None

    """   all the useful commands as methods   """

    def reset(self):
        self.send_command('*RST')

    def set_output_on(self):
        self.send_command(':OUTP ON')

    def set_output_off(self):
        self.send_command(':OUTP OFF')

    def set_source_type(self, source_type):
        source_type = source_type.lower()
        if source_type == 'current' or source_type == 'curr' or source_type == 'c
':
            self.send_command(':SOUR:FUNC CURRENT')
```

```
60.             elif source_type == 'voltage' or source_type == 'volt' or source_type ==
    'v':
61.                 self.send_command(':SOUR:FUNC VOLT')
62.             else:
63.                 print("Unknown source setting!")
64.
65.         def set_source_voltage(self, voltage):
66.             if type(voltage) == int or type(voltage) == float:
67.                 if voltage > 200:
68.                     voltage = 200
69.                     print("Voltage limits are  +/- 200 V")
70.                 if voltage < -200:
71.                     voltage = -200
72.                     print("Voltage limits are  +/- 200 V")
73.                 self.set_source_type('voltage')
74.                 self.send_command(':SOUR:VOLT %s' %voltage)
75.             else:
76.                 print("Bad voltage sent")
77.
78.         def set_source_current(self, current):
79.             if type(current) == int or type(current) == float:
80.                 if current > 1:
81.                     current = 1
82.                     print("Current limits are  +/- 1 A")
83.                 if current < -1:
84.                     current = -1
85.                     print("Current limits are  +/- 1 A")
86.                 self.set_source_type('current')
87.                 self.send_command(':SOUR:CURR %s' %current)
88.             else:
89.                 print("Bad current sent")
90.
91.         def set_sensor_type(self, sensor_type):
92.             sensor_type = sensor_type.lower()
93.             if sensor_type == 'current' or sensor_type == 'curr' or sensor_type == 'c
    ':
94.                 self.send_command(':SENS:FUNC "CURR"')
95.                 return 'current'
96.             elif sensor_type == 'voltage' or sensor_type == 'volt' or sensor_type ==
    'v':
97.                 self.send_command(':SENS:FUNC "VOLT"')
98.                 return 'voltage'
99.             else:
100.                print("Unknown sensor setting!")
101.                return None
102.
103.        def set_sensor_range(self, sensor_type, sensor_range):
104.            sensor_type = self.set_sensor_type(sensor_type)
105.            print("Sensor type set to %s" %sensor_type)
106.            if type(sensor_range) == int or type(sensor_range) == float:
107.                order = int(('%.2E' %sensor_range)[5:])
108.                if sensor_type == 'voltage':
109.                    self.send_command(':SENS:VOLT:RANG 10E%s' %order)
110.                elif sensor_type == 'current':
111.                    self.send_command(':SENS:CURR:RANG 10E%s' %order)
112.                else:
113.                    print("Shouldn't get here!")
114.            else:
115.                print("Bad range sent.")
116.
117.        def set_voltage_compliance(self, limit):
```

```
        if type(limit) == int or type(limit) == float:
            coeff = float(('%.2E' %limit)[:3])
            order = int(('%.2E' %limit)[5:])
            self.send_command(':SENS:VOLT:PROT %sE%s' %(coeff, order))
        else:
            print("Bad compliance value sent.")

    def set_current_compliance(self, limit):
        if type(limit) == int or type(limit) == float:
            coeff = float(('%.2E' %limit)[:3])
            order = int(('%.2E' %limit)[5:])
            self.send_command(':SENS:CURR:PROT %sE%s' %(coeff, order))
        else:
            print("Bad compliance value sent.")

    def set_num_triggers(self, num):
        if type(num) == int:
            if num < 1:
                print("Number of triggers must be between 1-2500")
                print("Triggers set to 1")
                num = 1
            if num > 2500:
                print("Number of triggers must be between 1-2500")
                print("Triggers set to 2500")
                num = 2500
            self.send_command(':TRIG:COUN %s' %num)
        else:
            print("Bad trigger value sent.")

    def get_num_triggers(self):
        response = self.get_response(':TRIG:COUN?')
        return int(response.replace(' ', ''))

    def read(self, data_type=None):
        if data_type == None:
            self.send_command(':FORM:ELEM TIME, VOLT, CURR, RES')
        elif data_type.lower() == 'v':
            self.send_command(':FORM:ELEM TIME, VOLT')
        elif data_type.lower() == 'c':
            self.send_command(':FORM:ELEM TIME, CURR')
        elif data_type.lower() == 'r':
            self.send_command(':FORM:ELEM TIME, RES')
        else:
            self.send_command(':FORM:ELEM TIME, VOLT, CURR, RES')
        self.send_command(':OUTP ON')
        time.sleep(.25)
        self.send_command(':INIT')
        time.sleep(.25)
        # FETCH? waits for INIT to complete
        response = self.get_response(':FETCH?')
        time.sleep(.25)
        self.send_command(':OUTP OFF')
        return parse_data(response, data_type=data_type)


"""
This function parses the data returned from the Keithley.
The data is stuctured as {Volts, Amps, Ohms, Timestamp, Status}.
This creates an array of {Time, Volts, Amps, Ohms}.
Each element is a column vector for easy plotting.
KEYWORDS:
```

```
data_type = 'v' returns only time and volts, 'c', only time and current,
'r' only time and ohms.
"""
def parse_data(data, data_type=None):
    # Clean up the strings, splits into list
    data = data.replace(' ', '').split(',')
    # Reshape in sections
    if data_type in ['v', 'c', 'r']:
        cols = 2
    else:
        cols = 4
    # NOTE If data is not returned in groups of "cols" this will drop elements!
    data = zip(*[iter(data)]*cols)
    # Use an array as they can be indexed, and for type conversion
    data = np.array(data).astype(np.float)
    if data_type == None:
        return np.array([data[:,3], data[:,0], data[:,1], data[:,2]])
    elif data_type.lower() == 'v':
        return np.array([data[:,1], data[:,0]])
    elif data_type.lower() == 'c':
        return np.array([data[:,2], data[:,0]])
    elif data_type.lower() == 'r':
        return np.array([data[:,3], data[:,0]])
    else:
        print("Bad data_type given, returning full data set")
        return np.array([data[:,3], data[:,0], data[:,1], data[:,2]])

"""  Fast Settings  """
start_up_commands = ["*RST",
                     ":SYST:TIME:RES:AUTO 1",
                     ":SYST:BEEP:STAT 0",
                     ":SOUR:FUNC CURR",
                     ":SENS:FUNC:CONC OFF",
                     ":SENS:AVER:STAT OFF",
                     ":SENS:CURR:NPLC 0.01",
                     ":SENS:VOLT:NPLC 0.01",
                     ":SENS:RES:NPLC 0.01",
                     ":SENS:FUNC 'VOLT'",
                     ":SENS:VOLT:RANG 1e1",
                     ":TRIG:DEL 0.0",
                     ":SYST:AZER:STAT OFF",
                     ":SOUR:DELAY 0.0",
                     ":DISP:ENAB OFF"]

class FakeKeithley(object):
    def __init__(self):
        for method in Keithley.__dict__:
            self.__dict__[method] = self.fake_method
    def fake_method(self, *arg, **kwarg):
        pass
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git
PROGRAM:    coordinates.py
AUTHOR:     Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Used to manage a Cartesian coordinate system for stage positions. Vector
(V2) positions can be defined in polar or non-polar coordintes, as
detailed above __init__ below. Vectors can be added, subtracted, and
multiplied by scalars (note that scalar multiplication must be done as
V2 * k, not k * V2). V2 callable properties include magnitude, unit
(returns the object's unit vector, if it has one), and perpendicular
(_clk = clockwise; _cntclk = counterclockwise).
"""


import math

""" V FOR VECTOR """
class V2(object):

    def __init__(self):
        self.x = 0
        self.y = 0

    # arg can be type:
    #   V2 -> returns new V2 with same coordinates
    #   tuple of ints/floats:
    #       if polar, then  arg[0] = r and arg[1] = theta (deg)
    #       if nonpolar, then   arg[0] = x and arg[1] = y
    #   int/float -> returns a unit vector with theta (deg) = arg
    def __init__(self, arg, polar = False):
        if type(arg) is V2:
            self.x = arg.x
            self.y = arg.y
        elif type(arg) is tuple:
            if not polar:
                self.x = arg[0]
                self.y = arg[1]
            else:
                V2.__init__(self, V2(arg[1]) * arg[0])
        elif type(arg) is float or type(arg) is int:
            self.x = math.cos(math.radians(arg))
            self.y = math.sin(math.radians(arg))
        elif type(arg) is str:      # in format "(x, y)")
            x_str, y_str = arg.split(",")
            self.x = float(x_str[1:])
            self.y = float(y_str[:-1])
        else:
            self.x = 0
            self.y = 0

    @property
    def magnitude(self):
        return ((self.x**2 + self.y**2)**0.5)

    # returns self's unit vector
    @property
    def unit(self):
        if self.magnitude == 0:
```

```
62.                 return V2((0, 0))
63.             return self / self.magnitude
64.
65.         # returns a vector perpendicular to and of the same magnitude as
66.         # self, by clockwise rotation
67.         @property
68.         def perpendicular_clk(self):
69.             return V2((self.y, -self.x))
70.
71.         # returns a vector perpendicular to and of the same magnitude as
72.         # self, by counterclockwise rotation
73.         @property
74.         def perpendicular_cntclk(self):
75.             return V2((-self.y, self.x))
76.
77.         def asV2(self):
78.             return V2((self.x, self.y))
79.
80.         # sets the coordinates of self to match those of pt (V2)
81.         def setXY(self, pt):
82.             self.x = pt.x
83.             self.y = pt.y
84.
85.         def round(self):
86.             return V2((int(self.x), int(self.y)))
87.
88.         def __add__(self, other):
89.             return V2((self.x + other.x, self.y + other.y))
90.
91.         def __sub__(self, other):
92.             return V2((self.x - other.x, self.y - other.y))
93.
94.         def __mul__(self, scalar):
95.             return V2((scalar * self.x, scalar * self.y))
96.
97.         def __truediv__(self, inv_scalar):
98.             return V2((self.x / inv_scalar, self.y / inv_scalar))
99.
100.        def __abs__(self):
101.            return V2((abs(self.x), abs(self.y)))
102.
103.        def __str__(self):
104.            return "V2({0:.4f}, {1:.4f})".format(self.x, self.y)
105.
106.
107. # V3 removed
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    binaryserial.py
AUTHOR:     Zaber Technologies, Inc.
MODIFIED:   Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Class used to manage USB connection and data communication to devices.

I've modified this and other files to allow commands to be sent to a
device without waiting for a response that the command has succeeded.
Ordinarily, waiting for this kind of a response is helpful because it
can delay sending subsequent commands for the appropriate amount of
time, without requiring that that time be calculated. Because we're
using multiple devices, though, this means that if I send a command to
device X, then I can't also send a command to Y until X is finished;
this interferes with our ability to make simultaneous moves, so I've
made it so that you can send select commands without awaiting reply by
setting a parameter ("await_reply = False") in the call to that command,
e.g., x_axis.move_abs({20mm}, await_reply = False).
"""


import logging
import sys
import serial

from .binarycommand import BinaryCommand
from .binaryreply import BinaryReply
from .timeouterror import TimeoutError
from .portlock import PortLock

# See https://docs.python.org/2/howto/logging.html#configuring-logging-
# for-a-library for info on why we have these two lines here.
logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())

MESSAGE_LENGTH = 6


class BinarySerial(object):
    """A class for interacting with Zaber devices using the Binary protocol.

    This class defines a few simple methods for writing to and reading
    from a device connected over the serial port. It is safe to use in multi-
    threaded environments.

    Attributes:
        baudrate: An integer representing the desired communication baud rate.
            Valid bauds are 115200, 57600, 38400, 19200, and 9600.
        timeout: A number representing the number of seconds to wait for input
            before timing out. Floating-point numbers can be used to specify
            times shorter than one second. A value of None can also be used to
            specify an infinite timeout. A value of 0 specifies that all reads
            and writes should be non-blocking (return immediately without
            waiting). Defaults to 5.
        lock: The threading.RLock guarding the port. Each method takes the lock
            and is therefore thread safe. However, to ensure no other threads
            access the port across multiple method calls, the caller should
            acquire the lock and release it once all methods have returned.
```

```
62.       """
63.
64.       def __init__(self, port, baud=9600, timeout=5, inter_char_timeout=0.5):
65.           """Creates a new instance of the BinarySerial class.
66.
67.           Args:
68.               port: A string containing the name of the serial port to
69.                   which to connect.
70.               baud: An integer representing the baud rate at which to
71.                   communicate over the serial port.
72.               timeout: A number representing the number of seconds to wait
73.                   for a reply. Fractional numbers are accepted and can be
74.                   used to specify times shorter than a second.
75.               inter_char_timeout : A number representing the number of seconds
76.                   to wait between bytes in a reply. If your computer is bad at
77.                   reading incoming serial data in a timely fashion, try
78.                   increasing this value.
79.
80.           Notes:
81.               This class will open the port immediately upon
82.               instantiation. This follows the pattern set by PySerial,
83.               which this class uses internally to perform serial
84.               communication.
85.
86.           Raises:
87.               TypeError: The port argument passed was not a string.
88.           """
89.           if not isinstance(port, str):
90.               raise TypeError("port must be a string.")
91.           try:
92.               self._ser = serial.serial_for_url(port, do_not_open=True)
93.               self._ser.baudrate = baud
94.               self._ser.timeout = timeout
95.               self._ser.interCharTimeout = inter_char_timeout
96.               self._ser.open()
97.           except AttributeError:
98.               # serial_for_url not supported; use fallback
99.               self._ser = serial.Serial(port, baud, timeout=timeout,
100.                                         interCharTimeout=inter_char_timeout)
101.
102.          self._lock = PortLock()
103.
104.          self.outstanding_replies = [None] * 3
105.
106.      def write(self, *args):
107.          r"""Writes a command to the port.
108.
109.          This function accepts either a BinaryCommand object, a set
110.          of integer arguments, a list of integers, or a string.
111.          If passed integer arguments or a list of integers, those
112.          integers must be in the same order as would be passed to the
113.          BinaryCommand constructor (ie. device number, then command
114.          number, then data, and then an optional message ID).
115.
116.          Args:
117.              *args: A BinaryCommand to be sent, or between 2 and 4
118.                  integer arguments, or a list containing between 2 and
119.                  4 integers, or a string representing a
120.                  properly-formatted Binary command.
121.
122.          Notes:
```

```
            Passing integers or a list of integers is equivalent to
            passing a BinaryCommand with those integers as constructor
            arguments.

            For example, all of the following are equivalent::

                >>> write(BinaryCommand(1, 55, 1000))
                >>> write(1, 55, 1000)
                >>> write([1, 55, 1000])
                >>> write(struct.pack("<2Bl", 1, 55, 1000))
                >>> write('\x01\x37\xe8\x03\x00\x00')

        Raises:
            TypeError: The arguments passed to write() did not conform
                to the specification of ``*args`` above.
            ValueError: A string of length other than 6 was passed.
        """
        if len(args) == 1:
            message = args[0]
            if isinstance(message, list):
                message = BinaryCommand(*message)
        elif 1 < len(args) < 5:
            message = BinaryCommand(*args)  # pylint: disable=E1120
        else:
            raise TypeError("write() takes at least 1 and no more than 4 "
                            "arguments ({0:d} given)".format(len(args)))

        if isinstance(message, str):
            logger.debug("> %s", message)
            if len(message) != MESSAGE_LENGTH:
                raise ValueError("write of a string expects length 6.")

            # pyserial doesn't handle hex strings.
            if sys.version_info > (3, 0):
                data = bytes(message, "UTF-8")
            else:
                data = bytes(message)

        elif isinstance(message, BinaryCommand):
            data = message.encode()
            logger.debug("> %s", message)

        else:
            raise TypeError("write must be passed several integers, or a "
                            "string, list, or BinaryCommand.")

        with self._lock.write_lock:
            self._ser.write(data)

    def read(self, message_id=False):
        """Reads six bytes from the port and returns a BinaryReply.

        Args:
            message_id: True if the response is expected to have a
                message ID. Defaults to False.

        Returns:
            A BinaryCommand containing all of the information read from
            the serial port.

        Raises:
```

```
            zaber.serial.TimeoutError: No data was read before the
                specified timeout elapsed.
        """
        with self._lock.read_lock:
            reply = self._ser.read(MESSAGE_LENGTH)

        if len(reply) != MESSAGE_LENGTH:
            logger.debug("< Receive timeout!")
            raise TimeoutError("read timed out.")
        parsed_reply = BinaryReply(reply, message_id)
        logger.debug("< %s", parsed_reply)
        return parsed_reply


    def read_device(self, device_number, message_id=False):
        """Reads six bytes from the port and returns a BinaryReply.

        Args:
            device_number: The number of the device we're reading from.

        Returns:
            A BinaryReply containing all of the information read from
            the serial port.

        Raises:
            zaber.serial.TimeoutError: No data was read before the
                specified timeout elapsed.
        """

        device_reply_outstanding = self.outstanding_replies[device_number - 1]
        if device_reply_outstanding is not None:
            self.outstanding_replies[device_number - 1] = None
            return device_reply_outstanding

        with self._lock.read_lock:
            reply = self._ser.read(MESSAGE_LENGTH)

        if len(reply) != MESSAGE_LENGTH:
            logger.debug("< Receive timeout!")
            raise TimeoutError("read timed out.")
        parsed_reply = BinaryReply(reply, message_id)
        logger.debug("< %s", parsed_reply)


        # cmd-
    s 40 and 101 respond after setting device auto_reply...  it's important not to in
    terpret replies for cmd-s 40 or 101 as substance
        while (parsed_reply.command_number == 40) or (parsed_reply.command_number
     == 101) or (parsed_reply.device_number != device_number):

            if parsed_reply.device_number != device_number:
                self.outstanding_replies[parsed_reply.device_number - 1] = parsed
    _reply

            with self._lock.read_lock:
                reply = self._ser.read(MESSAGE_LENGTH)

            if len(reply) != MESSAGE_LENGTH:
                logger.debug("< Receive timeout!")
                raise TimeoutError("read timed out.")
            parsed_reply = BinaryReply(reply)
```

```
241.             logger.debug("< %s", parsed_reply)
242.
243.         if parsed_reply.command_number is 255:
244.             print("(S): ERROR")
245.             print(parsed_reply)
246.             sys.exit(0)
247.
248.         return parsed_reply
249.
250.     def flush(self):
251.         """Flushes the buffers of the underlying serial port."""
252.         with self._lock.write_lock:
253.             self._ser.flush()
254.
255.     def can_read(self):
256.         """Checks if enough data has been received to read a response, without bl
    ocking.
257.
258.         If the return value is True, it means at least six bytes are available
259.         to read from the serial port, so calling read() will not block.
260.
261.         Returns:
262.             True if a response is available to read; False otherwise.
263.         """
264.         with self._lock.read_lock:
265.             if hasattr(self._ser, "in_waiting"):
266.                 return self._ser.in_waiting >= MESSAGE_LENGTH
267.             else:
268.                 return self._ser.inWaiting() >= MESSAGE_LENGTH
269.
270.     def open(self):
271.         """Opens the serial port."""
272.         with self._lock:
273.             self._ser.open()
274.
275.     def close(self):
276.         """Closes the serial port."""
277.         with self._lock:
278.             self._ser.close()
279.
280.     def __enter__(self):
281.         return self
282.
283.     def __exit__(self, exc_type, exc_value, traceback):
284.         self.close()
285.
286.     @property
287.     def lock(self):
288.         return self._lock
289.
290.     @property
291.     def timeout(self):
292.         """The number of seconds to wait for input while reading.
293.
294.         The ``timeout`` property accepts floating point numbers for
295.         fractional wait times.
296.         """
297.         with self._lock:
298.             return self._ser.timeout
299.
300.     @timeout.setter
```

```
    def timeout(self, value):
        with self._lock:
            self._ser.timeout = value

    @property
    def baudrate(self):
        """The baud rate at which to read and write.

        The default baud rate for the Binary protocol is 9600. T-Series
        devices are only capable of communication at 9600 baud.
        A-Series devices can communicate at 115200, 57600, 38400,
        19200, and 9600 baud.

        Note that this changes the baud rate of the computer on which
        this code is running. It does not change the baud rate of
        connected devices.
        """
        with self._lock:
            return self._ser.baudrate

    @baudrate.setter
    def baudrate(self, value):
        if value not in (115200, 57600, 38400, 19200, 9600):
            raise ValueError("Invalid baud rate: {:d}. Valid baud rates are "
                             "115200, 57600, 38400, 19200, and 9600.".format(valu
    e))
        with self._lock:
            self._ser.baudrate = value
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    binarydevice.py
AUTHOR:     Zaber Technologies, Inc.
MODIFIED:   Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Object used to represent Zaber stages for sending commands.
"""


import time
import logging
import uuid

from random import randint
from .binarycommand import BinaryCommand
from .unexpectedreplyerror import UnexpectedReplyError

# See https://docs.python.org/2/howto/logging.html#configuring-logging-
# for-a-library for info on why we have these two lines here.
logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())


class BinaryDevice(object):
    """A class to represent a Zaber device in the Binary protocol. It is safe
    to use in multi-threaded environments.

    Attributes:
        port: A BinarySerial object which represents the port to which
            this device is connected.
        number: The integer number of this device. 1-255.
    """

    def __init__(self, port, number):
        """
        Args:
            port: A BinarySerial object to use as a parent port.
            number: An integer between 1 and 255 which is the number of
                this device.

        Raises:
            ValueError: The device number was invalid.
        """
        if number > 255 or number < 1:
            raise ValueError("Device number must be 1-255.")
        self.number = number
        self.port = port

    def send(self, *args, await_reply = False):
        """Sends a command to this device, then waits for a response.

        Args:
            *args: Either a single BinaryCommand, or 1-3 integers
                specifying, in order, the command number, data value,
                and message ID of the command to be sent.

        Notes:
            The ability to pass integers to this function is provided
```

```
62.                 as a convenience to the programmer. Calling
63.                 ``device.send(2)`` is equivalent to calling
64.                 ``device.send(BinaryCommand(device.number, 2))``.
65.
66.                 Note that in the Binary protocol, devices will only reply
67.                 once they have completed a command. Since this function
68.                 waits for a reply from the device, this function may block
69.                 for a long time while it waits for a response. For the same
70.                 reason, it is important to set the timeout of this device's
71.                 parent port to a value sufficiently high that any command
72.                 sent will be completed within the timeout.
73.
74.                 Regardless of the device address specified to this function,
75.                 the device number of the transmitted command will be
76.                 overwritten with the number of this device.
77.
78.                 If the command has a message ID set, this function will return
79.                 a reply with a message ID, after checking whether the message
80.                 IDs match.
81.
82.             Raises:
83.                 UnexpectedReplyError: The reply read was not sent by this
84.                     device or the message ID of the reply (if in use) did not
85.                     match the message ID of the command.
86.
87.             Returns: A BinaryReply containing the reply received.
88.             """
89.
90.             if len(args) == 1 and isinstance(args[0], BinaryCommand):
91.                 command = args[0]
92.             elif len(args) < 4:
93.                 command = BinaryCommand(self.number, *args)  # pylint: disable=E1120

94.
95.             """
96.             # I don't think it's necessary to prevent replies being received in the w
    rong order (e.g., by checking message_id) because a device responds "busy" (cmd 2
    55, data 255) if an additional command is given before the present one is complet
    ed.
97.             command.message_id = self.command_index
98.             self.command_index += 1
99.             """
100.
101.            with self.port.lock:
102.                if await_reply:
103.                    self.enable_auto_reply()
104.                    self.port.write(command)
105.                    return self.port.read_device(self.number)
106.                else:
107.                    if await_reply is not None:
108.                        self.disable_auto_reply()
109.                    self.port.write(command)
110.                    return None
111.
112.            """
113.            if ((reply.device_number != self.number) or
114.                (reply.message_id or 0) != (command.message_id or 0)):
115.
116.                print("%d != %d" % (reply.device_number, self.number))
117.                print("OR:  %d != %d" % ((reply.message_id or 0), (command.message_id
     or 0)))
```

```
                raise UnexpectedReplyError(
                    "Received an unexpected reply from"
                    "device number %d:\n%s" %(reply.device_number, reply)
                    )
            """
            #return reply


    def home(self, await_reply = False):
        return self.send(1, await_reply = await_reply)

    def move_abs(self, position, await_reply = False):
        return self.send(20, position, await_reply = await_reply)

    def move_rel(self, distance, await_reply = False):
        return self.send(21, distance, await_reply = await_reply)

    def move_vel(self, speed, await_reply = False):
        """
        Notes:
            Unlike the other "move" commands, the device replies
            immediately to this command. This means that when this
            function returns, it is likely that the device is still
            moving.
        """
        return self.send(22, speed, await_reply = await_reply)

    def stop(self):
        return self.send(23, await_reply = True)

    def set_home_speed(self, speed, await_reply = False):
        return self.send(41, speed, await_reply = await_reply)

    def set_target_speed(self, speed, await_reply = False):
        return self.send(42, speed, await_reply = await_reply)

    def set_acceleration(self, accel, await_reply = False):
        return self.send(43, accel, await_reply = await_reply)

    def set_min_position(self, position, await_reply = False):
        return self.send(106, position, await_reply = await_reply)

    def set_max_position(self, position, await_reply = False):
        return self.send(44, position, await_reply = await_reply)

    """
    bit:    "1" equals:     "1" means:
    0       1               disable auto-reply
    1       2               /reserved
    2       4               /reserved
    3       8               disable knob
    4       16              enable move tracking
    5       32              disable manual move tracking
    6       64              enable message IDs
    7       128             home status
    8       256             disable auto-home
    9       512             reverse knob
    10      1024            /reserved
    11      2048            /reserved
    12      4096            set home switch logic (device has active high home se
nsor)
```

```
    13      8192            /reserved
    14      16384           /reserved
    15      32768           /reserved
    """
    def get_device_mode(self):
        """
        default device mode for linear stages: 0
        """
        return self.get_setting(40)

    def set_device_mode(self, mode, await_reply = False):
        return self.send(40, mode, await_reply = await_reply)

    def enable_auto_reply(self):
        if self.number is 1:  # rotary stage
            self.port.write(BinaryCommand(self.number, 101, 0))
        else:
            self.port.write(BinaryCommand(self.number, 40, 0))

    def disable_auto_reply(self):
        if self.number is 1:  # rotary stage
            self.port.write(BinaryCommand(self.number, 101, 1))
        else:
            self.port.write(BinaryCommand(self.number, 40, 1))

    """
    # to make more complete: https://www.zaber.com/wiki/Manuals/Binary_Protocol_Manual#Quick_Command_Reference
    """

    def get_setting(self, setting_number):
        return self.send(53, setting_number, await_reply = True)#.data

    def get_status(self):
        return self.send(54, await_reply = True).data

    def get_position(self):
        return self.send(60, await_reply = True).data

    def disable_manual_move_tracking(self):
        return self.port.write(BinaryCommand(self.number, 116, 1))
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    binarycommand.py
AUTHOR:     Zaber Technologies, Inc.
MODIFIED:   Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Object used to pass commands to Zaber stages.
"""


import logging
import struct

# See https://docs.python.org/2/howto/logging.html#configuring-logging-
# for-a-library for info on why we have these two lines here.
logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())


class BinaryCommand(object):
    """Models a single command in Zaber's Binary protocol.

    Attributes:
        device_number: An integer representing the number (*a.k.a.*
            address) of the device to which to send the command. A
            device number of 0 indicates the command should be executed
            by all devices. 0-255.
        command_number: An integer representing the command to be sent
            to the device. Command numbers are listed in Zaber's
            `Binary Protocol Manual`_. 0-255.
        data: The data value to be transmitted with the command.
        message_id: The `message ID`_ of the command. 0-255, or None if
            not present.

    .. _Binary Protocol Manual: http://www.zaber.com/wiki/Manuals/Binary
        _Protocol_Manual#Quick_Command_Reference
    .. _message ID: http://www.zaber.com/wiki/Manuals/Binary_Protocol_Ma
        nual#Set_Message_Id_Mode_-_Cmd_102
    """
    def __init__(self, device_number, command_number, data=0,
                 message_id=None):
        """
        Args:
            device_number: An integer specifying the number of the
                target device to which to send this command. 0-255.
            command_number: An integer specifying the command to be
                sent. 0-255.
            data: An optional integer containing the data value to be
                sent with the command. When omitted, *data* will be set
                to 0.
            message_id: An optional integer specifying a message ID to
                give to the message. 0-255, or None if no message ID is
                to be used.

        Raises:
            ValueError: An invalid value was passed.
        """
        if device_number < 0 or command_number < 0:
            raise ValueError(
```

```
                "Device and command number must be between 0 and 255."
            )
        self.device_number = device_number
        self.command_number = command_number
        self.data = data
        if message_id is not None and (message_id < 0 or message_id > 255):
            raise ValueError("Message ID must be between 0 and 255.")
        self.message_id = message_id

    def encode(self):
        """Encodes a 6-byte byte string to be transmitted to a device.

        Returns:
            A byte string of length 6, formatted according to Zaber's
            `Binary Protocol Manual`_.
        """
        packed = struct.pack("<2Bl",
                             self.device_number,
                             self.command_number,
                             self.data)
        if self.message_id is not None:
            packed = packed[:5] + struct.pack("B", self.message_id)

        return packed

    def __str__(self):
        return "[{:d}, {:d}, {:d}, id={:d}]".format(self.device_number,
                                                    self.command_number,
                                                    self.data,
                                                    self.message_id)
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    binaryreply.py
AUTHOR:     Zaber Technologies, Inc.
MODIFIED:   Harper O. W. Wallace
DATE:       9 Apr 2020

DESCRIPTION:
Object used to pass commands to Zaber stages.
"""


import logging
import struct

# See https://docs.python.org/2/howto/logging.html#configuring-logging-
# for-a-library for info on why we have these two lines here.
logger = logging.getLogger(__name__)
logger.addHandler(logging.NullHandler())


class BinaryReply(object):
    """Models a single reply in Zaber's Binary protocol.

    Attributes:
        device_number: The number of the device from which this reply
            was sent.
        command_number: The number of the command which triggered this
            reply.
        data: The data value associated with the reply.
        message_id: The message ID number, if present, otherwise None.
    """
    def __init__(self, reply, message_id=False):
        """
        Args:
            reply: A byte string of length 6 containing a binary reply
                encoded according to Zaber's Binary Protocol Manual.
            message_id: True if a message ID should be extracted from
                the reply, False if not.

        Notes:
            Because a Binary reply's message ID truncates the last byte
            of the data value of the reply, it is impossible to tell
            whether a reply contains a message ID or not. Therefore, the
            user must specify whether or not a message ID should be
            assumed to be present.

        Raises:
            TypeError: An invalid type was passed as *reply*. This may
                indicate that a unicode string was passed instead of a
                binary (ascii) string.
        """
        if isinstance(reply, bytes):
            """ ORIGINAL LINE FROM ZABER: """
            #self.device_number, self.command_number, self.data = struct.unpack("<2Bl", reply)

            """ MODIFICATION BECAUSE IT LOOKS LIKE command_number COMES LAST: """
            self.device_number, self.command_number, self.data = struct.unpack("<2Bl", reply)
```

```
59.
60.                 if message_id and len(reply) == 6:
61.                     self.message_id = reply[5]
62.                     """
63.                     ZABER REPLY INTERPRETATION
64.
65.                     # Use bitmasks to extract the message ID.
66.                     self.message_id = (self.data & 0xFF000000) >> 24
67.
68.                     self.data = self.data & 0x00FFFFFF
69.
70.                     # Sign extend 24 to 32 bits in the message ID case.
71.                     # If the data is more than 24 bits it will still be wrong,
72.                     # but now negative smaller values will be right.
73.                     if 0 != (self.data & 0x00800000):
74.                         self.data = (int)((self.data | 0xFF000000) - (1 << 32))
75.                 """
76.                 else:
77.                     self.message_id = None
78.
79.             elif isinstance(reply, list):
80.                 # Assume a 4th element is a message ID.
81.                 if len(reply) > 3:
82.                     message_id = True
83.                 self.device_number = reply[0]
84.                 self.command_number = reply[1]
85.                 self.data = reply[2]
86.                 self.message_id = reply[3] if message_id else None
87.
88.             else:
89.                 raise TypeError("BinaryReply must be passed a byte string "
90.                                 "('bytes' type) or a list.")
91.
92.         def encode(self):
93.             """Returns the reply as a binary string, in the form in which it
94.             would appear if it had been read from the serial port.
95.
96.             Returns:
97.                 A byte string of length 6 formatted according to the Binary
98.                 Protocol Manual.
99.             """
100.            return struct.pack("<2Bl",
101.                               self.device_number,
102.                               self.command_number,
103.                               self.data)
104.
105.        def __str__(self):
106.            return "[{:d}, {:d}, {:d}]".format(self.device_number,
107.                                               self.command_number,
108.                                               self.data)
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    portlock.py
AUTHOR:     Zaber Technologies, Inc.
DATE:       9 Apr 2020

DESCRIPTION:
Class used to separate command read/write.
"""


from threading import RLock


class PortLock(object):
    """ A class used to provide separate read and write access in multithread environment.

    Provides properties readLock and writeLock returning separate RLocks to allow specific access.
    In addition provides same interface as threading.RLock. Using this interface both locks are
    acquired in defined order.
    """

    def __init__(self):
        self._read_lock = RLock()
        self._write_lock = RLock()

    def acquire(self, blocking=True):
        if blocking:
            self._read_lock.acquire()
            self._write_lock.acquire()
            return None
        else:
            if not self._read_lock.acquire(blocking=False):
                return False

            if not self._write_lock.acquire(blocking=False):
                self._read_lock.release()
                return False

            return True

    def release(self):
        self._write_lock.release()
        self._read_lock.release()

    __enter__ = acquire

    def __exit__(self, exc_type, exc_value, traceback):
        self.release()

    @property
    def read_lock(self):
        return self._read_lock

    @property
    def write_lock(self):
        return self._write_lock
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    unexpectedreplyerror.py
AUTHOR:     Zaber Technologies, Inc.
DATE:       9 Apr 2020

DESCRIPTION:
Error raised when reply received from unexpected device.
"""


class UnexpectedReplyError(Exception):
    """Raised when a reply was read from an unexpected device."""

    def __init__(self, message, reply=None):
        """
        Args:
            message: The error message to display.
            reply: The unexpected reply which caused this exception to
                be raised.

        Notes:
            This exception preserves the unexpected reply for which it
            was thrown. You can access it through the *reply* attribute.
        """
        super(UnexpectedReplyError, self).__init__(message)

        self.reply = reply
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    timeouterror.py
AUTHOR:     Zaber Technologies, Inc.
DATE:       9 Apr 2020

DESCRIPTION:
Error raised when the device takes too long to reply.
"""


class TimeoutError(Exception):
    """Raised when a read operation exceeded its specified time limit."""
    pass
```

```
"""
DIRECTORY:  https://github.com/howwallace/howw-stage-controls.git/zaber/serial/
PROGRAM:    utils.py
AUTHOR:     Zaber Technologies, Inc.
DATE:       9 Apr 2020

DESCRIPTION:
Python-version dependencies.
"""


import sys

IS_PYTHON3_PLUS = sys.version_info[0] >= 3


def isstring(anything):
    if IS_PYTHON3_PLUS:
        return isinstance(anything, str)
    else:
        return isinstance(anything, basestring)
```

Appendix E. Compiled MATLAB scripts used for LPL-M image analysis

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    find_region_of_interest.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script allows for streamlined region selection within LPL images using a dra
ggable tool.
ROI coordinates obtained using this tool are used in process_img.m.
}%


clear();

N = 4;
d = 180/N;

PATH = “/etc./...”;
SUFFIX = ".jpg";

RGB0 = imread(convertStringsToChars(PATH + SUFFIX));
%RGB90 = imread(convertStringsToChars(PATH + "90" + SUFFIX));

%Igray0 = rgb2gray(RGB0);

% ROI dimensions
X = 34;
Y = 34;

% Enlarge figure to full screen.
%set(gcf, 'Units', 'Normalized', 'OuterPosition', [0, 0.04, 1, 0.96]);

S = [0 0 X Y];  %the size of your ROI starts at point X1, Y1
ax1 = subplot(1, 1, 1);
imshow(RGB0);
h = imrect(ax1, S);

% position callback possible to allow title-
setting or printout of ROI coordinates
%addNewPositionCallback(h, @(p)title(mat2str(round([p(1,1) p(1,1)+X p(1,1)+X p(1,
1)])) + ", " + mat2str(round([p(1,2) p(1,2) p(1,2)+Y p(1,2)+Y]))));
%addNewPositionCallback(h, @(p)disp(mat2str(round([p(1,1) p(1,1)+X p(1,1)+X p(1,1
)])) + ", " + mat2str(round([p(1,2) p(1,2) p(1,2)+Y p(1,2)+Y]))));


% comparable to above, but for LPL image at 90 degrees
%{
ax2 = subplot(1, 2, 2);
imshow(RGB90);
hold on;
red_roi = plot(ax2, [0, X, X, 0], [0, 0, Y, Y], 'r', 'LineWidth', 2);
addNewPositionCallback(h, @(p)set(red_roi, 'xdata', round([p(1,1) p(1,1)+X p(1,1)
+X p(1,1) p(1,1)]), 'ydata', round([p(1,2) p(1,2) p(1,2)+Y p(1,2)+Y p(1,2)])));

fcn = makeConstrainToRectFcn('imrect',get(gca,'XLim'),get(gca,'YLim'));
setPositionConstraintFcn(h,fcn)
position = wait(h);
%}
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    process_img.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script calculates the average intensity over each aligned region (specified by x_coords
and y_coords, with width and height REGION_DIM, in units of pixels) and the average intensity
of each corresponding isotropic border (squarely spaced ISO_GAP pixels from the region and
with ISO_WIDTH pixels thickness, such that the area of the isotropic border is (REGION_DIM
+ 2*ISO_GAP + 2*ISO_WIDTH)^2 - (REGION_DIM + 2*ISO_GAP)^2) for each LPL image. Average
intensities of aligned regions are normalized against isotropic border intensities.
}%


clear();

% path and suffix to image files
PATH = “/etc./...”;
SUFFIX = ".jpg";

x_coords = [427, 512, 600, 688, 773, 856, 424, 512, 600, 685, 771, 856, 424, 512, 600, 685, 773, 859, 424, 512, 600, 683, 768, 856, 424, 512, 600, 683, 773, 859, 424, 509, 597, 685, 768, 856];
y_coords = [155, 155, 155, 155, 155, 157, 240, 243, 243, 243, 243, 243, 325, 325, 328, 328, 328, 328, 411, 413, 413, 413, 413, 416, 501, 499, 499, 501, 499, 501, 584, 587, 587, 587, 587, 589];

REGION_DIM = 34;
%REGION_DIM = 26;   % used for a series images taken at lower magnification


ISO_GAP = 8;
ISO_WIDTH = 8;

DISP_ROI = true;
USE_CELLS = false;
NUM_REGIONS = length(x_coords);
INSET = 0;

USE_MAX_POOL = false;
SUB_REGION_DIM = 2;

% used to iterate through image files, in format PATH + [val] + SUFFIX
IMAGE_CASES = [0:5:175]; %[0, 30, 45, 90]; %[0, 45, 90, 135]; %

%%%  INSERT REGIONS ABOVE  %%%


NUM_CASES = length(IMAGE_CASES);    % number of LPL angles
%d = 180/NUM_CASES;  % degree measure between angles

RGBs{NUM_CASES} = [];
Igrays{NUM_CASES} = [];
```

```
isos = [];
for r = 1:NUM_CASES
    img = imread(convertStringsToChars(PATH + int2str(IMAGE_CASES(r)) + SUFFIX));

    RGBs{r} = img;
    Igrays{r} = rgb2gray(img);     % double(1/2*img(:, :, 1) + 1/2*img(:, :, 2))
;
end


Igray0 = Igrays{1};
Igray90 = Igrays{NUM_CASES/2 + 1};


if ~USE_MAX_POOL
    SUB_REGION_DIM = REGION_DIM - 2*INSET;
    NUM_SUB_X = 1;
    NUM_SUB = 1;
else
    NUM_SUB_X = (REGION_DIM - 2*INSET) / SUB_REGION_DIM;
end

if DISP_ROI
    set(gcf, 'Position', get(0, 'Screensize'));

    subplot(1, 3, 1);
    imshow(RGBs{1}, []);
    title('Original RGB Image', 'FontSize', 12);

    subplot(1, 3, 2);
    imshow(Igray0, []);
    title('0∫ Grayscale Image', 'FontSize', 12);

    subplot(1, 3, 3);
    imshow(Igray90, []);
    title('90∫ Grayscale Image', 'FontSize', 12);
end

isoBWs = cell(NUM_REGIONS);
alignedBWs = cell(NUM_REGIONS, NUM_SUB_X, NUM_SUB_X);

for r = 1:NUM_REGIONS
    if USE_CELLS
        x0 = REGION_DIM * x_coords(r);
        y0 = REGION_DIM * y_coords(r);
    else
        x0 = x_coords(r);
        y0 = y_coords(r);
    end

    iso_edge = ISO_GAP + ISO_WIDTH;
    iso_xs = [x0 - iso_edge, x0 + REGION_DIM + iso_edge, x0 + REGION_DIM + iso_ed
ge, x0 - iso_edge, x0 - iso_edge, x0 - ISO_GAP, x0 - ISO_GAP, x0 + REGION_DIM + ISO_GAP
, x0 + REGION_DIM + ISO_GAP, x0 - iso_edge];
    iso_ys = [y0 - iso_edge, y0 - iso_edge, y0 + REGION_DIM + iso_edge, y0 + REGI
ON_DIM + iso_edge, y0 - ISO_GAP, y0 - ISO_GAP, y0 + REGION_DIM + ISO_GAP, y0 + REGION_D
IM + ISO_GAP, y0 - ISO_GAP, y0 - ISO_GAP];
    [BW, xs, ys] = roipoly(Igray0, iso_xs, iso_ys);
    isoBWs{r} = BW;

    if DISP_ROI
```

```
107.          subplot(1, 3, 2);
108.          hold on;
109.          plot(xs, ys, 'b', 'LineWidth', 2);
110.
111.          subplot(1, 3, 3);
112.          hold on;
113.          plot(xs, ys, 'b', 'LineWidth', 2);
114.      end
115.
116.      for sub_x = 1:NUM_SUB_X
117.          for sub_y = 1:NUM_SUB_X
118.              x = x0 + INSET + (sub_x - 1) * SUB_REGION_DIM;
119.              y = y0 + INSET + (sub_y - 1) * SUB_REGION_DIM;
120.
121.              [BW, xs, ys] = roipoly(Igray0, [x, x + SUB_REGION_DIM, x + SUB_REGION
_DIM, x], [y, y, y + SUB_REGION_DIM, y + SUB_REGION_DIM]);
122.
123.              alignedBWs{r, sub_x, sub_y} = BW;
124.
125.              if DISP_ROI
126.                  subplot(1, 3, 2);
127.                  hold on;
128.                  plot(xs, ys, 'r', 'LineWidth', 2);
129.
130.                  subplot(1, 3, 3);
131.                  hold on;
132.                  plot(xs, ys, 'r', 'LineWidth', 2);
133.              end
134.          end
135.      end
136.  end
137.
138.  iso_means = zeros(NUM_CASES, NUM_REGIONS);
139.  ali_means = zeros(NUM_CASES, NUM_REGIONS, NUM_SUB_X, NUM_SUB_X);
140.  adj_ali_means = zeros(NUM_CASES, NUM_REGIONS, NUM_SUB_X, NUM_SUB_X);
141.
142.  for c = 1:NUM_CASES
143.      for region = 1:NUM_REGIONS
144.          for sub_x = 1:NUM_SUB_X
145.              for sub_y = 1:NUM_SUB_X
146.                  ali_means(c, region, sub_x, sub_y) = mean2(Igrays{c}(alignedBWs{r
egion, sub_x, sub_y}));
147.              end
148.          end
149.          iso_means(c, region) = mean2(Igrays{c}(isoBWs{region}));
150.          adj_ali_means(c, region, :, :) = ali_means(c, region, :, :) - iso_means(c
, region);
151.      end
152.  end
153.
154.  iso_ave = mean2(iso_means);
155.  adj_ali_means = adj_ali_means + iso_ave;    % CORRECTED
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    fit_sine.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script uses average intensities through aligned regions, stored in
adj_ali_means or defects_removed (output values from process_img.m and
defects_removed.m, respectively) as the DATASET for three-point spline of
transmittances at LPL angles specified in ALIGN_ANGLES.

It outputs amplitude, midline, phase-shift, and corresponding theta_w,fit.

IMPORTANT NOTE:
To ensure proper functionality, be sure to run this script after running
process_img.m or defects_removed.m, and in the same MATLAB environment.
%}


DATASET = adj_ali_means; %defects_removed; %


inds = [0, 45, 90]/5 + 1;

T1 = pi / 180 * IMAGE_CASES(inds(1));
T2 = pi / 180 * IMAGE_CASES(inds(2));
T3 = pi / 180 * IMAGE_CASES(inds(3));

w = 2;
det = sin(w*T1)*(cos(w*T2) - cos(w*T3)) - cos(w*T1)*(sin(w*T2) - sin(w*T3)) + sin
(w*(T2 - T3));
inv = 1/det * [ cos(w*T2) - cos(w*T3), cos(w*T3) - cos(w*T1), cos(w*T1) - cos(w*T
2); sin(w*T3) - sin(w*T2), sin(w*T1) - sin(w*T3), sin(w*T2) - sin(w*T1); sin(w*(T2 - T3
)), sin(w*(T3 - T1)), sin(w*(T1 - T2)) ];

Ys = zeros(1, NUM_REGIONS);
phis = zeros(1, NUM_REGIONS);
bs = zeros(1, NUM_REGIONS);
angles = zeros(1, NUM_REGIONS);

for i = 1:NUM_REGIONS

    prod = inv * DATASET(inds, i, :);
    A = prod(1);
    B = prod(2);

    bs(i) = prod(3);

    Ys(i) = sqrt(A^2 + B^2);
    phis(i) = (180 / pi) * atan(B / A);

    if A < 0
        phis(i) = phis(i) + 180;
    end

    angles(i) = 180 - (phis(i) + 90) / 2;
end

fprintf('Y\t%s\n', num2str(Ys, '%f6\t'));
fprintf('phi\t%s\n', num2str(phis, '%f6\t'));
```

```
59.   fprintf('b\t%s\n', num2str(bs, '%f6\t'));
60.   fprintf('ANGLE\t%s\n', num2str(angles, '%f6\t'));
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    fit_sine_ls.m
AUTHOR:     Harper O. W. Wallace
DATE:       27 Apr 2020

DESCRIPTION:
This script uses average intensities through aligned regions, stored
inadj_ali_means or defects_removed (output values from process_img.m and
defects_removed.m, respectively) as the DATASET for least-squares sine
curve fitting over LPL angles specified in ALIGN_ANGLES.

It outputs amplitude, midline, phase-shift, and corresponding theta_w,fit.

IMPORTANT NOTE:
To ensure proper functionality, be sure to run this script after running
process_img.m or defects_removed.m, and in the same MATLAB environment.
%}


DATASET = adj_ali_means; %defects_removed; %iso_means; %

w = 2;

ALIGN_ANGLES = [0, 30, 45, 90] + 45; %IMAGE_CASES + 45; %IMAGE_CASES;

Ts = (pi / 180 * ALIGN_ANGLES).';
D = [cos(w * Ts), sin(w * Ts), ones(length(ALIGN_ANGLES),1)];
mult = inv(D.' * D) * D.';


Ys = zeros(1, NUM_REGIONS);
phis = zeros(1, NUM_REGIONS);
bs = zeros(1, NUM_REGIONS);
angles = zeros(1, NUM_REGIONS);

for i = 1:NUM_REGIONS

    prod = mult * DATASET((ALIGN_ANGLES - 45) / 5 + 1, i, :);  %DATASET(:, i, :);

    A = prod(1);
    B = prod(2);

    bs(i) = prod(3);

    Ys(i) = sqrt(A^2 + B^2);
    phis(i) = (180 / pi) * atan(B / A);

    if A < 0
        phis(i) = phis(i) + 180;
    end

    angles(i) = (270 - phis(i)) / 2;
end

fprintf('Y\t%s\n', num2str(Ys, '%f6\t'));
fprintf('phi\t%s\n', num2str(phis, '%f6\t'));
fprintf('b\t%s\n', num2str(bs, '%f6\t'));
fprintf('ANGLE\t%s\n', num2str(angles, '%f6\t'));
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    process_img_stability.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script calculates average intensities (in isolated red, green, and blue color channels)
over regions of interest in image stability analysis. Contrast ratios calculated according to
https://www.w3.org/TR/2008/REC-WCAG20-20081211/#contrast-ratiodef.
}%


clear();

%%%  INSERT REGIONS BELOW  %%%
PATH = “/etc./...”;
SUFFIX = ".jpg";

COORDS = [];
COORDS{1} = [[7.43661971830986 4.95774647887322 32.3474178403756 342.694835680751];
    [7.43661971830986 4.95774647887322 338.075117370892 32.8356807511738];
    [313.164319248826 4.95774647887322 32.3474178403757 341.868544600939];
    [7.43661971830977 314.816901408451 338.075117370892 32.0093896713616]];
COORDS{2} = [[87.2845528455284 52.2276422764227 177.804878048781 13.2520325203253];
    [53.3170731707317 87.2845528455284 12.5040650406504 174.943089430894];
    [286.552845528455 90.1463414634146 12.5040650406504 174.943089430894];
    [90.1463414634146 284.747967479675 177.804878048781 13.2520325203253]];
COORDS{3} = [[88.7719298245614 83.3684210526316 38.6315789473684 37.859649122807];
    [207.649122807018 81.8245614035088 54.8421052631579 39.4035087719298];
    [125.052631578947 206.877192982456 18.5614035087719 60.2456140350878];
    [209.19298245614 201.473684210526 18.5614035087719 60.2456140350878];
    [211.508771929825 239.298245614035 50.9824561403509 27.0526315789474];
    [92.6315789473684 240.070175438597 50.9824561403509 27.0526315789474]];
COORDS{4} = [[153.04347826087 92.5062801932367 46.2434782608695 163.236714975845];
    [97.2676328502415 130.597101449275 159.155555555555 31.9594202898551];
    [97.2676328502415 130.597101449275 26.5178743961352 61.887922705314];
    [229.905314009662 131.27729468599 26.5178743961352 61.887922705314]];

IMAGE_CASES = [0, 45, 90];

%%%  INSERT REGIONS ABOVE  %%%

NUM_REGIONS = length(COORDS);
NUM_SUB_REGIONS = [size(COORDS{1}, 1), size(COORDS{2}, 1), size(COORDS{3}, 1), size(COORDS{4}, 1)];
NUM_CASES = length(IMAGE_CASES);

INSET = 1.4;

set(gcf, 'Position', get(0, 'Screensize'));

RGBs{NUM_CASES} = [];
TOTAL_BWs{NUM_REGIONS} = [];

```

```
55.   for i = 1:NUM_CASES
56.
57.       RGBs{i} = imread(convertStringsToChars(PATH + int2str(IMAGE_CASES(i)) + SUFFI
X));
58.
59.       subplot(1, NUM_CASES, i);
60.       imshow(RGBs{i}, []);
61.       title(convertStringsToChars(IMAGE_CASES(i) + "ʃ RGB Image"), 'FontSize', 12);

62.       hold on;
63.
64.       red = RGBs{i}(:, :, 1);
65.       green = RGBs{i}(:, :, 2);
66.       blue = RGBs{i}(:, :, 3);
67.
68.       for r = 1:NUM_REGIONS
69.           TOTAL_BWs{r} = 0;
70.
71.           for sub = 1:NUM_SUB_REGIONS(r)
72.
73.               origin = COORDS{r}(sub, 1:2);
74.               size = COORDS{r}(sub, 3:4);
75.
76.               x_coords = [origin(1) + INSET, origin(1) + size(1) - 2*INSET, origin(
1) + size(1) - 2*INSET, origin(1) + INSET];
77.               y_coords = [origin(2) + INSET, origin(2) + INSET, origin(2) + size(2)
 - 2*INSET, origin(2) + size(2) - 2*INSET];
78.
79.               [BW, xs, ys] = roipoly(RGBs{i}, x_coords, y_coords);
80.
81.               if (isempty(TOTAL_BWs{r}))
82.                   TOTAL_BWs{r} = BW;
83.               else
84.                   TOTAL_BWs{r} = TOTAL_BWs{r} | BW;
85.               end
86.           end
87.
88.           region_mask = TOTAL_BWs{r};
89.           bounds = bwboundaries(region_mask);
90.           visboundaries(bounds);%, 'Color', colors(r));
91.
92.           fprintf("%iʃ, R %i:\t%f\t%f\t%f\n", IMAGE_CASES(i), r, mean2(red(region_m
ask)), mean2(green(region_mask)), mean2(blue(region_mask)));
93.       end
94.   end
95.
96.   %colors = ['g', 'r', 'b', 'y'];
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    remove_defects.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
Defects in alignment are identified based on the principle that well-aligned regions exhibit
a high degree of dichroism, which corresponds to a large difference in minimum and maximum
intensities across all LPLs (i.e., better-aligned regions are expected to have a wider range of
intensities across LPLs:  very low intensity at some LPL, theta, and very high intensity at the
orthogonal LPL, theta + pi/2). Conversely, the intensity of relatively poorly-aligned (or
defective) regions should not vary as widely across LPLs.

This script calculates the range of intensities (max - min) across all LPLs for each 2px-by-2px
(defined by SUB_REGION_DIM in process_img.m) subregion within an aligned region. It then
compares each subregion intensity range to the maximum subregion intensity range (i.e., to the
subregion with the greatest variation between high and low intensities) and flags subregions
with intensity ranges below THRESHOLD times this maximum. THRESHOLD = 0.6 = 60% is used in the
example of defect removal given in the Supplementary Information.

IMPORTANT NOTE:
defect_analysis.m is intended to be used only after images have been initially processed by
process_img.m. For scripting simplicity, several parameters referenced in this script are
defined in process_img.m. To ensure proper functionality, be sure to run this script after
running process_img.m, and in the same MATLAB environment.
%}


THRESHOLD = 0.6;
DISP_ROI = true;


max_diffs = zeros(NUM_REGIONS, NUM_SUB_X, NUM_SUB_X);
defect_indices = cell(NUM_REGIONS);
defect_count = zeros(1, NUM_REGIONS);
defects_removed = zeros(NUM_CASES, NUM_REGIONS);

for region = 1:NUM_REGIONS
    for sub_x = 1:NUM_SUB_X
        for sub_y = 1:NUM_SUB_X
            means = adj_ali_means(:, region, sub_x, sub_y);         % array of mean region intensity for each LPL image
            max_diffs(region, sub_x, sub_y) = max(means) - min(means);      % min-to-max range in intensity, across all LPL images
        end
    end

```

```
47.       max_region_diffs = max_diffs(region, :, :);
48.       [M, I] = max(max_region_diffs(:));
49.
50.       quality_indices = find(max_diffs(region, :, :) > M*THRESHOLD);
51.
52.       quality = adj_ali_means(:, region, quality_indices);
53.
54.       defects_removed(:, region) = reshape(mean(quality, 3), [NUM_CASES, 1]);
55.
56.       defect_indices{region} = find(max_diffs(region, :, :) < M*THRESHOLD);
57.       defect_count(region) = length(defect_indices{region});
58.
59.   end
60.
61.
62.   if size(adj_ali_means(3)) == 1
63.       for c = 1:NUM_CASES
64.           fprintf('%i\t%s\n', IMAGE_CASES(c), num2str(defects_removed(c, :, :), '%f
6\t'));
65.       end
66.       fprintf('DEFECTS\t%s\n', num2str(defect_count, '%d\t'));
67.       %{
68.       for region = 1:NUM_REGIONS
69.           disp(max(max_diffs(region, :)));
70.       end
71.       %}
72.   else
73.       for sub = 1:NUM_SUB
74.           fprintf('%s\n', num2str(max_diffs(:, sub), '%f6\t'));
75.       end
76.   end
77.
78.
79.   if DISP_ROI
80.       set(gcf, 'Position', get(0, 'Screensize'));
81.
82.       subplot(1, 3, 1);
83.       imshow(RGBs{1}, []);
84.       title('Original RGB Image', 'FontSize', 12);
85.
86.       subplot(1, 3, 2);
87.       imshow(Igray0, []);
88.       title('0∫ Grayscale Image', 'FontSize', 12);
89.
90.       subplot(1, 3, 3);
91.       imshow(Igray90, []);
92.       title('90∫ Grayscale Image', 'FontSize', 12);
93.
94.       for region = 1:NUM_REGIONS
95.           if USE_CELLS
96.               x0 = REGION_DIM * x_coords(region);
97.               y0 = REGION_DIM * y_coords(region);
98.           else
99.               x0 = x_coords(region);
100.              y0 = y_coords(region);
101.          end
102.
103.          % BLUE ISOTROPIC BORDERS
104.          iso_edge = ISO_GAP + ISO_WIDTH;
```

```
105.          iso_xs = [x0 - iso_edge, x0 + REGION_DIM + iso_edge, x0 + REGION_DIM + is
o_edge, x0 - iso_edge, x0 - iso_edge, x0 - ISO_GAP, x0 - ISO_GAP, x0 + REGION_DIM + ISO
_GAP, x0 + REGION_DIM + ISO_GAP, x0 - iso_edge];
106.          iso_ys = [y0 - iso_edge, y0 - iso_edge, y0 + REGION_DIM + iso_edge, y0 +
REGION_DIM + iso_edge, y0 - ISO_GAP, y0 - ISO_GAP, y0 + REGION_DIM + ISO_GAP, y0 + REGI
ON_DIM + ISO_GAP, y0 - ISO_GAP, y0 - ISO_GAP];
107.          [~, xs, ys] = roipoly(Igray0, iso_xs, iso_ys);
108.
109.          subplot(1, 3, 2);
110.          hold on;
111.          plot(xs, ys, 'b', 'LineWidth', 2);
112.
113.          subplot(1, 3, 3);
114.          hold on;
115.          plot(xs, ys, 'b', 'LineWidth', 2);
116.
117.
118.          % RED REGION BORDER
119.
120.          [BW, xs, ys] = roipoly(Igray0, [x0, x0 + REGION_DIM, x0 + REGION_DIM, x0]
, [y0, y0, y0 + REGION_DIM, y0 + REGION_DIM]);
121.
122.          subplot(1, 3, 2);
123.          hold on;
124.          plot(xs, ys, 'r', 'LineWidth', 2);
125.
126.          subplot(1, 3, 3);
127.          hold on;
128.          plot(xs, ys, 'r', 'LineWidth', 2);
129.
130.
131.          % GREEN DEFECT FLAGS
132.          max_region_diffs = max_diffs(region, :, :);
133.          [M, I] = max(max_region_diffs(:));
134.
135.          for i = 1:length(defect_indices{region})
136.              [sub_x, sub_y] = ind2sub([NUM_SUB_X, NUM_SUB_X], defect_indices{regio
n}(i));
137.
138.              x = x0 + (sub_x - 1) * SUB_REGION_DIM;
139.              y = y0 + (sub_y - 1) * SUB_REGION_DIM;
140.
141.              [BW, xs, ys] = roipoly(Igray0, [x, x + SUB_REGION_DIM, x + SUB_REGION
_DIM, x], [y, y, y + SUB_REGION_DIM, y + SUB_REGION_DIM]);
142.
143.              g = 1 - 0.4 * max_diffs(region, sub_x, sub_y) / (M*THRESHOLD);
   % more-certain defects are brighter
144.
145.              subplot(1, 3, 2);
146.              hold on;
147.              plot(xs, ys, 'Color', [0, g, 0], 'LineWidth', 2);
148.
149.              subplot(1, 3, 3);
150.              hold on;
151.              plot(xs, ys, 'Color', [0, g, 0], 'LineWidth', 2);
152.          end
153.      end
154.  end
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    disp_results.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script formats output values (either from process_img.m or from remove_defects.m; cf.
those files for further details and important notes) for ease of copy-and-paste to Microsoft
Excel for figure-creation and further analysis.

IMPORTANT NOTE:
disp_results.m references adj_ali_means or defects_removed, output values from process_img.m
and defects_removed.m, respectively, as the DATASET to be displayed. To ensure proper
functionality, be sure to run this script after running process_img.m or defects_removed.m,
and in the same MATLAB environment.
}%


DATASET = adj_ali_means; %defects_removed;
PRINT_FORMAT = 1;


if PRINT_FORMAT == 1
    for c = 1:NUM_CASES
        fprintf('%i\t%s\n', IMAGE_CASES(c), num2str(DATASET(c, :, :), '%f6\t'));

    end
    disp("ISOTROPIC:");
    for c = 1:NUM_CASES
        fprintf('%i\t%s\n', IMAGE_CASES(c), num2str(iso_means(c, :), '%f6\t'));
    end
else
    for region = 1:NUM_REGIONS
        for c = 1:NUM_CASES
            fprintf('%i\t%s\n', IMAGE_CASES(c), num2str(DATASET(c, region, :), '%f6\t'));
        end
        fprintf('\n');
    end
end

%{
ortho_adj_ali_means = zeros(NUM_CASES, NUM_REGIONS, NUM_SUB);
ortho_adj_ali_means(1:NUM_CASES/2, :, :) = DATASET(NUM_CASES/2 + 1:end, :, :);
ortho_adj_ali_means(NUM_CASES/2 + 1:end, :, :) = DATASET(1:NUM_CASES/2, :, :);

diffs = DATASET(:, :, :) - ortho_adj_ali_means(:, :, :);
%}

%{
for angle = 1:NUM_CASES
    region_aves = zeros(1, NUM_REGIONS); %mean2(adj_ali_means(1, :, :));
    ortho_region_aves = mean2(adj_ali_means(angle, :, :));
    for region = 1:NUM_REGIONS
        region_aves(1, region) = mean2(adj_ali_means(angle, region, :));
```

```
55.           ortho_region_aves(1, region) = mean2(ortho_adj_ali_means(angle, region, :
));
56.       end
57.   end
58.   %}
59.
60.   %disp(iso_ave);
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    convert_img.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script converts a pixelated input image (with “pixel” dimension D) to a matrix of
average grayscale intensities (scale 0 to 1), stored in an output variable aves. It is
intended to be used in conjunction with convert_data.m, which takes the output variable aves
and converts it to a matrix of alignment angles relative to 0-degrees LPL to achieve
transmittance values corresponding to aves.
}%


PATH = “/etc./.../img.jpg”;

original = imread(convertStringsToChars(PATH));
gray = rgb2gray(original);

%imshow(img);
hold on;

WIDTH = size(gray,1);
HEIGHT = size(gray,2);

D = 32;
%{
for k = 1:D:WIDTH
    x = [1 HEIGHT];
    y = [k k];
    plot(x,y,'Color','b','LineStyle','-');
end

for k = 1:D:HEIGHT
    x = [k k];
    y = [1 WIDTH];
    plot(x,y,'Color','b','LineStyle','-');
end
%}
W = floor(WIDTH / D);
H = floor(HEIGHT / D);

aves = zeros(W, H);
for x = 1:W
    for y = 1:H
        [BW, xs, ys] = roipoly(gray, [D*(y - 1), D*(y - 1), D*y, D*y], [D*(x - 1), D*x, D*x, D*(x - 1)]);
        aves(x, y) = mean2(gray(BW)) / 255; %round(N * mean2(gray(BW)) / 255) / N;
    end
    fprintf('%s\n', num2str(aves(x, :), '%d\t'));
end

imshow(aves)


```

```
56.   for x = 1:W
57.       for y = 1:H
58.           new_img(D*(x - 1) + 1:D*x + 1, D*(y - 1) + 1:D*y + 1) = aves(x, y);
59.       end
60.   end
61.
62.   aves
63.
64.   %imshow(new_img);
65.
66.   %{
67.   [BW, xs, ys] = roipoly(img, [0, 10, 10, 0], [0, 0, 10, 10]);
68.
69.   bounds = bwboundaries(BW);
70.   xy = bounds{1};
71.   x = xy(:, 2);
72.   y = xy(:, 1);
73.
74.   hold on;
75.   plot(x, y, 'r', 'LineWidth', 2);
76.   %}
```

```
%{
DIRECTORY:  https://github.com/howwallace/reczekj-et-al-2020.git
PROGRAM:    convert_data.m
AUTHOR:     Harper O. W. Wallace
DATE:       17 Jan 2020

DESCRIPTION:
This script converts transmittance values (either directly input on scale of 0 to 1, or
stored in aves, the output variable from convert_img.m) to alignment angles relative to
0-degrees LPL.

NOTE:
convert_data.m can be used with convert_img.m (cf. header for convert_img.m for further
details). To ensure proper functionality, be sure to run this script after running
convert_img.m, and in the same MATLAB environment.
}%


N = 8;  % BASE
POL = 0;
ROUND = 5;

M = [0, 0.5, 1.0; 0.5, 1.0, 0; 1.0, 0, 0.5];
%M = aves;  % if used in conjunction with convert_img.m

intensity_adj = 2 * M - 1

ALIGN_PLUS = 180 / pi * mod(POL + 1/2 * acos(intensity_adj) - pi / 2, pi);
ALIGN_MINUS = 180 / pi * mod(POL - 1/2 * acos(intensity_adj) - pi / 2, pi);

%{
OUTPUT INTENSITIES
1/2 * (cos(pi / 180 * (POL - ALIGN_PLUS)) + 1);
%}

for x = 1:size(ALIGN_PLUS,1)
    fprintf('%s\n', num2str(ROUND * round(ALIGN_PLUS(x, :) / ROUND), '%d\t'));
end


INTENSITIES = 1/2 * (cos(pi / 180 * 2 * (POL - ALIGN_MINUS) + pi) + 1);

%subplot(1, 2, 2);
imshow(INTENSITIES);
```

Appendix F. Compiled Java scripts used to simulate encryption schemes

```
PROGRAM:    TransmittanceEncrypt.java
AUTHOR:     Harper O. W. Wallace
DATE:       26 Dec 2017

DESCRIPTION:
This script converts a code expressed in binary values to alignment angles at
which a film will transmit below some threshold value (corresponding to "1"s;
above that value, corresponding to "0"s) on application of the specified angle
of LPL. It performs this conversion using a relationship similar but not
equivalent to the one derived empirically (we had not yet derived the empirical
relationship when this module was built), though the actual alignment values
are not used in any analysis, but rather only to demonstrate the principle.

The module presents an interactive window view that allows the user to adjust
LPL application angle and interpretation threshold to demonstrate the efficacy
of the encryption scheme it represents.
*/


import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import java.lang.Math;

import java.math.BigDecimal;

import java.util.ArrayList;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;


public class TransmittanceEncrypt
{
    final int[][] CODE_1 = new int[][]{
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0, 1, 0, 1}};
    final int[][] CODE_2 = new int[][]{
```

```
            {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
            {1, 0, 1, 0, 0, 1, 1, 0, 1, 1},
            {1, 0, 1, 0, 1, 0, 1, 1, 1, 0},
            {1, 1, 1, 0, 0, 0, 0, 0, 1, 1},
            {1, 0, 0, 1, 0, 0, 1, 0, 0, 0},
            {1, 1, 0, 0, 1, 0, 0, 1, 0, 1},
            {1, 0, 0, 0, 0, 0, 0, 1, 1, 0},
            {1, 0, 1, 1, 0, 0, 0, 0, 1, 1},
            {1, 0, 0, 0, 0, 0, 0, 1, 0, 0},
            {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};
        final int[][] CODE_3 = new int[][]{
            {1, 0, 1, 0, 1, 0, 1, 0, 1, 0},
            {1, 0, 1, 1, 0, 1, 0, 0, 0, 1},
            {1, 0, 0, 1, 0, 1, 1, 1, 1, 0},
            {1, 1, 1, 1, 1, 1, 1, 0, 1, 1},
            {1, 0, 1, 1, 0, 1, 0, 0, 1, 0},
            {1, 0, 1, 0, 1, 1, 1, 1, 0, 1},
            {1, 0, 1, 1, 1, 1, 1, 0, 0, 0},
            {1, 1, 1, 0, 1, 0, 0, 0, 1, 1},
            {1, 0, 1, 0, 0, 1, 1, 0, 1, 0},
            {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};
        final int[][] CODE_H = new int[][]{
            {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
            {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {1, 0, 0, 1, 0, 0, 1, 0, 0, 1},
            {1, 0, 0, 1, 0, 0, 1, 0, 0, 1},
            {1, 0, 0, 1, 1, 1, 1, 0, 0, 1},
            {1, 0, 0, 1, 0, 0, 1, 0, 0, 1},
            {1, 0, 0, 1, 0, 0, 1, 0, 0, 1},
            {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
            {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};

        /**********************/
        /**** USER-DEFINED ****/
        /**********************/

        final int[][] CODE = CODE_H;

        // 0 is pointing up from below; 180 is pointing down from above; -
    1 indicates the nth light angle is not used
        int THETA_1;
        int THETA_2;
        int THETA_3;
        double THRESHOLD;

        /**********************/
        /******* STATIC *******/
        /**********************/

        final int X_CELLS = CODE[0].length;
        final int Y_CELLS = CODE.length;

        JFrame frame = new JFrame("Transmittance Encrypt");
        JPanel panel = new JPanel();

        JLabel lightAngle1Label = new JLabel("  Light Angle 1 (0º)  ");
        JSlider lightAngle1Slider = new JSlider(JSlider.HORIZONTAL, 0, 180, 0);
        JLabel lightAngle2Label = new JLabel("  Light Angle 2 (-º)  ");
        JSlider lightAngle2Slider = new JSlider(JSlider.HORIZONTAL, 0, 180, 0);
```

```
122.      JLabel lightAngle3Label = new JLabel("  Light Angle 3 (-º)  ");
123.      JSlider lightAngle3Slider = new JSlider(JSlider.HORIZONTAL, 0, 180, 0);
124.
125.      JLabel thresholdLabel = new JLabel("    Threshold (100%)    ");
126.      JSlider thresholdSlider = new JSlider(JSlider.HORIZONTAL, 0, 100, 100);
127.
128.      JButton colorButton = new JButton("      Color      ");
129.
130.
131.      public static void main(String[] args)
132.      {
133.          // PASSED
134.          //Tester.exclusiveTestAllConditions();
135.          //Tester.negativeTestSingleThetaConditions();
136.          //Tester.negativeTestDoubleThetaConditions();
137.
138.          if (args.length != 4)
139.          {
140.              System.out.println("encode using java TransmittanceEncrypt [t1] [t2]
      [t3] [thresh]");
141.              return;
142.          }
143.
144.          int t1 = Integer.parseInt(args[0]);
145.          int t2 = Integer.parseInt(args[1]);
146.          int t3 = Integer.parseInt(args[2]);
147.          double thresh = Double.parseDouble(args[3]);
148.
149.          new TransmittanceEncrypt(t1, t2, t3, thresh);
150.      }
151.
152.      public TransmittanceEncrypt(int t1, int t2, int t3, double thresh)
153.      {
154.          CodeComponent codeComponent = new CodeComponent(X_CELLS, Y_CELLS);
155.          codeComponent.setTheta(1, 0);
156.          codeComponent.setTheta(2, -1);
157.          codeComponent.setTheta(3, -1);
158.          codeComponent.setThreshold(100);
159.
160.          if (t2 == -1)
161.              codeComponent.setCells(Encrypter.encrypt(CODE, t1, thresh));
162.          else if (t3 == -1)
163.              codeComponent.setCells(Encrypter.encrypt(CODE, t1, t2, thresh));
164.          else
165.              codeComponent.setCells(Encrypter.encrypt(CODE, t1, t2, t3, thresh));
166.
167.          panel.add(codeComponent);
168.          panel.add(lightAngle1Label);
169.          panel.add(lightAngle1Slider);
170.          panel.add(lightAngle2Label);
171.          panel.add(lightAngle2Slider);
172.          panel.add(lightAngle3Label);
173.          panel.add(lightAngle3Slider);
174.          panel.add(thresholdLabel);
175.          panel.add(thresholdSlider);
176.          panel.add(colorButton);
177.
178.          setupLightAngleSlider(codeComponent, lightAngle1Slider, lightAngle1Label,
      1);
```

```
179.          setupLightAngleSlider(codeComponent, lightAngle2Slider, lightAngle2Label,
      2);
180.          setupLightAngleSlider(codeComponent, lightAngle3Slider, lightAngle3Label,
      3);
181.
182.          thresholdSlider.setMajorTickSpacing(20);
183.          thresholdSlider.setMinorTickSpacing(10);
184.          thresholdSlider.setPaintTicks(true);
185.          thresholdSlider.setPaintLabels(true);
186.          thresholdSlider.addChangeListener(new ChangeListener()
187.          {
188.              public void stateChanged(ChangeEvent e)
189.              {
190.                  JSlider source = (JSlider)e.getSource();
191.                  codeComponent.setThreshold(source.getValue());
192.                  thresholdLabel.setText("    Threshold (" + source.getValue() + "%
      )    ");
193.              }
194.          });
195.
196.          colorButton.addActionListener(new ActionListener() {
197.              @Override
198.              public void actionPerformed(ActionEvent e) {
199.                  codeComponent.toggleColored();
200.              }
201.          });
202.
203.          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
204.          frame.add(panel);
205.          frame.pack();
206.          frame.setVisible(true);
207.
208.          int predictedW = 40 * X_CELLS + 20;
209.          frame.setSize(predictedW < 360 ? 360 : predictedW, 40 * Y_CELLS + 300);
210.      }
211.
212.      public void setupLightAngleSlider(CodeComponent comp, JSlider slider, JLabel
      label, int whichTheta)
213.      {
214.          slider.setMajorTickSpacing(60);
215.          slider.setMinorTickSpacing(30);
216.          slider.setPaintTicks(true);
217.          slider.setPaintLabels(true);
218.          slider.addChangeListener(new ChangeListener()
219.          {
220.              public void stateChanged(ChangeEvent e)
221.              {
222.                  JSlider source = (JSlider)e.getSource();
223.                  comp.setTheta(whichTheta, source.getValue());
224.                  label.setText("  Light Angle " + whichTheta + " (" + source.getVa
      lue() + "º)  ");
225.              }
226.          });
227.      }
228.  }
229.
230.
231.  class CodeComponent extends JComponent
232.  {
233.      final int CELL_SIZE = 40;
234.
```

```
235.      int xCells;
236.      int yCells;
237.
238.      BigDecimal theta1;      // light angle1 between 0 and 180 degrees
239.      BigDecimal theta2;      // light angle2
240.      BigDecimal theta3;      // light angle3
241.      BigDecimal threshold;   // how much light needed to signal
242.
243.      boolean thresholdApplied = false;
244.      boolean colored = false;
245.
246.      Cell[][] cells;
247.
248.      public CodeComponent(int xCells, int yCells)
249.      {
250.          this.xCells = xCells;
251.          this.yCells = yCells;
252.
253.          cells = new Cell[yCells][xCells];
254.
255.          setPreferredSize(new Dimension(xCells * CELL_SIZE, yCells * CELL_SIZE));

256.      }
257.
258.      public void setCells(BigDecimal[][] code)
259.      {
260.          for (int x = 0; x < xCells; x ++)
261.          {
262.              for (int y = 0; y < yCells; y ++)
263.                  cells[y][x] = new Cell(code[y][x]);
264.          }
265.      }
266.
267.      public void setTheta(int which, double theta)
268.      {
269.          switch (which)
270.          {
271.              case 1:     this.theta1 = new BigDecimal(theta % 180);
272.                          break;
273.              case 2:     this.theta2 = new BigDecimal(theta % 180);
274.                          break;
275.              case 3:     this.theta3 = new BigDecimal(theta % 180);
276.                          break;
277.              default:    break;
278.          }
279.
280.          if (this.colored)
281.              repaint();
282.      }
283.
284.      public void setThreshold(double threshold)
285.      {
286.          this.threshold = new BigDecimal(threshold / 100);
287.
288.          if (this.colored)
289.              repaint();
290.      }
291.
292.      public void toggleColored()
293.      {
294.          this.colored = !this.colored;
```

```
295.          repaint();
296.      }
297.
298.      public void paintComponent(Graphics g)
299.      {
300.          if (this.colored)
301.          {
302.              g.setColor(Color.BLACK);
303.              g.drawRect(0, 0, xCells * CELL_SIZE, yCells * CELL_SIZE);
304.
305.              for (int x = 0; x < xCells; x ++)
306.              {
307.                  for (int y = 0; y < yCells; y ++)
308.                  {
309.                      Cell cell = cells[y][x];
310.
311.                      BigDecimal signal;
312.
313.                      if (Conditions.isNilAngle(this.theta2))
314.                          signal = Conditions.signal(cell.polarization, this.theta1
    );
315.                      else if (Conditions.isNilAngle(this.theta3))
316.                          signal = Conditions.signal(cell.polarization, this.theta1
    , this.theta2);
317.                      else
318.                          signal = Conditions.signal(cell.polarization, this.theta1
    , this.theta2, this.theta3);
319.
320.                      double alpha = Conditions.willSignal(signal, threshold) ? 1.0
     : signal.doubleValue();
321.
322.                      g.setColor(new Color((float)0.0, (float)0.0, (float)0.0, (flo
    at)alpha));
323.                      g.fillRect((int)(x * CELL_SIZE), (int)(y * CELL_SIZE), CELL_S
    IZE, CELL_SIZE);
324.                  }
325.              }
326.          }
327.          else
328.          {
329.              int l = 8;  // shortest distance (orthogonal) between parallel lines

330.
331.              g.setColor(Color.BLACK);
332.              g.drawRect(0, 0, xCells * CELL_SIZE, yCells * CELL_SIZE);
333.
334.              for (int x = 0; x < xCells; x ++)
335.              {
336.                  for (int y = 0; y < yCells; y ++)
337.                  {
338.                      g.drawRect((int)(x * CELL_SIZE), (int)(y * CELL_SIZE), CELL_S
    IZE, CELL_SIZE);
339.
340.                      double angle = Math.toRadians(cells[y][x].polarization.double
    Value() % 180);
341.
342.                      double deltaX = -
    (double)(l) / Math.cos(angle);     // l * sec(theta)
343.                      double deltaY = (double)(l) / Math.sin(angle);      // l * cs
    c(theta)
344.
```

```
                    for (int line = 1; line < (double)(CELL_SIZE * Math.pow(2, 0.5)) / (double)(l); line ++)
                    {
                        // in case the part of the segment would otherwise appear outside the box
                        double adjustX = -(line * deltaY - CELL_SIZE) * Math.tan(angle);                        // delta * tan(theta)
                        double adjustY = Math.abs((line * Math.abs(deltaX) - CELL_SIZE) / Math.tan(angle));     // delta * cot(theta)

                        double startingX    = angle >= Math.PI / 2 ? x * CELL_SIZE : (x + 1) * CELL_SIZE;
                        double endingX      = angle >= Math.PI / 2 ? (x + 1) * CELL_SIZE : x * CELL_SIZE;

                        // still working in upper left half / upper right half
                        if (Math.abs(line * deltaX) <= (double)(CELL_SIZE) && line * deltaY <= (double)(CELL_SIZE))
                        {
                            g.drawLine((int)(startingX), (int)(y * CELL_SIZE + line * deltaY),
                                       (int)(startingX + line * deltaX), (int)(y * CELL_SIZE));
                        }
                        // x-coord beyond right boundary / left boundary
                        else if (Math.abs(line * deltaX) > (double)(CELL_SIZE) && line * deltaY <= CELL_SIZE)
                        {
                            g.drawLine((int)(startingX), (int)(y * CELL_SIZE + line * deltaY),
                                       (int)(endingX), (int)(y * CELL_SIZE + adjustY));
                        }
                        // y-coord beyond bottom boundary
                        else if (Math.abs(line * deltaX) <= (double)(CELL_SIZE) && line * deltaY > (double)(CELL_SIZE))
                        {
                            g.drawLine((int)(startingX + adjustX), (int)((y + 1) * CELL_SIZE),
                                       (int)(startingX + line * deltaX), (int)(y * CELL_SIZE));
                        }
                        else if (Math.abs(adjustX) <= CELL_SIZE && adjustY <= CELL_SIZE)
                        {
                            g.drawLine((int)(startingX + adjustX), (int)((y + 1) * CELL_SIZE),
                                       (int)(endingX), (int)(y * CELL_SIZE + adjustY));
                        }
                    }
                }
            }
        }
    }
}
```

```
/*
PROGRAM:    AlignmentView.java
AUTHOR:     Harper O. W. Wallace
DATE:       26 Dec 2017

DESCRIPTION:
This script renders the alignment angles of individual regions as parallel
lines, in order to illustrate the principles of independent alignment behind
transmittance- and alignment-based data encoding and encryption.
*/

import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import java.lang.Math;

import java.math.BigDecimal;

import java.util.ArrayList;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;

import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;


public class AlignmentView
{
    // 0 is pointing up from below; 180 is pointing down from above;

    /**********************/
    /**** USER-DEFINED ****/
    /**********************/

    final int[][] MARIO_ANGLES = new int[][]{
        {90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90},
        {90,90,90,90,90,90,90,35,35,35,35,35,90,90,65,65,65,90,90},
        {90,90,90,90,90,90,35,35,35,35,35,35,35,35,35,65,65,90,90},
        {90,90,90,90,90,90,30,30,30,65,65,180,65,90,35,35,35,90,90},
        {90,90,90,90,90,30,65,30,65,65,65,180,65,65,65,35,35,90,90},
        {90,90,90,90,90,30,65,30,30,65,65,65,180,65,65,65,35,90,90},
        {90,90,90,90,90,30,30,65,65,65,65,180,180,180,180,35,90,90,90},
        {90,90,90,90,90,90,90,65,65,65,65,65,65,65,35,35,90,90,90},
        {90,65,65,65,35,35,35,35,25,35,35,35,25,35,35,90,90,30,90},
        {90,65,65,65,35,35,35,35,35,25,35,35,35,25,90,90,30,30,90},
        {90,90,65,90,90,90,35,35,35,25,25,25,25,70,25,25,30,30,90},
        {90,90,90,90,90,90,90,25,25,25,70,25,25,25,25,25,30,30,90},
        {90,90,90,90,90,30,30,25,25,25,25,25,25,25,25,25,30,30,90},
```

```
            {90,90,90,90,30,30,30,25,25,25,25,25,25,90,90,90,90,90,90},
            {90,90,90,90,30,30,90,90,90,90,90,90,90,90,90,90,90,90,90},
            {90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90,90}};

        final int[][] ANGLES = MARIO_ANGLES;

        /**********************/
        /******* STATIC *******/
        /**********************/

        final int X_CELLS = ANGLES[0].length;
        final int Y_CELLS = ANGLES.length;

        JFrame frame = new JFrame("AlignmentView");
        JPanel panel = new JPanel();

        JLabel lightAngleLabel = new JLabel("  Light Angle: 0º  ");
        JSlider lightAngleSlider = new JSlider(JSlider.HORIZONTAL, 0, 180, 0);

        JButton colorButton = new JButton("      Color      ");


        public static void main(String[] args)
        {
            // PASSED
            //Tester.exclusiveTestAllConditions();

            //Tester.negativeTestSingleThetaConditions();
            //Tester.negativeTestDoubleThetaConditions();

            if (args.length != 1)
            {
                System.out.println("encode using: java BioEncrpyion [line_spacing]");

                return;
            }

            int l_space = Integer.parseInt(args[0]);

            new AlignmentView(l_space);
        }

        public AlignmentView(int l_space)
        {
            CodeComponent codeComponent = new CodeComponent(ANGLES, l_space);
            codeComponent.setTheta(0);

            panel.add(codeComponent);
            panel.add(lightAngleLabel);
            panel.add(lightAngleSlider);

            panel.add(colorButton);


            lightAngleSlider.setMajorTickSpacing(60);
            lightAngleSlider.setMinorTickSpacing(30);
            lightAngleSlider.setPaintTicks(true);
            lightAngleSlider.setPaintLabels(true);
            lightAngleSlider.addChangeListener(new ChangeListener()
            {
                public void stateChanged(ChangeEvent e)
```

```
122.             {
123.                 JSlider source = (JSlider)e.getSource();
124.                 codeComponent.setTheta((int)(source.getValue()));
125.                 lightAngleLabel.setText("  Light Angle: " + (int)(source.getValue
     ()) + "º  ");
126.             }
127.         });
128.
129.         colorButton.addActionListener(new ActionListener() {
130.             @Override
131.             public void actionPerformed(ActionEvent e) {
132.                 codeComponent.toggleColored();
133.             }
134.         });
135.
136.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
137.         frame.add(panel);
138.         frame.pack();
139.         frame.setVisible(true);
140.
141.         int predictedW = 40 * X_CELLS + 20;
142.         frame.setSize(predictedW < 360 ? 360 : predictedW, 40 * Y_CELLS + 300);
143.     }
144. }
145.
146.
147. class CodeComponent extends JComponent
148. {
149.     final int CELL_SIZE = 40;
150.
151.     int[][] cells;
152.     int x_cells;
153.     int y_cells;
154.
155.     int l_space;            // spacing between lines in alignment view
156.     int theta;              // light angle between 0 and 180 degrees
157.
158.     boolean colored = false;
159.
160.
161.     public CodeComponent(int[][] _cells, int _l_space)
162.     {
163.         this.cells = _cells;
164.         this.x_cells = _cells[0].length;
165.         this.y_cells = _cells.length;
166.
167.         this.l_space = _l_space;
168.
169.         setPreferredSize(new Dimension(this.x_cells * CELL_SIZE, this.y_cells * C
     ELL_SIZE));
170.     }
171.
172.     public void setTheta(int theta)
173.     {
174.         this.theta = theta % 180;
175.
176.         if (this.colored)
177.             repaint();
178.     }
179.
180.     public void toggleColored()
```

```
181.     {
182.         this.colored = !this.colored;
183.         repaint();
184.     }
185.
186.     public void paintComponent(Graphics g)
187.     {
188.         if (this.colored)
189.         {
190.             g.setColor(Color.BLACK);
191.             g.drawRect(0, 0, this.x_cells * CELL_SIZE, this.y_cells * CELL_SIZE);
192.
193.             for (int x = 0; x < this.x_cells; x ++)
194.             {
195.                 for (int y = 0; y < this.y_cells; y ++)
196.                 {
197.                     int angle = this.cells[y][x];
198.
199.                     double alpha = Math.pow(Math.cos(Math.toRadians(angle - this.
    theta)), 2);
200.
201.                     g.setColor(new Color((float)0.0, (float)0.0, (float)0.0, (flo
    at)alpha));
202.                     g.fillRect((int)(x * CELL_SIZE), (int)(y * CELL_SIZE), CELL_S
    IZE, CELL_SIZE);
203.                 }
204.             }
205.         }
206.         else
207.         {
208.             g.setColor(Color.BLACK);
209.             g.drawRect(0, 0, this.x_cells * CELL_SIZE, this.y_cells * CELL_SIZE);
210.
211.             for (int x = 0; x < this.x_cells; x ++)
212.             {
213.                 for (int y = 0; y < this.y_cells; y ++)
214.                 {
215.                     g.setColor(Color.BLACK);
216.                     g.drawRect((int)(x * CELL_SIZE), (int)(y * CELL_SIZE), CELL_S
    IZE, CELL_SIZE);
217.
218.                     // lines
219.                     g.setColor(new Color(0, 0, 0, 100));
220.
221.                     if ((this.cells[y][x] + 90) % 180 == 0)
222.                     {
223.                         for (int line = 1; line < (double)(CELL_SIZE) / (double)(
    this.l_space); line ++)
224.                         {
225.                             g.drawLine(x * CELL_SIZE + line*this.l_space, y * CEL
    L_SIZE, x * CELL_SIZE + line*this.l_space, (y + 1) * CELL_SIZE);
226.                         }
227.                     }
228.                     else
229.                     {
230.                         double angle = Math.toRadians((this.cells[y][x] + 90) % 1
    80);
231.
```

```
                        double deltaX = -(double)(this.l_space) / Math.cos(angle);      // l * sec(theta)
                        double deltaY = (double)(this.l_space) / Math.sin(angle);       // l * csc(theta)

                        for (int line = 1; line < (double)(CELL_SIZE * Math.pow(2, 0.5)) / (double)(this.l_space); line ++)
                        {
                            // in case the part of the segment would otherwise appear outside the box
                            double adjustX = -(line * deltaY - CELL_SIZE) * Math.tan(angle);                        // delta * tan(theta)
                            double adjustY = Math.abs((line * Math.abs(deltaX) - CELL_SIZE) / Math.tan(angle));     // delta * cot(theta)

                            double startingX    = angle >= Math.PI / 2 ? x * CELL_SIZE : (x + 1) * CELL_SIZE;
                            double endingX      = angle >= Math.PI / 2 ? (x + 1) * CELL_SIZE : x * CELL_SIZE;

                            // still working in upper left half / upper right half
                            if (Math.abs(line * deltaX) <= (double)(CELL_SIZE) && line * deltaY <= (double)(CELL_SIZE))
                            {
                                g.drawLine((int)(startingX), (int)(y * CELL_SIZE + line * deltaY),
                                           (int)(startingX + line * deltaX), (int)(y * CELL_SIZE));
                            }
                            // x-coord beyond right boundary / left boundary
                            else if (Math.abs(line * deltaX) > (double)(CELL_SIZE) && line * deltaY <= CELL_SIZE)
                            {
                                g.drawLine((int)(startingX), (int)(y * CELL_SIZE + line * deltaY),
                                           (int)(endingX), (int)(y * CELL_SIZE + adjustY));
                            }
                            // y-coord beyond bottom boundary
                            else if (Math.abs(line * deltaX) <= (double)(CELL_SIZE) && line * deltaY > (double)(CELL_SIZE))
                            {
                                g.drawLine((int)(startingX + adjustX), (int)((y + 1) * CELL_SIZE),
                                           (int)(startingX + line * deltaX), (int)(y * CELL_SIZE));
                            }
                            else if (Math.abs(adjustX) <= CELL_SIZE && adjustY <= CELL_SIZE)
                            {
                                g.drawLine((int)(startingX + adjustX), (int)((y + 1) * CELL_SIZE),
                                           (int)(endingX), (int)(y * CELL_SIZE + adjustY));
                            }
                        }
                    }
                }
            }
```

```
271.          }
272.      }
273.  }
```

```java
/*
PROGRAM:    Encrypter.java
AUTHOR:     Harper O. W. Wallace
DATE:       26 Dec 2017

DESCRIPTION:
This script does calculations to determine alignment angles that will yield the
correct transmittances at a given angle of LPL and transmittance threshold. It
performs this conversion using a relationship similar but not equivalent to the
one derived empirically (we had not yet derived the empirical relationship when
this module was built), though the actual alignment values are not used in any
analysis, but rather only to demonstrate the principle.
*/

import java.math.BigDecimal;
import java.math.RoundingMode;

public class Encrypter
{
    public static final BigDecimal NINETY = new BigDecimal(90);

    // code is non-empty 2-D array of 1s and 0s;  theta1,2,and3 are light angles (degrees, between 0 and 180);  threshold between 0.00 and 1.00
    public static BigDecimal[][] encrypt(int[][] code, double theta, double threshold)
    {
        BigDecimal[][] encrypted = new BigDecimal[code.length][code[0].length];

        // POSITIVE CONDITIONS

        final BigDecimal[] pBounds1 = Conditions.singleThetaCondition1Bounds(theta, threshold);
        final BigDecimal[] pBounds2 = Conditions.singleThetaCondition2Bounds(theta, threshold);

        final double pRange1 = Conditions.range(pBounds1);
        final double pRange2 = Conditions.range(pBounds2);
        final double totalPRange = pRange1 + pRange2;

        // NEGATIVE CONDITIONS

        final BigDecimal[] nBounds1 = Conditions.singleThetaNegativeCondition1Bounds(theta, threshold);
        final BigDecimal[] nBounds2 = Conditions.singleThetaNegativeCondition2Bounds(theta, threshold);
        final BigDecimal[] nBounds3 = Conditions.singleThetaNegativeCondition3Bounds(theta, threshold);

        final double nRange1 = Conditions.range(nBounds1);
        final double nRange2 = Conditions.range(nBounds2);
        final double nRange3 = Conditions.range(nBounds3);
        final double totalNRange = nRange1 + nRange2 + nRange3;

        for (int y = 0; y < code.length; y ++)
        {
            for (int x = 0; x < code[y].length; x ++)
            {
                if (code[y][x] == 1)
                {
                    final double random = Math.random();
```

```
54.
55.                         if (random < (pRange1) / totalPRange)
56.                             encrypted[y][x] = Conditions.randomAngleBetween(pBounds1)
    ;
57.                         else
58.                             encrypted[y][x] = Conditions.randomAngleBetween(pBounds2)
    ;
59.                     }
60.                     else
61.                     {
62.                         final double random = Math.random();
63.
64.                         if (random < (nRange1) / totalNRange)
65.                             encrypted[y][x] = Conditions.randomAngleBetween(nBounds1)
    ;
66.                         else if (random < (nRange1 + nRange2) / totalNRange)
67.                             encrypted[y][x] = Conditions.randomAngleBetween(nBounds2)
    ;
68.                         else
69.                             encrypted[y][x] = Conditions.randomAngleBetween(nBounds3)
    ;
70.                     }
71.                 }
72.             }
73.             return encrypted;
74.         }
75.
76.         public static BigDecimal[][] encrypt(int[][] code, double theta1, double thet
    a2, double threshold)
77.         {
78.             BigDecimal[][] encrypted = new BigDecimal[code.length][code[0].length];
79.
80.             // POSITIVE CONDITIONS
81.
82.             final BigDecimal[] pBounds1 = Conditions.doubleThetaCondition1Bounds(thet
    a1, theta2, threshold);
83.             final BigDecimal[] pBounds2 = Conditions.doubleThetaCondition2Bounds(thet
    a1, theta2, threshold);
84.             final BigDecimal[] pBounds3 = Conditions.doubleThetaCondition3Bounds(thet
    a1, theta2, threshold);
85.             final BigDecimal[] pBounds4 = Conditions.doubleThetaCondition4Bounds(thet
    a1, theta2, threshold);
86.             final BigDecimal[] pBounds5 = Conditions.doubleThetaCondition5Bounds(thet
    a1, theta2, threshold);
87.             final BigDecimal[] pBounds6 = Conditions.doubleThetaCondition6Bounds(thet
    a1, theta2, threshold);
88.             final BigDecimal[] pBounds7 = Conditions.doubleThetaCondition7Bounds(thet
    a1, theta2, threshold);
89.             final BigDecimal[] pBounds8 = Conditions.doubleThetaCondition8Bounds(thet
    a1, theta2, threshold);
90.             final BigDecimal[] pBounds9 = Conditions.doubleThetaCondition9Bounds(thet
    a1, theta2, threshold);
91.
92.             final double pRange1 = Conditions.range(pBounds1);
93.             final double pRange2 = Conditions.range(pBounds2);
94.             final double pRange3 = Conditions.range(pBounds3);
95.             final double pRange4 = Conditions.range(pBounds4);
96.             final double pRange5 = Conditions.range(pBounds5);
97.             final double pRange6 = Conditions.range(pBounds6);
98.             final double pRange7 = Conditions.range(pBounds7);
99.             final double pRange8 = Conditions.range(pBounds8);
```

```
100.          final double pRange9 = Conditions.range(pBounds9);
101.          final double totalPRange = pRange1 + pRange2 + pRange3 + pRange4 + pRange
      5 + pRange6 + pRange7 + pRange8 + pRange9;
102.
103.          // NEGATIVE CONDITIONS
104.
105.          final BigDecimal[] nBounds1 = Conditions.doubleThetaNegativeCondition1Bou
      nds(theta1, theta2, threshold);
106.          final BigDecimal[] nBounds2 = Conditions.doubleThetaNegativeCondition2Bou
      nds(theta1, theta2, threshold);
107.          final BigDecimal[] nBounds3 = Conditions.doubleThetaNegativeCondition3Bou
      nds(theta1, theta2, threshold);
108.          final BigDecimal[] nBounds4 = Conditions.doubleThetaNegativeCondition4Bou
      nds(theta1, theta2, threshold);
109.          final BigDecimal[] nBounds5 = Conditions.doubleThetaNegativeCondition5Bou
      nds(theta1, theta2, threshold);
110.          final BigDecimal[] nBounds6 = Conditions.doubleThetaNegativeCondition6Bou
      nds(theta1, theta2, threshold);
111.          final BigDecimal[] nBounds7 = Conditions.doubleThetaNegativeCondition7Bou
      nds(theta1, theta2, threshold);
112.          final BigDecimal[] nBounds8 = Conditions.doubleThetaNegativeCondition8Bou
      nds(theta1, theta2, threshold);
113.          final BigDecimal[] nBounds9 = Conditions.doubleThetaNegativeCondition9Bou
      nds(theta1, theta2, threshold);
114.
115.          final double nRange1 = Conditions.range(nBounds1);
116.          final double nRange2 = Conditions.range(nBounds2);
117.          final double nRange3 = Conditions.range(nBounds3);
118.          final double nRange4 = Conditions.range(nBounds4);
119.          final double nRange5 = Conditions.range(nBounds5);
120.          final double nRange6 = Conditions.range(nBounds6);
121.          final double nRange7 = Conditions.range(nBounds7);
122.          final double nRange8 = Conditions.range(nBounds8);
123.          final double nRange9 = Conditions.range(nBounds9);
124.          final double totalNRange = nRange1 + nRange2 + nRange3 + nRange4 + nRange
      5 + nRange6 + nRange7 + nRange8 + nRange9;
125.
126.
127.          for (int y = 0; y < code.length; y ++)
128.          {
129.              for (int x = 0; x < code[y].length; x ++)
130.              {
131.                  if (code[y][x] == 1)
132.                  {
133.                      final double random = Math.random();
134.
135.                      if (random < (pRange1) / totalPRange)
136.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds1)
      ;
137.                      else if (random < (pRange1 + pRange2) / totalPRange)
138.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds2)
      ;
139.                      else if (random < (pRange1 + pRange2 + pRange3) / totalPRange
      )
140.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds3)
      ;
141.                      else if (random < (pRange1 + pRange2 + pRange3 + pRange4) / t
      otalPRange)
142.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds4)
      ;
```

```
143.                     else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pR
    ange5) / totalPRange)
144.                         encrypted[y][x] = Conditions.randomAngleBetween(pBounds5)
    ;
145.                     else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pR
    ange5 + pRange6) / totalPRange)
146.                         encrypted[y][x] = Conditions.randomAngleBetween(pBounds6)
    ;
147.                     else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pR
    ange5 + pRange6 + pRange7) / totalPRange)
148.                         encrypted[y][x] = Conditions.randomAngleBetween(pBounds7)
    ;
149.                     else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pR
    ange5 + pRange6 + pRange7 + pRange8) / totalPRange)
150.                         encrypted[y][x] = Conditions.randomAngleBetween(pBounds8)
    ;
151.                     else
152.                         encrypted[y][x] = Conditions.randomAngleBetween(pBounds9)
    ;
153.                 }
154.                 else
155.                 {
156.                     final double random = Math.random();
157.
158.                     if (random < (nRange1) / totalNRange)
159.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds1)
    ;
160.                     else if (random < (nRange1 + nRange2) / totalNRange)
161.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds2)
    ;
162.                     else if (random < (nRange1 + nRange2 + nRange3) / totalNRange
    )
163.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds3)
    ;
164.                     else if (random < (nRange1 + nRange2 + nRange3 + nRange4) / t
    otalNRange)
165.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds4)
    ;
166.                     else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nR
    ange5) / totalNRange)
167.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds5)
    ;
168.                     else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nR
    ange5 + nRange6) / totalNRange)
169.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds6)
    ;
170.                     else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nR
    ange5 + nRange6 + nRange7) / totalNRange)
171.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds7)
    ;
172.                     else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nR
    ange5 + nRange6 + nRange7 + nRange8) / totalNRange)
173.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds8)
    ;
174.                     else
175.                         encrypted[y][x] = Conditions.randomAngleBetween(nBounds9)
    ;
176.                 }
177.             }
178.         }
179.         return encrypted;
```

```
180.     }
181.
182.     public static BigDecimal[][] encrypt(int[][] code, double theta1, double thet
     a2, double theta3, double threshold)
183.     {
184.         BigDecimal[][] encrypted = new BigDecimal[code.length][code[0].length];
185.
186.         // POSITIVE CONDITIONS
187.
188.         final BigDecimal[] pBounds1 =  Conditions.tripleThetaCondition1Bounds(  t
     heta1, theta2, theta3, threshold);
189.         final BigDecimal[] pBounds2 =  Conditions.tripleThetaCondition2Bounds(  t
     heta1, theta2, theta3, threshold);
190.         final BigDecimal[] pBounds3 =  Conditions.tripleThetaCondition3Bounds(  t
     heta1, theta2, theta3, threshold);
191.         final BigDecimal[] pBounds4 =  Conditions.tripleThetaCondition4Bounds(  t
     heta1, theta2, theta3, threshold);
192.         final BigDecimal[] pBounds5 =  Conditions.tripleThetaCondition5Bounds(  t
     heta1, theta2, theta3, threshold);
193.         final BigDecimal[] pBounds6 =  Conditions.tripleThetaCondition6Bounds(  t
     heta1, theta2, theta3, threshold);
194.         final BigDecimal[] pBounds7 =  Conditions.tripleThetaCondition7Bounds(  t
     heta1, theta2, theta3, threshold);
195.         final BigDecimal[] pBounds8 =  Conditions.tripleThetaCondition8Bounds(  t
     heta1, theta2, theta3, threshold);
196.         final BigDecimal[] pBounds9 =  Conditions.tripleThetaCondition9Bounds(  t
     heta1, theta2, theta3, threshold);
197.         final BigDecimal[] pBounds10 = Conditions.tripleThetaCondition10Bounds( t
     heta1, theta2, theta3, threshold);
198.         final BigDecimal[] pBounds11 = Conditions.tripleThetaCondition11Bounds( t
     heta1, theta2, theta3, threshold);
199.         final BigDecimal[] pBounds12 = Conditions.tripleThetaCondition12Bounds( t
     heta1, theta2, theta3, threshold);
200.         final BigDecimal[] pBounds13 = Conditions.tripleThetaCondition13Bounds( t
     heta1, theta2, theta3, threshold);
201.
202.         final double pRange1 =  Conditions.range(pBounds1);
203.         final double pRange2 =  Conditions.range(pBounds2);
204.         final double pRange3 =  Conditions.range(pBounds3);
205.         final double pRange4 =  Conditions.range(pBounds4);
206.         final double pRange5 =  Conditions.range(pBounds5);
207.         final double pRange6 =  Conditions.range(pBounds6);
208.         final double pRange7 =  Conditions.range(pBounds7);
209.         final double pRange8 =  Conditions.range(pBounds8);
210.         final double pRange9 =  Conditions.range(pBounds9);
211.         final double pRange10 = Conditions.range(pBounds10);
212.         final double pRange11 = Conditions.range(pBounds11);
213.         final double pRange12 = Conditions.range(pBounds12);
214.         final double pRange13 = Conditions.range(pBounds13);
215.         final double totalPRange = pRange1 + pRange2 + pRange3 + pRange4 + pRange
     5 + pRange6 + pRange7 + pRange8 + pRange9 + pRange10 + pRange11 + pRange12 + pRan
     ge13;
216.
217.         // NEGATIVE CONDITIONS
218.
219.         final BigDecimal[] nBounds1 =  Conditions.tripleThetaNegativeCondition1Bo
     unds(  theta1, theta2, theta3, threshold);
220.         final BigDecimal[] nBounds2 =  Conditions.tripleThetaNegativeCondition2Bo
     unds(  theta1, theta2, theta3, threshold);
221.         final BigDecimal[] nBounds3 =  Conditions.tripleThetaNegativeCondition3Bo
     unds(  theta1, theta2, theta3, threshold);
```

```
222.          final BigDecimal[] nBounds4 =  Conditions.tripleThetaNegativeCondition4Bo
      unds(  theta1, theta2, theta3, threshold);
223.          final BigDecimal[] nBounds5 =  Conditions.tripleThetaNegativeCondition5Bo
      unds(  theta1, theta2, theta3, threshold);
224.          final BigDecimal[] nBounds6 =  Conditions.tripleThetaNegativeCondition6Bo
      unds(  theta1, theta2, theta3, threshold);
225.          final BigDecimal[] nBounds7 =  Conditions.tripleThetaNegativeCondition7Bo
      unds(  theta1, theta2, theta3, threshold);
226.          final BigDecimal[] nBounds8 =  Conditions.tripleThetaNegativeCondition8Bo
      unds(  theta1, theta2, theta3, threshold);
227.          final BigDecimal[] nBounds9 =  Conditions.tripleThetaNegativeCondition9Bo
      unds(  theta1, theta2, theta3, threshold);
228.          final BigDecimal[] nBounds10 = Conditions.tripleThetaNegativeCondition10B
      ounds( theta1, theta2, theta3, threshold);
229.          final BigDecimal[] nBounds11 = Conditions.tripleThetaNegativeCondition11B
      ounds( theta1, theta2, theta3, threshold);
230.          final BigDecimal[] nBounds12 = Conditions.tripleThetaNegativeCondition12B
      ounds( theta1, theta2, theta3, threshold);
231.          final BigDecimal[] nBounds13 = Conditions.tripleThetaNegativeCondition13B
      ounds( theta1, theta2, theta3, threshold);
232.          final BigDecimal[] nBounds14 = Conditions.tripleThetaNegativeCondition14B
      ounds( theta1, theta2, theta3, threshold);
233.
234.          final double nRange1 =  Conditions.range(nBounds1);
235.          final double nRange2 =  Conditions.range(nBounds2);
236.          final double nRange3 =  Conditions.range(nBounds3);
237.          final double nRange4 =  Conditions.range(nBounds4);
238.          final double nRange5 =  Conditions.range(nBounds5);
239.          final double nRange6 =  Conditions.range(nBounds6);
240.          final double nRange7 =  Conditions.range(nBounds7);
241.          final double nRange8 =  Conditions.range(nBounds8);
242.          final double nRange9 =  Conditions.range(nBounds9);
243.          final double nRange10 = Conditions.range(nBounds10);
244.          final double nRange11 = Conditions.range(nBounds11);
245.          final double nRange12 = Conditions.range(nBounds12);
246.          final double nRange13 = Conditions.range(nBounds13);
247.          final double nRange14 = Conditions.range(nBounds14);
248.          final double totalNRange = nRange1 + nRange2 + nRange3 + nRange4 + nRange
      5 + nRange6 + nRange7 + nRange8 + nRange9 + nRange10 + nRange11 + nRange12 + nRan
      ge13 + nRange14;
249.
250.
251.          for (int y = 0; y < code.length; y ++)
252.          {
253.              for (int x = 0; x < code[y].length; x ++)
254.              {
255.                  if (code[y][x] == 1)
256.                  {
257.                      final double random = Math.random();
258.
259.                      if (random < (pRange1) / totalPRange)
260.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds1)
      ;
261.                      else if (random < (pRange1 + pRange2) / totalPRange)
262.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds2)
      ;
263.                      else if (random < (pRange1 + pRange2 + pRange3) / totalPRange
      )
264.                          encrypted[y][x] = Conditions.randomAngleBetween(pBounds3)
      ;
```

```
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds4);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds5);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds6);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6 + pRange7) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds7);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6 + pRange7 + pRange8) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds8);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6 + pRange7 + pRange8 + pRange9) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds9);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6 + pRange7 + pRange8 + pRange9 + pRange10) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds10);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6 + pRange7 + pRange8 + pRange9 + pRange10 + pRange11) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds11);
                    else if (random < (pRange1 + pRange2 + pRange3 + pRange4 + pRange5 + pRange6 + pRange7 + pRange8 + pRange9 + pRange10 + pRange11 + pRange12) / totalPRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds12);
                    else
                        encrypted[y][x] = Conditions.randomAngleBetween(pBounds13);
                }
                else
                {
                    final double random = Math.random();

                    if (random < (nRange1) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds1);
                    else if (random < (nRange1 + nRange2) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds2);
                    else if (random < (nRange1 + nRange2 + nRange3) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds3);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds4);
```

```
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds5);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds6);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds7);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7 + nRange8) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds8);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7 + nRange8 + nRange9) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds9);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7 + nRange8 + nRange9 + nRange10) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds10);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7 + nRange8 + nRange9 + nRange10 + nRange11) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds11);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7 + nRange8 + nRange9 + nRange10 + nRange11 + nRange12) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds12);
                    else if (random < (nRange1 + nRange2 + nRange3 + nRange4 + nRange5 + nRange6 + nRange7 + nRange8 + nRange9 + nRange10 + nRange11 + nRange12 + nRange13) / totalNRange)
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds13);
                    else
                        encrypted[y][x] = Conditions.randomAngleBetween(nBounds14);
                }
            }
        }
        return encrypted;
    }
}
```

```
1.  /*
2.  PROGRAM:    Conditions.java
3.  AUTHOR:     Harper O. W. Wallace
4.  DATE:       26 Dec 2017
5.
6.  DESCRIPTION:
7.  This script does calculations to determine whether calculated alignment angles
8.  will yield the correct transmittances at a given angle of LPL and transmittance
9.  threshold. It performs this conversion using a relationship similar but not
10. equivalent to the one derived empirically (we had not yet derived the empirical
11. relationship when this module was built), though the actual alignment values
12. are not used in any analysis, but rather only to demonstrate the principle.
13. */
14.
15. import java.math.BigDecimal;
16. import java.math.RoundingMode;
17.
18. public class Conditions
19. {
20.     private final static BigDecimal ZERO = new BigDecimal(0);
21.     private final static BigDecimal THIRTY = new BigDecimal(30);
22.     private final static BigDecimal SIXTY = new BigDecimal(60);
23.     private final static BigDecimal NINETY = new BigDecimal(90);
24.     private final static BigDecimal ONE_TWENTY = new BigDecimal(120);
25.     private final static BigDecimal ONE_EIGHTY = new BigDecimal(180);
26.     private final static BigDecimal TWO_SEVENTY = new BigDecimal(270);
27.     private final static BigDecimal THREE_SIXTY = new BigDecimal(360);
28.
29.     private final static BigDecimal NEG_1 = new BigDecimal(-1);
30.     private final static BigDecimal[] NIL_BOUNDS = new BigDecimal[]{NEG_1, NEG_1}
    ;
31.
32.
33. // SINGLE THETA CONDITIONS
34.
35.     // theta - 180 + 90T < alpha < theta - 90T
36.     public static BigDecimal[] singleThetaCondition1Bounds(double theta, double t
    hreshold)
37.     {
38.         return singleThetaCondition1Bounds(new BigDecimal(theta), new BigDecimal(
    threshold));
39.     }
40.     public static BigDecimal[] singleThetaCondition1Bounds(BigDecimal theta, BigD
    ecimal threshold)
41.     {
42.         BigDecimal T90 = threshold.multiply(NINETY);
43.
44.         BigDecimal lowerBound = roundLower(max(theta.subtract(ONE_EIGHTY).add(T90
    ), ZERO));
45.         BigDecimal upperBound = roundUpper(theta.subtract(T90));
                // guaranteed below 180
46.
47.         return checkBounds(lowerBound, upperBound);
48.     }
49.
50.     // theta + 90T < alpha < theta + 180 - 90T
51.     public static BigDecimal[] singleThetaCondition2Bounds(double theta, double t
    hreshold)
52.     {
53.         return singleThetaCondition2Bounds(new BigDecimal(theta), new BigDecimal(
    threshold));
```

```
54.       }
55.       public static BigDecimal[] singleThetaCondition2Bounds(BigDecimal theta, BigD
      ecimal threshold)
56.       {
57.           BigDecimal T90 = threshold.multiply(NINETY);
58.
59.           BigDecimal lowerBound = roundLower(theta.add(T90));
                          // guaranteed above 0
60.           BigDecimal upperBound = roundUpper(min(theta.add(ONE_EIGHTY).subtract(T90
      ), ONE_EIGHTY));
61.
62.           return checkBounds(lowerBound, upperBound);
63.       }
64.
65.   // SINGLE THETA NEGATIVE CONDITIONS
66.
67.       // theta - 90T < alpha < theta + 90T
68.       public static BigDecimal[] singleThetaNegativeCondition1Bounds(double theta,
      double threshold)
69.       {
70.           return singleThetaNegativeCondition1Bounds(new BigDecimal(theta), new Big
      Decimal(threshold));
71.       }
72.       public static BigDecimal[] singleThetaNegativeCondition1Bounds(BigDecimal the
      ta, BigDecimal threshold)
73.       {
74.           BigDecimal T90 = threshold.multiply(NINETY);
75.
76.           BigDecimal lowerBound = roundLower(max(theta.subtract(T90), ZERO));
77.           BigDecimal upperBound = roundUpper(min(theta.add(T90), ONE_EIGHTY));
78.
79.           return checkBounds(lowerBound, upperBound);
80.       }
81.
82.       // 0 < alpha < theta - 180 + 90T
83.       public static BigDecimal[] singleThetaNegativeCondition2Bounds(double theta,
      double threshold)
84.       {
85.           return singleThetaNegativeCondition2Bounds(new BigDecimal(theta), new Big
      Decimal(threshold));
86.       }
87.       public static BigDecimal[] singleThetaNegativeCondition2Bounds(BigDecimal the
      ta, BigDecimal threshold)
88.       {
89.           BigDecimal T90 = threshold.multiply(NINETY);
90.
91.           BigDecimal lowerBound = roundLower(ZERO);
92.           BigDecimal upperBound = roundUpper(theta.subtract(ONE_EIGHTY).add(T90));
         // guaranteed below 180
93.
94.           return checkBounds(lowerBound, upperBound);
95.       }
96.
97.       // theta + 180 - 90T < alpha < 180
98.       public static BigDecimal[] singleThetaNegativeCondition3Bounds(double theta,
      double threshold)
99.       {
100.          return singleThetaNegativeCondition3Bounds(new BigDecimal(theta), new Big
      Decimal(threshold));
101.      }
```

```
102.     public static BigDecimal[] singleThetaNegativeCondition3Bounds(BigDecimal the
     ta, BigDecimal threshold)
103.     {
104.         BigDecimal T90 = threshold.multiply(NINETY);
105.
106.         BigDecimal lowerBound = roundLower(theta.add(ONE_EIGHTY)).subtract(T90);
        // guaranteed above 0
107.         BigDecimal upperBound = roundUpper(ONE_EIGHTY);
108.
109.         return checkBounds(lowerBound, upperBound);
110.     }
111.
112. // DOUBLE THETA CONDITIONS
113.
114.     // PASSED
115.     // theta1 - 90 < alpha < min(theta2, theta_ave - 90T)
116.     public static BigDecimal[] doubleThetaCondition1Bounds(double t1, double t2,
     double threshold)
117.     {
118.         return doubleThetaCondition1Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(threshold));
119.     }
120.     public static BigDecimal[] doubleThetaCondition1Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal threshold)
121.     {
122.         BigDecimal min = min(t1, t2);
123.         BigDecimal max = max(t1, t2);
124.         BigDecimal theta_ave = ave(t1, t2);
125.         BigDecimal T90 = threshold.multiply(NINETY);
126.
127.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), ZERO));
128.         BigDecimal upperBound = roundUpper(min(min, theta_ave.subtract(T90)));
        // guaranteed below 180
129.
130.         return checkBounds(lowerBound, upperBound);
131.     }
132.
133.     // PASSED
134.     // theta_ave - 180 + 90T < alpha < theta2 - 90
135.     public static BigDecimal[] doubleThetaCondition2Bounds(double t1, double t2,
     double threshold)
136.     {
137.         return doubleThetaCondition2Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(threshold));
138.     }
139.     public static BigDecimal[] doubleThetaCondition2Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal threshold)
140.     {
141.         BigDecimal min = min(t1, t2);
142.         BigDecimal max = max(t1, t2);
143.         BigDecimal theta_ave = ave(t1, t2);
144.         BigDecimal T90 = threshold.multiply(NINETY);
145.
146.         BigDecimal lowerBound = roundLower(max(theta_ave.subtract(ONE_EIGHTY).add
     (T90), ZERO));
147.         BigDecimal upperBound = roundUpper(min.subtract(NINETY));
            // guaranteed below 180
148.
149.         return checkBounds(lowerBound, upperBound);
150.     }
151.
```

```
152.     // PASSED
153.     // max(theta2 + 90, theta_ave) < alpha < min(theta1, theta_ave + 90 - 90T)
154.     public static BigDecimal[] doubleThetaCondition3Bounds(double t1, double t2,
     double threshold)
155.     {
156.         return doubleThetaCondition3Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(threshold));
157.     }
158.     public static BigDecimal[] doubleThetaCondition3Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal threshold)
159.     {
160.         BigDecimal min = min(t1, t2);
161.         BigDecimal max = max(t1, t2);
162.         BigDecimal theta_ave = ave(t1, t2);
163.         BigDecimal T90 = threshold.multiply(NINETY);
164.
165.         BigDecimal lowerBound = roundLower(max(min.add(NINETY), theta_ave));
               // guaranteed above 0
166.         BigDecimal upperBound = roundUpper(min(max, theta_ave.add(NINETY).subtrac
     t(T90)));  // guaranteed below 180
167.
168.         return checkBounds(lowerBound, upperBound);
169.     }
170.
171.     // PASSED
172.     // max(theta2, theta_ave - 90 + 90T) < alpha < theta1 - 90
173.     public static BigDecimal[] doubleThetaCondition4Bounds(double t1, double t2,
     double threshold)
174.     {
175.         return doubleThetaCondition4Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(threshold));
176.     }
177.     public static BigDecimal[] doubleThetaCondition4Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal threshold)
178.     {
179.         BigDecimal min = min(t1, t2);
180.         BigDecimal max = max(t1, t2);
181.         BigDecimal theta_ave = ave(t1, t2);
182.         BigDecimal T90 = threshold.multiply(NINETY);
183.
184.         BigDecimal lowerBound = roundLower(max(min, theta_ave.subtract(NINETY).ad
     d(T90)));  // guaranteed above 0
185.         BigDecimal upperBound = roundUpper(max.subtract(NINETY));
               // guaranteed below 180
186.
187.         return checkBounds(lowerBound, upperBound);
188.     }
189.
190.     // PASSED
191.     // max(theta1, theta_ave + 90T) < alpha < theta2 + 90
192.     public static BigDecimal[] doubleThetaCondition5Bounds(double t1, double t2,
     double threshold)
193.     {
194.         return doubleThetaCondition5Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(threshold));
195.     }
196.     public static BigDecimal[] doubleThetaCondition5Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal threshold)
197.     {
198.         BigDecimal min = min(t1, t2);
199.         BigDecimal max = max(t1, t2);
```

```
200.          BigDecimal theta_ave = ave(t1, t2);
201.          BigDecimal T90 = threshold.multiply(NINETY);
202.
203.          BigDecimal lowerBound = roundLower(max(max, theta_ave.add(T90)));   // gu
      aranteed above 0
204.          BigDecimal upperBound = roundUpper(min(min.add(NINETY), ONE_EIGHTY));
205.
206.          return checkBounds(lowerBound, upperBound);
207.      }
208.
209.      // PASSED
210.      // theta1 + 90 < alpha < theta_ave + 180 - 90T
211.      public static BigDecimal[] doubleThetaCondition6Bounds(double t1, double t2,
      double threshold)
212.      {
213.          return doubleThetaCondition6Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(threshold));
214.      }
215.      public static BigDecimal[] doubleThetaCondition6Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal threshold)
216.      {
217.          BigDecimal min = min(t1, t2);
218.          BigDecimal max = max(t1, t2);
219.          BigDecimal theta_ave = ave(t1, t2);
220.          BigDecimal T90 = threshold.multiply(NINETY);
221.
222.          BigDecimal lowerBound = roundLower(max.add(NINETY));            // guaran
      teed above 0
223.          BigDecimal upperBound = roundUpper(min(theta_ave.add(ONE_EIGHTY).subtract
      (T90), ONE_EIGHTY));
224.
225.          return checkBounds(lowerBound, upperBound);
226.      }
227.
228.      // PASSED
229.      // IF 0 < |(theta1 - theta2)| < 180 - 180T
230.      // theta2 - 90 < alpha < theta1 - 90
231.      public static BigDecimal[] doubleThetaCondition7Bounds(double t1, double t2,
      double threshold)
232.      {
233.          return doubleThetaCondition7Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(threshold));
234.      }
235.      public static BigDecimal[] doubleThetaCondition7Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal threshold)
236.      {
237.          BigDecimal min = min(t1, t2);
238.          BigDecimal max = max(t1, t2);
239.          BigDecimal T180 = threshold.multiply(ONE_EIGHTY);
240.
241.          BigDecimal absDiff = roundLower(max.subtract(min));
242.          BigDecimal diffLimit = roundUpper(ONE_EIGHTY.subtract(T180));
243.
244.          // KEEP IF STATEMENT - ADDITIONAL CONTINGENCY
245.          if (!(absDiff.compareTo(diffLimit)< 0))
246.              return NIL_BOUNDS;
247.
248.          BigDecimal lowerBound = roundLower(max(min.subtract(NINETY), ZERO));
249.          BigDecimal upperBound = roundUpper(max.subtract(NINETY));              /
      / guaranteed below 180
250.
```

```
251.          return checkBounds(lowerBound, upperBound);
252.      }
253.
254.      // PASSED
255.      // IF 0 < |(theta1 - theta2)| < 180 - 180T
256.      // theta2 + 90 < alpha < theta1 + 90
257.      public static BigDecimal[] doubleThetaCondition8Bounds(double t1, double t2,
      double threshold)
258.      {
259.          return doubleThetaCondition8Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(threshold));
260.      }
261.      public static BigDecimal[] doubleThetaCondition8Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal threshold)
262.      {
263.          BigDecimal min = min(t1, t2);
264.          BigDecimal max = max(t1, t2);
265.          BigDecimal T180 = threshold.multiply(ONE_EIGHTY);
266.
267.          BigDecimal absDiff = roundLower(max.subtract(min));
268.          BigDecimal diffLimit = roundUpper(ONE_EIGHTY.subtract(T180));
269.
270.          // KEEP IF STATEMENT - ADDITIONAL CONTINGENCY
271.          if (!(absDiff.compareTo(diffLimit)< 0))
272.              return NIL_BOUNDS;
273.
274.          BigDecimal lowerBound = roundLower(min.add(NINETY));                    /
      / guaranteed above 0
275.          BigDecimal upperBound = roundUpper(min(max.add(NINETY), ONE_EIGHTY));
276.
277.          return checkBounds(lowerBound, upperBound);
278.      }
279.
280.      // PASSED
281.      // IF 180T < |(theta1 - theta2)| < 180
282.      // max(theta2, theta1 - 90) < alpha < min(theta1, theta2 + 90)
283.      public static BigDecimal[] doubleThetaCondition9Bounds(double t1, double t2,
      double threshold)
284.      {
285.          return doubleThetaCondition9Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(threshold));
286.      }
287.      public static BigDecimal[] doubleThetaCondition9Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal threshold)
288.      {
289.          BigDecimal min = min(t1, t2);
290.          BigDecimal max = max(t1, t2);
291.          BigDecimal T180 = threshold.multiply(ONE_EIGHTY);
292.
293.          BigDecimal absDiff = max.subtract(min);
294.
295.          // KEEP IF STATEMENT - ADDITIONAL CONTINGENCY
296.          if (!(T180.compareTo(absDiff) < 0 && absDiff.compareTo(ONE_EIGHTY) < 0))

297.              return NIL_BOUNDS;
298.
299.          BigDecimal lowerBound = roundLower(max(min, max.subtract(NINETY)));     /
      / guaranteed above 0
300.          BigDecimal upperBound = roundUpper(min(max, min.add(NINETY)));          /
      / guaranteed below 180
301.
```

```
302.          return checkBounds(lowerBound, upperBound);
303.      }
304.
305.  // DOUBLE THETA NEGATIVE CONDITIONS
306.
307.      // PASSED
308.      // max(theta1 - 90, theta_ave - 90T) < alpha < theta2
309.      public static BigDecimal[] doubleThetaNegativeCondition1Bounds(double t1, dou
      ble t2, double threshold)
310.      {
311.          return doubleThetaNegativeCondition1Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
312.      }
313.      public static BigDecimal[] doubleThetaNegativeCondition1Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
314.      {
315.          BigDecimal min = min(t1, t2);
316.          BigDecimal max = max(t1, t2);
317.          BigDecimal theta_ave = ave(t1, t2);
318.          BigDecimal T90 = threshold.multiply(NINETY);
319.
320.          BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), theta_ave.su
      btract(T90), ZERO));
321.          BigDecimal upperBound = roundUpper(min);
                              // guaranteed below 180
322.
323.          return checkBounds(lowerBound, upperBound);
324.      }
325.
326.      // PASSED
327.      // theta1 < alpha < min(theta2 + 90, theta_ave + 90T)
328.      public static BigDecimal[] doubleThetaNegativeCondition2Bounds(double t1, dou
      ble t2, double threshold)
329.      {
330.          return doubleThetaNegativeCondition2Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
331.      }
332.      public static BigDecimal[] doubleThetaNegativeCondition2Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
333.      {
334.          BigDecimal min = min(t1, t2);
335.          BigDecimal max = max(t1, t2);
336.          BigDecimal theta_ave = ave(t1, t2);
337.          BigDecimal T90 = threshold.multiply(NINETY);
338.
339.          BigDecimal lowerBound = roundLower(max);
                          // guaranteed above 0
340.          BigDecimal upperBound = roundUpper(min(min.add(NINETY), theta_ave.add(T90
      ), ONE_EIGHTY));
341.
342.          return checkBounds(lowerBound, upperBound);
343.      }
344.
345.      // PASSED
346.      // IF |(theta2 - theta1)| < 180T
347.      // max(theta1 - 90, theta2) < alpha < min(theta1, theta2 + 90)
348.      public static BigDecimal[] doubleThetaNegativeCondition3Bounds(double t1, dou
      ble t2, double threshold)
349.      {
350.          return doubleThetaNegativeCondition3Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
```

```
351.     }
352.     public static BigDecimal[] doubleThetaNegativeCondition3Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal threshold)
353.     {
354.         BigDecimal min = min(t1, t2);
355.         BigDecimal max = max(t1, t2);
356.         BigDecimal T180 = threshold.multiply(ONE_EIGHTY);
357.
358.         BigDecimal absDiff = roundLower(max.subtract(min));
359.
360.         // KEEP IF STATEMENT - ADDITIONAL CONTINGENCY
361.         if (!(absDiff.compareTo(T180) < 0))
362.             return NIL_BOUNDS;
363.
364.         BigDecimal T90 = threshold.multiply(NINETY);
365.
366.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), min));
        // guaranteed above 0
367.         BigDecimal upperBound = roundUpper(min(max, min.add(T90)));
        // guaranteed below 180
368.
369.         return checkBounds(lowerBound, upperBound);
370.     }
371.
372.     // PASSED
373.     // IF |(theta2 - theta1)| > 180 - 180T
374.     // theta2 - 90 < alpha < min(theta1 - 90, theta2)
375.     public static BigDecimal[] doubleThetaNegativeCondition4Bounds(double t1, dou
     ble t2, double threshold)
376.     {
377.         return doubleThetaNegativeCondition4Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(threshold));
378.     }
379.     public static BigDecimal[] doubleThetaNegativeCondition4Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal threshold)
380.     {
381.         BigDecimal min = min(t1, t2);
382.         BigDecimal max = max(t1, t2);
383.         BigDecimal T180 = threshold.multiply(ONE_EIGHTY);
384.
385.         BigDecimal diffLimit = roundLower(ONE_EIGHTY.subtract(T180));
386.         BigDecimal absDiff = roundUpper(max.subtract(min));
387.
388.         // KEEP IF STATEMENT - ADDITIONAL CONTINGENCY
389.         if (!(diffLimit.compareTo(absDiff) < 0))
390.             return NIL_BOUNDS;
391.
392.         BigDecimal lowerBound = roundLower(max(min.subtract(NINETY), ZERO));
393.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), min));     /
     / guaranteed below 180
394.
395.         return checkBounds(lowerBound, upperBound);
396.     }
397.
398.     // PASSED
399.     // max(theta2 + 90, theta_ave + 90 - 90T) < alpha < theta1
400.     public static BigDecimal[] doubleThetaNegativeCondition5Bounds(double t1, dou
     ble t2, double threshold)
401.     {
402.         return doubleThetaNegativeCondition5Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(threshold));
```

```
403.      }
404.      public static BigDecimal[] doubleThetaNegativeCondition5Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
405.      {
406.          BigDecimal min = min(t1, t2);
407.          BigDecimal max = max(t1, t2);
408.          BigDecimal theta_ave = ave(t1, t2);
409.          BigDecimal T90 = threshold.multiply(NINETY);
410.
411.          BigDecimal lowerBound = roundLower(max(min.add(NINETY), theta_ave.add(NIN
      ETY).subtract(T90)));          // guaranteed above 0
412.          BigDecimal upperBound = roundUpper(max);
                                     // guaranteed below 180
413.
414.          return checkBounds(lowerBound, upperBound);
415.      }
416.
417.      // PASSED
418.      // theta2 < alpha < min(theta1 - 90, theta_ave - 90 + 90T)
419.      public static BigDecimal[] doubleThetaNegativeCondition6Bounds(double t1, dou
      ble t2, double threshold)
420.      {
421.          return doubleThetaNegativeCondition6Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
422.      }
423.      public static BigDecimal[] doubleThetaNegativeCondition6Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
424.      {
425.          BigDecimal min = min(t1, t2);
426.          BigDecimal max = max(t1, t2);
427.          BigDecimal theta_ave = ave(t1, t2);
428.          BigDecimal T90 = threshold.multiply(NINETY);
429.
430.          BigDecimal lowerBound = roundLower(min);
                                     // guaranteed above 0
431.          BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), theta_ave.su
      btract(NINETY).add(T90)));     // guaranteed below 180
432.
433.          return checkBounds(lowerBound, upperBound);
434.      }
435.
436.      // PASSED
437.      // IF |(theta2 - theta1)| > 180 - 180T
438.      // max(theta1, theta2 + 90) < alpha < theta1 + 90
439.      public static BigDecimal[] doubleThetaNegativeCondition7Bounds(double t1, dou
      ble t2, double threshold)
440.      {
441.          return doubleThetaNegativeCondition7Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
442.      }
443.      public static BigDecimal[] doubleThetaNegativeCondition7Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
444.      {
445.          BigDecimal min = min(t1, t2);
446.          BigDecimal max = max(t1, t2);
447.          BigDecimal T180 = threshold.multiply(ONE_EIGHTY);
448.
449.          BigDecimal diffLimit = roundLower(ONE_EIGHTY.subtract(T180));
450.          BigDecimal absDiff = roundUpper(max.subtract(min));
451.
452.          // KEEP IF STATEMENT - ADDITIONAL CONTINGENCY
```

```
453.          if (!(diffLimit.compareTo(absDiff) < 0))
454.              return NIL_BOUNDS;
455.
456.          BigDecimal lowerBound = roundLower(max(max, min.add(NINETY)));
                                                  // guaranteed above 0
457.          BigDecimal upperBound = roundUpper(min(max.add(NINETY), ONE_EIGHTY));
458.
459.          return checkBounds(lowerBound, upperBound);
460.      }
461.
462.      // PASSED
463.      // 0 < alpha < min(theta2 - 90, theta_ave - 180 + 90T)
464.      public static BigDecimal[] doubleThetaNegativeCondition8Bounds(double t1, dou
      ble t2, double threshold)
465.      {
466.          return doubleThetaNegativeCondition8Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
467.      }
468.      public static BigDecimal[] doubleThetaNegativeCondition8Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
469.      {
470.          BigDecimal min = min(t1, t2);
471.          BigDecimal max = max(t1, t2);
472.          BigDecimal theta_ave = ave(t1, t2);
473.          BigDecimal T90 = threshold.multiply(NINETY);
474.
475.          BigDecimal lowerBound = roundLower(ZERO);
476.          BigDecimal upperBound = roundUpper(min(min.subtract(NINETY), theta_ave.su
      btract(ONE_EIGHTY).add(T90))); // guaranteed below 180
477.
478.          return checkBounds(lowerBound, upperBound);
479.      }
480.
481.      // PASSED
482.      // max(theta1 + 90, theta_ave + 180 - 90T) < alpha < 180
483.      public static BigDecimal[] doubleThetaNegativeCondition9Bounds(double t1, dou
      ble t2, double threshold)
484.      {
485.          return doubleThetaNegativeCondition9Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(threshold));
486.      }
487.      public static BigDecimal[] doubleThetaNegativeCondition9Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal threshold)
488.      {
489.          BigDecimal min = min(t1, t2);
490.          BigDecimal max = max(t1, t2);
491.          BigDecimal theta_ave = ave(t1, t2);
492.          BigDecimal T90 = threshold.multiply(NINETY);
493.
494.          BigDecimal lowerBound = roundLower(max(max.add(NINETY), theta_ave.add(ONE
      _EIGHTY).subtract(T90)));      // guaranteed above 0
495.          BigDecimal upperBound = roundUpper(ONE_EIGHTY);
496.
497.          return checkBounds(lowerBound, upperBound);
498.      }
499.
500.  // TRIPLE THETA CONDITIONS
501.
502.      // red
503.      // theta_ave - 180 + 90T < alpha < min - 90
```

```
504.     public static BigDecimal[] tripleThetaCondition1Bounds(double t1, double t2, double t3, double threshold)
505.     {
506.         return tripleThetaCondition1Bounds(new BigDecimal(t1), new BigDecimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
507.     }
508.     public static BigDecimal[] tripleThetaCondition1Bounds(BigDecimal t1, BigDecimal t2, BigDecimal t3, BigDecimal threshold)
509.     {
510.         BigDecimal min = min(t1, t2, t3);
511.         BigDecimal theta_ave = ave(t1, t2, t3);
512.         BigDecimal T90 = threshold.multiply(NINETY);
513.
514.         BigDecimal lowerBound = roundLower(max(theta_ave.subtract(ONE_EIGHTY).add(T90), ZERO));
515.         BigDecimal upperBound = roundUpper(min.subtract(NINETY));
                                                // guaranteed below 180
516.
517.         return checkBounds(lowerBound, upperBound);
518.
519.         /*
520.         double theta_ave = ave(theta1, theta2, theta3);
521.         double min = min(theta1, theta2, theta3);
522.
523.         double lowerBound = max(theta_ave - 180 + 90*threshold, 0);
524.         double upperBound = min - 90;
525.
526.         return checkBounds(lowerBound, upperBound);*/
527.     }
528.
529.     // red
530.     // max - 90 < alpha < min(min, theta_ave - 90T)
531.     public static BigDecimal[] tripleThetaCondition2Bounds(double t1, double t2, double t3, double threshold)
532.     {
533.         return tripleThetaCondition2Bounds(new BigDecimal(t1), new BigDecimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
534.     }
535.     public static BigDecimal[] tripleThetaCondition2Bounds(BigDecimal t1, BigDecimal t2, BigDecimal t3, BigDecimal threshold)
536.     {
537.         BigDecimal min = min(t1, t2, t3);
538.         BigDecimal max = max(t1, t2, t3);
539.         BigDecimal theta_ave = ave(t1, t2, t3);
540.         BigDecimal T90 = threshold.multiply(NINETY);
541.
542.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), ZERO));
543.         BigDecimal upperBound = roundUpper(min(min, theta_ave.subtract(T90)));  // guaranteed below 180
544.
545.         return checkBounds(lowerBound, upperBound);
546.     }
547.
548.     // DOES NOT WORK IN PRACTICE
549.     /*
550.     // orange
551.     // max(min - 90, max + mid - min - 360 + 270T) < alpha < min(mid - 90, min, max + mid - min - 90)
552.     public static BigDecimal[] tripleThetaCondition3Bounds(double t1, double t2, double t3, double threshold)
553.     {
```

```
554.          return tripleThetaCondition1Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(t3), new BigDecimal(threshold));
555.      }
556.      public static BigDecimal[] tripleThetaCondition3Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal t3, BigDecimal threshold)
557.      {
558.          BigDecimal min = min(t1, t2, t3);
559.          BigDecimal mid = mid(t1, t2, t3);
560.          BigDecimal max = max(t1, t2, t3);
561.          BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
562.
563.          BigDecimal lowerBound = roundLower(max(min.subtract(NINETY), max.add(mid)
      .subtract(min).add(T270), ZERO));
564.          BigDecimal upperBound = roundUpper(min(mid.subtract(NINETY), min, max.add
      (mid).subtract(min).subtract(NINETY)));    // guaranteed below 180
565.
566.          return checkBounds(lowerBound, upperBound);
567.      }
568.      */
569.
570.      // brown
571.      // max(max - 90, min, max + mid - min - 270) < alpha < min(max, min + 90, max
       + mid - min - 270T)
572.      public static BigDecimal[] tripleThetaCondition3Bounds(double t1, double t2,
      double t3, double threshold)
573.      {
574.          return tripleThetaCondition3Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(t3), new BigDecimal(threshold));
575.      }
576.      public static BigDecimal[] tripleThetaCondition3Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal t3, BigDecimal threshold)
577.      {
578.          BigDecimal min = min(t1, t2, t3);
579.          BigDecimal mid = mid(t1, t2, t3);
580.          BigDecimal max = max(t1, t2, t3);
581.          BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
582.
583.          BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), min, max.add
      (mid).subtract(min).subtract(TWO_SEVENTY)));   // guaranteed above 0
584.          BigDecimal upperBound = roundUpper(min(max, min.add(NINETY), max.add(mid)
      .subtract(min).subtract(T270)));               // guaranteed below 180
585.
586.          return checkBounds(lowerBound, upperBound);
587.      }
588.
589.      // blue
590.      // max(min, theta_ave - 120 + 90T) < alpha < min(mid - 90, theta_ave - 30)
591.      public static BigDecimal[] tripleThetaCondition4Bounds(double t1, double t2,
      double t3, double threshold)
592.      {
593.          return tripleThetaCondition4Bounds(new BigDecimal(t1), new BigDecimal(t2)
      , new BigDecimal(t3), new BigDecimal(threshold));
594.      }
595.      public static BigDecimal[] tripleThetaCondition4Bounds(BigDecimal t1, BigDeci
      mal t2, BigDecimal t3, BigDecimal threshold)
596.      {
597.          BigDecimal min = min(t1, t2, t3);
598.          BigDecimal mid = mid(t1, t2, t3);
599.          BigDecimal max = max(t1, t2, t3);
600.          BigDecimal theta_ave = ave(t1, t2, t3);
601.          BigDecimal T90 = threshold.multiply(NINETY);
```

```
602.
603.         BigDecimal lowerBound = roundLower(max(min, theta_ave.subtract(ONE_TWENTY
     ).add(T90)));      // guaranteed above 0
604.         BigDecimal upperBound = roundUpper(min(mid.subtract(NINETY), theta_ave.su
     btract(THIRTY)));  // guaranteed below 180
605.
606.         return checkBounds(lowerBound, upperBound);
607.     }
608.
609.     // blue
610.     // max(min + 90, theta_ave - 30) < alpha < min(mid, theta_ave + 60 - 90T)
611.     public static BigDecimal[] tripleThetaCondition5Bounds(double t1, double t2,
     double t3, double threshold)
612.     {
613.         return tripleThetaCondition5Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(t3), new BigDecimal(threshold));
614.     }
615.     public static BigDecimal[] tripleThetaCondition5Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal t3, BigDecimal threshold)
616.     {
617.         BigDecimal min = min(t1, t2, t3);
618.         BigDecimal mid = mid(t1, t2, t3);
619.         BigDecimal theta_ave = ave(t1, t2, t3);
620.         BigDecimal T90 = threshold.multiply(NINETY);
621.
622.         BigDecimal lowerBound = roundLower(max(min.add(NINETY), theta_ave.subtrac
     t(THIRTY)));   // guaranteed above 0
623.         BigDecimal upperBound = roundUpper(min(mid, theta_ave.add(SIXTY).subtract
     (T90)));       // guaranteed below 180
624.
625.         return checkBounds(lowerBound, upperBound);
626.     }
627.
628.     // pink
629.     // max(mid, theta_ave - 60 + 90T) < alpha < min(max - 90, theta_ave + 30)
630.     public static BigDecimal[] tripleThetaCondition6Bounds(double t1, double t2,
     double t3, double threshold)
631.     {
632.         return tripleThetaCondition6Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(t3), new BigDecimal(threshold));
633.     }
634.     public static BigDecimal[] tripleThetaCondition6Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal t3, BigDecimal threshold)
635.     {
636.         BigDecimal mid = mid(t1, t2, t3);
637.         BigDecimal max = max(t1, t2, t3);
638.         BigDecimal theta_ave = ave(t1, t2, t3);
639.         BigDecimal T90 = threshold.multiply(NINETY);
640.
641.         BigDecimal lowerBound = roundLower(max(mid, theta_ave.subtract(SIXTY).add
     (T90)));       // guaranteed above 0
642.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), theta_ave.ad
     d(THIRTY)));   // guaranteed below 180
643.
644.         return checkBounds(lowerBound, upperBound);
645.     }
646.
647.     // pink
648.     // max(mid + 90, theta_ave + 30) < alpha < min(max, theta_ave + 120 - 90T)
649.     public static BigDecimal[] tripleThetaCondition7Bounds(double t1, double t2,
     double t3, double threshold)
```

```
650.     {
651.         return tripleThetaCondition7Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(t3), new BigDecimal(threshold));
652.     }
653.     public static BigDecimal[] tripleThetaCondition7Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal t3, BigDecimal threshold)
654.     {
655.         BigDecimal mid = mid(t1, t2, t3);
656.         BigDecimal max = max(t1, t2, t3);
657.         BigDecimal theta_ave = ave(t1, t2, t3);
658.         BigDecimal T90 = threshold.multiply(NINETY);
659.
660.         BigDecimal lowerBound = roundLower(max(mid.add(NINETY), theta_ave.add(THI
     RTY)));            // guaranteed above 0
661.         BigDecimal upperBound = roundUpper(min(max, theta_ave.add(ONE_TWENTY).sub
     tract(T90)));      // guaranteed below 180
662.
663.         return checkBounds(lowerBound, upperBound);
664.     }
665.
666.     // gray
667.     // max(max - 90, mid, -max + mid + min + 270T) < alpha < min(max, min + 90, -
     max + mid + min + 270)
668.     public static BigDecimal[] tripleThetaCondition8Bounds(double t1, double t2,
     double t3, double threshold)
669.     {
670.         return tripleThetaCondition8Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(t3), new BigDecimal(threshold));
671.     }
672.     public static BigDecimal[] tripleThetaCondition8Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal t3, BigDecimal threshold)
673.     {
674.         BigDecimal min = min(t1, t2, t3);
675.         BigDecimal mid = mid(t1, t2, t3);
676.         BigDecimal max = max(t1, t2, t3);
677.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
678.
679.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), mid, min.add
     (mid).subtract(max).add(T270)));       // guaranteed above 0
680.         BigDecimal upperBound = roundUpper(min(max, min.add(NINETY), min.add(mid)
     .subtract(max).add(TWO_SEVENTY)));     // guaranteed below 180
681.
682.         return checkBounds(lowerBound, upperBound);
683.     }
684.
685.     // light green
686.     // max(mid, min + 90, max - mid + min - 90) < alpha < min(max, mid + 90, max
     - mid + min + 180 - 270T)
687.     public static BigDecimal[] tripleThetaCondition9Bounds(double t1, double t2,
     double t3, double threshold)
688.     {
689.         return tripleThetaCondition9Bounds(new BigDecimal(t1), new BigDecimal(t2)
     , new BigDecimal(t3), new BigDecimal(threshold));
690.     }
691.     public static BigDecimal[] tripleThetaCondition9Bounds(BigDecimal t1, BigDeci
     mal t2, BigDecimal t3, BigDecimal threshold)
692.     {
693.         BigDecimal min = min(t1, t2, t3);
694.         BigDecimal mid = mid(t1, t2, t3);
695.         BigDecimal max = max(t1, t2, t3);
696.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
```

```
697.
698.         BigDecimal lowerBound = roundLower(max(mid, min.add(NINETY), max.subtract
     (mid).add(min).subtract(NINETY)));                  // guaranteed above 0
699.         BigDecimal upperBound = roundUpper(min(max, mid.add(NINETY), max.subtract
     (mid).add(min).add(ONE_EIGHTY).subtract(T270)));   // guaranteed below 180
700.
701.         return checkBounds(lowerBound, upperBound);
702.     }
703.
704.     // brown
705.     // max(mid - 90, -max + mid + min - 90) < alpha < min(max - 90, min, -
     max + mid + min + 180 - 270T)
706.     public static BigDecimal[] tripleThetaCondition10Bounds(double t1, double t2,
      double t3, double threshold)
707.     {
708.         return tripleThetaCondition10Bounds(new BigDecimal(t1), new BigDecimal(t2
     ), new BigDecimal(t3), new BigDecimal(threshold));
709.     }
710.     public static BigDecimal[] tripleThetaCondition10Bounds(BigDecimal t1, BigDec
     imal t2, BigDecimal t3, BigDecimal threshold)
711.     {
712.         BigDecimal min = min(t1, t2, t3);
713.         BigDecimal mid = mid(t1, t2, t3);
714.         BigDecimal max = max(t1, t2, t3);
715.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
716.
717.         BigDecimal lowerBound = roundLower(max(mid.subtract(NINETY), min.add(mid)
     .subtract(max).subtract(NINETY), ZERO));
718.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), min, min.add
     (mid).subtract(max).add(ONE_EIGHTY).subtract(T270)));  // guaranteed below 180
719.
720.         return checkBounds(lowerBound, upperBound);
721.     }
722.
723.     // yellow
724.     // max(mid - 90, min, max - mid + min - 180 + 270T) < alpha < min(max - 90, m
     id, max - mid + min + 90)
725.     public static BigDecimal[] tripleThetaCondition11Bounds(double t1, double t2,
      double t3, double threshold)
726.     {
727.         return tripleThetaCondition11Bounds(new BigDecimal(t1), new BigDecimal(t2
     ), new BigDecimal(t3), new BigDecimal(threshold));
728.     }
729.     public static BigDecimal[] tripleThetaCondition11Bounds(BigDecimal t1, BigDec
     imal t2, BigDecimal t3, BigDecimal threshold)
730.     {
731.         BigDecimal min = min(t1, t2, t3);
732.         BigDecimal mid = mid(t1, t2, t3);
733.         BigDecimal max = max(t1, t2, t3);
734.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
735.
736.         BigDecimal lowerBound = roundLower(max(mid.subtract(NINETY), min, max.sub
     tract(mid).add(min).subtract(ONE_EIGHTY).add(T270)));  // guaranteed above 0
737.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), mid, max.sub
     tract(mid).add(min).add(NINETY)));                     // guaranteed below 180
738.
739.         return checkBounds(lowerBound, upperBound);
740.     }
741.
742.     // purple
743.     // max(max, theta_ave + 90T) < alpha < min + 90
```

```
744.     public static BigDecimal[] tripleThetaCondition12Bounds(double t1, double t2, double t3, double threshold)
745.     {
746.         return tripleThetaCondition12Bounds(new BigDecimal(t1), new BigDecimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
747.     }
748.     public static BigDecimal[] tripleThetaCondition12Bounds(BigDecimal t1, BigDecimal t2, BigDecimal t3, BigDecimal threshold)
749.     {
750.         BigDecimal min = min(t1, t2, t3);
751.         BigDecimal max = max(t1, t2, t3);
752.         BigDecimal theta_ave = ave(t1, t2, t3);
753.         BigDecimal T90 = threshold.multiply(NINETY);
754.
755.         BigDecimal lowerBound = roundLower(max(max, theta_ave.add(T90)));
        // guaranteed above 0
756.         BigDecimal upperBound = roundUpper(min(min.add(NINETY), ONE_EIGHTY));
757.
758.         return checkBounds(lowerBound, upperBound);
759.     }
760.
761.     // purple
762.     // max + 90 < alpha < theta_ave + 180 - 90T
763.     public static BigDecimal[] tripleThetaCondition13Bounds(double t1, double t2, double t3, double threshold)
764.     {
765.         return tripleThetaCondition13Bounds(new BigDecimal(t1), new BigDecimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
766.     }
767.     public static BigDecimal[] tripleThetaCondition13Bounds(BigDecimal t1, BigDecimal t2, BigDecimal t3, BigDecimal threshold)
768.     {
769.         BigDecimal max = max(t1, t2, t3);
770.         BigDecimal theta_ave = ave(t1, t2, t3);
771.         BigDecimal T90 = threshold.multiply(NINETY);
772.
773.         BigDecimal lowerBound = roundLower(max.add(NINETY));
                        // guaranteed above 0
774.         BigDecimal upperBound = roundUpper(min(theta_ave.add(ONE_EIGHTY).subtract(T90), ONE_EIGHTY));
775.
776.         return checkBounds(lowerBound, upperBound);
777.     }
778.
779. // TRIPLE THETA NEGATIVE CONDITIONS
780.
781.     // max(max - 90, theta_ave - 90T, 0) < alpha < min
782.     public static BigDecimal[] tripleThetaNegativeCondition1Bounds(double t1, double t2, double t3, double threshold)
783.     {
784.         return tripleThetaNegativeCondition1Bounds(new BigDecimal(t1), new BigDecimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
785.     }
786.     public static BigDecimal[] tripleThetaNegativeCondition1Bounds(BigDecimal t1, BigDecimal t2, BigDecimal t3, BigDecimal threshold)
787.     {
788.         BigDecimal min = min(t1, t2, t3);
789.         BigDecimal max = max(t1, t2, t3);
790.         BigDecimal theta_ave = ave(t1, t2, t3);
791.         BigDecimal T90 = threshold.multiply(NINETY);
792.
```

```
793.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), theta_ave.su
     btract(T90), ZERO));
794.         BigDecimal upperBound = roundUpper(min);
                            // guaranteed below 180
795.
796.         return checkBounds(lowerBound, upperBound);
797.     }
798.
799.     //2
800.     // max(max - 90, min, max + mid - min - 270T) < alpha < min(mid, min + 90, ma
     x + mid - min)
801.     public static BigDecimal[] tripleThetaNegativeCondition2Bounds(double t1, dou
     ble t2, double t3, double threshold)
802.     {
803.         return tripleThetaNegativeCondition2Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
804.     }
805.     public static BigDecimal[] tripleThetaNegativeCondition2Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal t3, BigDecimal threshold)
806.     {
807.         BigDecimal min = min(t1, t2, t3);
808.         BigDecimal mid = mid(t1, t2, t3);
809.         BigDecimal max = max(t1, t2, t3);
810.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
811.
812.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), min, max.add
     (mid).subtract(min).subtract(T270)));      // guaranteed above 0
813.         BigDecimal upperBound = roundUpper(min(mid, min.add(NINETY), max.add(mid)
     .subtract(min)));                          // guaranteed below 180
814.
815.         return checkBounds(lowerBound, upperBound);
816.     }
817.
818.     //3
819.     // max(max - 90, mid, -max + mid + min) < alpha < min(max, min + 90, -
     max + mid + min + 270T)
820.     public static BigDecimal[] tripleThetaNegativeCondition3Bounds(double t1, dou
     ble t2, double t3, double threshold)
821.     {
822.         return tripleThetaNegativeCondition3Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
823.     }
824.     public static BigDecimal[] tripleThetaNegativeCondition3Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal t3, BigDecimal threshold)
825.     {
826.         BigDecimal min = min(t1, t2, t3);
827.         BigDecimal mid = mid(t1, t2, t3);
828.         BigDecimal max = max(t1, t2, t3);
829.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
830.
831.         BigDecimal lowerBound = roundLower(max(max.subtract(NINETY), mid, min.add
     (mid).subtract(max)));             // guaranteed above 0
832.         BigDecimal upperBound = roundUpper(min(max, min.add(NINETY), min.add(mid)
     .subtract(max).add(T270)));        // guaranteed below 180
833.
834.         return checkBounds(lowerBound, upperBound);
835.     }
836.
837.     //4
838.     // max(min + 90, theta_ave + 60 - 90T) < alpha < min(mid, theta_ave + 60)
```

```
839.     public static BigDecimal[] tripleThetaNegativeCondition4Bounds(double t1, dou
     ble t2, double t3, double threshold)
840.     {
841.         return tripleThetaNegativeCondition4Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
842.     }
843.     public static BigDecimal[] tripleThetaNegativeCondition4Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal t3, BigDecimal threshold)
844.     {
845.         BigDecimal min = min(t1, t2, t3);
846.         BigDecimal mid = mid(t1, t2, t3);
847.         BigDecimal theta_ave = ave(t1, t2, t3);
848.         BigDecimal T90 = threshold.multiply(NINETY);
849.
850.         BigDecimal lowerBound = roundLower(max(min.add(NINETY), theta_ave.add(SIX
     TY).subtract(T90)));   // guaranteed above 0
851.         BigDecimal upperBound = roundUpper(min(mid, theta_ave.add(SIXTY)));
                         // guaranteed below 180
852.
853.         return checkBounds(lowerBound, upperBound);
854.     }
855.
856.     //5
857.     // max(mid, theta_ave - 60) < alpha < min(max - 90, theta_ave - 60 + 90T)
858.     public static BigDecimal[] tripleThetaNegativeCondition5Bounds(double t1, dou
     ble t2, double t3, double threshold)
859.     {
860.         return tripleThetaNegativeCondition5Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
861.     }
862.     public static BigDecimal[] tripleThetaNegativeCondition5Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal t3, BigDecimal threshold)
863.     {
864.         BigDecimal mid = mid(t1, t2, t3);
865.         BigDecimal max = max(t1, t2, t3);
866.         BigDecimal theta_ave = ave(t1, t2, t3);
867.         BigDecimal T90 = threshold.multiply(NINETY);
868.
869.         BigDecimal lowerBound = roundLower(max(mid, theta_ave.subtract(SIXTY)));
                           // guaranteed above 0
870.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), theta_ave.su
     btract(SIXTY).add(T90)));  // guaranteed below 180
871.
872.         return checkBounds(lowerBound, upperBound);
873.     }
874.
875.     //6 light green
876.     // max(mid, min + 90, max - mid + min + 180 - 270T) < alpha < min(max, mid +
     90, max - mid + min + 180)
877.     public static BigDecimal[] tripleThetaNegativeCondition6Bounds(double t1, dou
     ble t2, double t3, double threshold)
878.     {
879.         return tripleThetaNegativeCondition6Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
880.     }
881.     public static BigDecimal[] tripleThetaNegativeCondition6Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal t3, BigDecimal threshold)
882.     {
883.         BigDecimal min = min(t1, t2, t3);
884.         BigDecimal mid = mid(t1, t2, t3);
885.         BigDecimal max = max(t1, t2, t3);
```

```
886.          BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
887.
888.          BigDecimal lowerBound = roundLower(max(mid, min.add(NINETY), max.subtract
      (mid).add(min).add(ONE_EIGHTY).subtract(T270)));   // guaranteed above 0
889.          BigDecimal upperBound = roundUpper(min(max, mid.add(NINETY), max.subtract
      (mid).add(min).add(ONE_EIGHTY)));  // guaranteed below 180
890.
891.          return checkBounds(lowerBound, upperBound);
892.      }
893.
894.      //7
895.      // max(max, min + 90, max + mid - min - 180) < alpha < min(mid + 90, max + mi
      d - min - 180 + 270T, 180)
896.      public static BigDecimal[] tripleThetaNegativeCondition7Bounds(double t1, dou
      ble t2, double t3, double threshold)
897.      {
898.          return tripleThetaNegativeCondition7Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
899.      }
900.      public static BigDecimal[] tripleThetaNegativeCondition7Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal t3, BigDecimal threshold)
901.      {
902.          BigDecimal min = min(t1, t2, t3);
903.          BigDecimal mid = mid(t1, t2, t3);
904.          BigDecimal max = max(t1, t2, t3);
905.          BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
906.
907.          BigDecimal lowerBound = roundLower(max(max, min.add(NINETY), max.add(mid)
      .subtract(min).subtract(ONE_EIGHTY)));              // guaranteed above 0
908.          BigDecimal upperBound = roundUpper(min(mid.add(NINETY), max.add(mid).subt
      ract(min).subtract(ONE_EIGHTY).add(T270), ONE_EIGHTY));
909.
910.          return checkBounds(lowerBound, upperBound);
911.      }
912.
913.      //8
914.      // 0 < alpha < min(min - 90, theta_ave - 180 + 90T)
915.      public static BigDecimal[] tripleThetaNegativeCondition8Bounds(double t1, dou
      ble t2, double t3, double threshold)
916.      {
917.          return tripleThetaNegativeCondition8Bounds(new BigDecimal(t1), new BigDec
      imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
918.      }
919.      public static BigDecimal[] tripleThetaNegativeCondition8Bounds(BigDecimal t1,
       BigDecimal t2, BigDecimal t3, BigDecimal threshold)
920.      {
921.          BigDecimal min = min(t1, t2, t3);
922.          BigDecimal theta_ave = ave(t1, t2, t3);
923.          BigDecimal T90 = threshold.multiply(NINETY);
924.
925.          BigDecimal lowerBound = roundLower(ZERO);
926.          BigDecimal upperBound = roundUpper(min(min.subtract(NINETY), theta_ave.su
      btract(ONE_EIGHTY).add(T90)));     // guaranteed below 180
927.
928.          return checkBounds(lowerBound, upperBound);
929.      }
930.
931.      //9
932.      // max(max + 90, theta_ave + 180 - 90T) < alpha < 180
933.      public static BigDecimal[] tripleThetaNegativeCondition9Bounds(double t1, dou
      ble t2, double t3, double threshold)
```

```
934.     {
935.         return tripleThetaNegativeCondition9Bounds(new BigDecimal(t1), new BigDec
     imal(t2), new BigDecimal(t3), new BigDecimal(threshold));
936.     }
937.     public static BigDecimal[] tripleThetaNegativeCondition9Bounds(BigDecimal t1,
      BigDecimal t2, BigDecimal t3, BigDecimal threshold)
938.     {
939.         BigDecimal max = max(t1, t2, t3);
940.         BigDecimal theta_ave = ave(t1, t2, t3);
941.         BigDecimal T90 = threshold.multiply(NINETY);
942.
943.         BigDecimal lowerBound = roundLower(max(max.add(NINETY), theta_ave.add(ONE
     _EIGHTY).subtract(T90)));  // guaranteed above 0
944.         BigDecimal upperBound = roundUpper(ONE_EIGHTY);
945.
946.         return checkBounds(lowerBound, upperBound);
947.     }
948.
949.     //10
950.     // max(mid + 90, theta_ave + 120 - 90T) < alpha < min(max, theta_ave + 120)
951.     public static BigDecimal[] tripleThetaNegativeCondition10Bounds(double t1, do
     uble t2, double t3, double threshold)
952.     {
953.         return tripleThetaNegativeCondition10Bounds(new BigDecimal(t1), new BigDe
     cimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
954.     }
955.     public static BigDecimal[] tripleThetaNegativeCondition10Bounds(BigDecimal t1
     , BigDecimal t2, BigDecimal t3, BigDecimal threshold)
956.     {
957.         BigDecimal mid = mid(t1, t2, t3);
958.         BigDecimal max = max(t1, t2, t3);
959.         BigDecimal theta_ave = ave(t1, t2, t3);
960.         BigDecimal T90 = threshold.multiply(NINETY);
961.
962.         BigDecimal lowerBound = roundLower(max(mid.add(NINETY), theta_ave.add(ONE
     _TWENTY).subtract(T90)));  // guaranteed above 0
963.         BigDecimal upperBound = roundUpper(min(max, theta_ave.add(ONE_TWENTY)));
                                  // guaranteed below 180
964.
965.         return checkBounds(lowerBound, upperBound);
966.     }
967.
968.     //11 brown
969.     // max(max, mid + 90, -
     max + mid + min + 360 - 270T) < alpha < min(max + 90, -max + mid + min + 360)
970.     public static BigDecimal[] tripleThetaNegativeCondition11Bounds(double t1, do
     uble t2, double t3, double threshold)
971.     {
972.         return tripleThetaNegativeCondition11Bounds(new BigDecimal(t1), new BigDe
     cimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
973.     }
974.     public static BigDecimal[] tripleThetaNegativeCondition11Bounds(BigDecimal t1
     , BigDecimal t2, BigDecimal t3, BigDecimal threshold)
975.     {
976.         BigDecimal min = min(t1, t2, t3);
977.         BigDecimal mid = mid(t1, t2, t3);
978.         BigDecimal max = max(t1, t2, t3);
979.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
980.
981.         BigDecimal lowerBound = roundLower(max(max, mid.add(NINETY), min.add(mid)
     .subtract(max).add(THREE_SIXTY).subtract(T270)));  // guaranteed above 0
```

```
982.          BigDecimal upperBound = roundUpper(min(max.add(NINETY), min.add(mid).subt
      ract(max).add(THREE_SIXTY), ONE_EIGHTY));
983.
984.          return checkBounds(lowerBound, upperBound);
985.      }
986.
987.      //12 teal
988.      // max(min - 90, max + mid - min - 360, 0) < alpha < min(mid - 90, min, max +
       mid - min - 360 + 270T)
989.      public static BigDecimal[] tripleThetaNegativeCondition12Bounds(double t1, do
      uble t2, double t3, double threshold)
990.      {
991.          return tripleThetaNegativeCondition12Bounds(new BigDecimal(t1), new BigDe
      cimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
992.      }
993.      public static BigDecimal[] tripleThetaNegativeCondition12Bounds(BigDecimal t1
      , BigDecimal t2, BigDecimal t3, BigDecimal threshold)
994.      {
995.          BigDecimal min = min(t1, t2, t3);
996.          BigDecimal mid = mid(t1, t2, t3);
997.          BigDecimal max = max(t1, t2, t3);
998.          BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
999.
1000.         BigDecimal lowerBound = roundLower(max(min.subtract(NINETY), max.add(mid)
      .subtract(min).subtract(THREE_SIXTY), ZERO));
1001.         BigDecimal upperBound = roundUpper(min(mid.subtract(NINETY), min, max.add
      (mid).subtract(min).subtract(THREE_SIXTY).add(T270))); // guaranteed below 180
1002.
1003.         return checkBounds(lowerBound, upperBound);
1004.     }
1005.
1006.     //13
1007.     // max(min, theta_ave - 120) < alpha < min(mid - 90, theta_ave - 120 + 90T)
1008.     public static BigDecimal[] tripleThetaNegativeCondition13Bounds(double t1, do
      uble t2, double t3, double threshold)
1009.     {
1010.         return tripleThetaNegativeCondition13Bounds(new BigDecimal(t1), new BigDe
      cimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
1011.     }
1012.     public static BigDecimal[] tripleThetaNegativeCondition13Bounds(BigDecimal t1
      , BigDecimal t2, BigDecimal t3, BigDecimal threshold)
1013.     {
1014.         BigDecimal min = min(t1, t2, t3);
1015.         BigDecimal mid = mid(t1, t2, t3);
1016.         BigDecimal max = max(t1, t2, t3);
1017.         BigDecimal theta_ave = ave(t1, t2, t3);
1018.         BigDecimal T90 = threshold.multiply(NINETY);
1019.
1020.         BigDecimal lowerBound = roundLower(max(min, theta_ave.subtract(ONE_TWENTY
      )));                              // guaranteed above 0
1021.         BigDecimal upperBound = roundUpper(min(mid.subtract(NINETY), theta_ave.su
      btract(ONE_TWENTY).add(T90))); // guaranteed below 180
1022.
1023.         return checkBounds(lowerBound, upperBound);
1024.     }
1025.
1026.     // DOES NOT WORK IN PRACTICE
1027.     /*
1028.     //15 gray
1029.     // max(mid - 90, min, max - mid + min - 180 + 270T) < alpha < min(max - 90, m
      id, max - mid + min - 180)
```

```
1030.     public static BigDecimal[] tripleThetaNegativeCondition15Bounds(double t1, do
      uble t2, double t3, double threshold)
1031.     {
1032.         return tripleThetaNegativeCondition15Bounds(new BigDecimal(t1), new BigDe
      cimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
1033.     }
1034.     public static BigDecimal[] tripleThetaNegativeCondition15Bounds(BigDecimal t1
      , BigDecimal t2, BigDecimal t3, BigDecimal threshold)
1035.     {
1036.         BigDecimal min = min(t1, t2, t3);
1037.         BigDecimal mid = mid(t1, t2, t3);
1038.         BigDecimal max = max(t1, t2, t3);
1039.         BigDecimal theta_ave = ave(t1, t2, t3);
1040.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
1041.
1042.         BigDecimal lowerBound = roundLower(max(mid.subtract(NINETY), min, max.sub
      tract(mid).add(min).subtract(ONE_EIGHTY).add(T270)));  // guaranteed above 0
1043.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), mid, max.sub
      tract(mid).add(min).subtract(ONE_EIGHTY)));            // guaranteed below 180
1044.
1045.         return checkBounds(lowerBound, upperBound);
1046.     }
1047.     */
1048.
1049.     //14
1050.     // max(mid - 90, -
      max + mid + min + 180 - 270T, 0) < alpha < min(max - 90, min, -
      max + mid + min + 180)
1051.     public static BigDecimal[] tripleThetaNegativeCondition14Bounds(double t1, do
      uble t2, double t3, double threshold)
1052.     {
1053.         return tripleThetaNegativeCondition14Bounds(new BigDecimal(t1), new BigDe
      cimal(t2), new BigDecimal(t3), new BigDecimal(threshold));
1054.     }
1055.     public static BigDecimal[] tripleThetaNegativeCondition14Bounds(BigDecimal t1
      , BigDecimal t2, BigDecimal t3, BigDecimal threshold)
1056.     {
1057.         BigDecimal min = min(t1, t2, t3);
1058.         BigDecimal mid = mid(t1, t2, t3);
1059.         BigDecimal max = max(t1, t2, t3);
1060.         BigDecimal theta_ave = ave(t1, t2, t3);
1061.         BigDecimal T270 = threshold.multiply(TWO_SEVENTY);
1062.
1063.         BigDecimal lowerBound = roundLower(max(mid.subtract(NINETY), min.add(mid)
      .subtract(max).add(ONE_EIGHTY).subtract(T270), ZERO));
1064.         BigDecimal upperBound = roundUpper(min(max.subtract(NINETY), min, min.add
      (mid).subtract(max).add(ONE_EIGHTY)));                 // guaranteed below 180
1065.
1066.         return checkBounds(lowerBound, upperBound);
1067.     }
1068.
1069.
1070. // HELPERS
1071.
1072.     // ensures a < b
1073.     public static BigDecimal[] checkBounds(BigDecimal a, BigDecimal b)
1074.     {
1075.         if (a.compareTo(b) < 0)
1076.             return new BigDecimal[]{a, b};
1077.         return NIL_BOUNDS;
1078.     }
```

```
1079.
1080.     public static boolean isNilBounds(BigDecimal[] bounds) {
1081.         return isNilAngle(bounds[0]) && isNilAngle(bounds[1]);
1082.     }
1083.     public static boolean isNilAngle(BigDecimal angle) {
1084.         return angle.compareTo(NEG_1) == 0;
1085.     }
1086.
1087.
1088. // CALCULATION
1089.
1090.     public static BigDecimal roundLower(BigDecimal l) {
1091.         return l.setScale(8, RoundingMode.HALF_UP);
1092.     }
1093.     public static BigDecimal roundUpper(BigDecimal u) {
1094.         return u.setScale(8, RoundingMode.HALF_DOWN);
1095.     }
1096.
1097.     public static BigDecimal ave(BigDecimal a, BigDecimal b) {
1098.         return (a.add(b)).divide(new BigDecimal(2));
1099.     }
1100.     public static BigDecimal ave(BigDecimal a, BigDecimal b, BigDecimal c) {
1101.         return (a.add(b).add(c)).divide(new BigDecimal(3), 16, RoundingMode.HALF_
      DOWN);
1102.     }
1103.     public static BigDecimal min(BigDecimal a, BigDecimal b) {
1104.         return a.min(b);
1105.     }
1106.     public static BigDecimal min(BigDecimal a, BigDecimal b, BigDecimal c) {
1107.         return min(a, min(b, c));
1108.     }
1109.     public static BigDecimal min(BigDecimal a, BigDecimal b, BigDecimal c, BigDec
      imal d) {
1110.         return min(a, min(b, c, d));
1111.     }
1112.     public static BigDecimal max(BigDecimal a, BigDecimal b) {
1113.         return a.max(b);
1114.     }
1115.     public static BigDecimal max(BigDecimal a, BigDecimal b, BigDecimal c) {
1116.         return max(a, max(b, c));
1117.     }
1118.     public static BigDecimal mid(BigDecimal a, BigDecimal b, BigDecimal c)
1119.     {
1120.         BigDecimal min = min(a, b, c);
1121.         BigDecimal max = max(a, b, c);
1122.
1123.         if ((min.equals(a) && max.equals(b)) || (min.equals(b) && max.equals(a)))

1124.             return c;
1125.         else if ((min.equals(a) && max.equals(c)) || (min.equals(c) && max.equals
      (a)))
1126.             return b;
1127.         return a;
1128.     }
1129.
1130. // HELPERS
1131.
1132.     public static BigDecimal random() {
1133.         return new BigDecimal(Math.random());
1134.     }
1135.     public static BigDecimal random180() {
```

```
1136.         return (new BigDecimal(Math.random())).multiply(ONE_EIGHTY);
1137.     }
1138.
1139.     // IF a > b or bounds are nil, returns -1
1140.     public static BigDecimal randomAngleBetween(BigDecimal a, BigDecimal b)
1141.     {
1142.         if (a.compareTo(b) < 0)
1143.             return a.add(random().multiply(b.subtract(a)));
1144.         return NEG_1;
1145.     }
1146.     public static BigDecimal randomAngleBetween(BigDecimal[] bounds) {
1147.         return randomAngleBetween(bounds[0], bounds[1]);
1148.     }
1149.
1150.     // ASSUMING a < b
1151.     public static double range(BigDecimal[] bounds)
1152.     {
1153.         if (!Conditions.isNilBounds(bounds))
1154.             return (bounds[1].subtract(bounds[0])).doubleValue();
1155.         return 0;
1156.     }
1157.
1158. // SIGNALING
1159.
1160.     public static BigDecimal signal(BigDecimal angle, BigDecimal theta)
1161.     {
1162.         return NINETY.subtract(  (NINETY.subtract(  (theta.subtract(angle)).abs()
        )).abs()  ).divide(NINETY, 8, RoundingMode.HALF_DOWN);
1163.     }
1164.     public static BigDecimal signal(BigDecimal a, BigDecimal t1, BigDecimal t2)
1165.     {
1166.         return ave(signal(a, t1), signal(a, t2));
1167.     }
1168.     public static BigDecimal signal(BigDecimal a, BigDecimal t1, BigDecimal t2, B
      igDecimal t3)
1169.     {
1170.         return ave(signal(a, t1), signal(a, t2), signal(a, t3));
1171.     }
1172.
1173.     public static boolean willSignal(BigDecimal signal, BigDecimal threshold) {
1174.         return signal.compareTo(threshold) > 0;
1175.     }
1176.     public static boolean willSignal(BigDecimal angle, BigDecimal theta, BigDecim
      al threshold) {
1177.         return signal(angle, theta).compareTo(threshold) > 0;
1178.     }
1179.     public static boolean willSignal(BigDecimal angle, BigDecimal theta1, BigDeci
      mal theta2, BigDecimal threshold) {
1180.         return signal(angle, theta1, theta2).compareTo(threshold) > 0;
1181.     }
1182.     public static boolean willSignal(BigDecimal angle, BigDecimal theta1, BigDeci
      mal theta2, BigDecimal theta3, BigDecimal threshold) {
1183.         return signal(angle, theta1, theta2, theta3).compareTo(threshold) > 0;
1184.     }
1185.
1186. // REFERENCE
1187.
1188.     public static BigDecimal[] singleThetaConditionBounds(int n, BigDecimal theta
      , BigDecimal threshold)
1189.     {
1190.         switch (n)
```

```
1191.         {
1192.             case 1:     return singleThetaCondition1Bounds(theta, threshold);
1193.             case 2:     return singleThetaCondition2Bounds(theta, threshold);
1194.             default:    return NIL_BOUNDS;
1195.         }
1196.     }
1197.     public static BigDecimal[] singleThetaNegativeConditionBounds(int n, BigDecim
      al theta, BigDecimal threshold)
1198.     {
1199.         switch (n)
1200.         {
1201.             case 1:     return singleThetaNegativeCondition1Bounds(theta, thresho
      ld);
1202.             case 2:     return singleThetaNegativeCondition2Bounds(theta, thresho
      ld);
1203.             case 3:     return singleThetaNegativeCondition3Bounds(theta, thresho
      ld);
1204.             default:    return NIL_BOUNDS;
1205.         }
1206.     }
1207.     public static BigDecimal[] doubleThetaConditionBounds(int n, BigDecimal theta
      1, BigDecimal theta2, BigDecimal threshold)
1208.     {
1209.         switch (n)
1210.         {
1211.             case 1:     return doubleThetaCondition1Bounds(theta1, theta2, thresh
      old);
1212.             case 2:     return doubleThetaCondition2Bounds(theta1, theta2, thresh
      old);
1213.             case 3:     return doubleThetaCondition3Bounds(theta1, theta2, thresh
      old);
1214.             case 4:     return doubleThetaCondition4Bounds(theta1, theta2, thresh
      old);
1215.             case 5:     return doubleThetaCondition5Bounds(theta1, theta2, thresh
      old);
1216.             case 6:     return doubleThetaCondition6Bounds(theta1, theta2, thresh
      old);
1217.             case 7:     return doubleThetaCondition7Bounds(theta1, theta2, thresh
      old);
1218.             case 8:     return doubleThetaCondition8Bounds(theta1, theta2, thresh
      old);
1219.             case 9:     return doubleThetaCondition9Bounds(theta1, theta2, thresh
      old);
1220.             default:    return NIL_BOUNDS;
1221.         }
1222.     }
1223.     public static BigDecimal[] doubleThetaNegativeConditionBounds(int n, BigDecim
      al theta1, BigDecimal theta2, BigDecimal threshold)
1224.     {
1225.         switch (n)
1226.         {
1227.             case 1:     return doubleThetaNegativeCondition1Bounds(theta1, theta2
      , threshold);
1228.             case 2:     return doubleThetaNegativeCondition2Bounds(theta1, theta2
      , threshold);
1229.             case 3:     return doubleThetaNegativeCondition3Bounds(theta1, theta2
      , threshold);
1230.             case 4:     return doubleThetaNegativeCondition4Bounds(theta1, theta2
      , threshold);
1231.             case 5:     return doubleThetaNegativeCondition5Bounds(theta1, theta2
      , threshold);
```

```
1232.             case 6:     return doubleThetaNegativeCondition6Bounds(theta1, theta2
      , threshold);
1233.             case 7:     return doubleThetaNegativeCondition7Bounds(theta1, theta2
      , threshold);
1234.             case 8:     return doubleThetaNegativeCondition8Bounds(theta1, theta2
      , threshold);
1235.             case 9:     return doubleThetaNegativeCondition9Bounds(theta1, theta2
      , threshold);
1236.             default:    return NIL_BOUNDS;
1237.         }
1238.     }
1239.     public static BigDecimal[] tripleThetaConditionBounds(int n, BigDecimal theta
      1, BigDecimal theta2, BigDecimal theta3, BigDecimal threshold)
1240.     {
1241.         switch (n)
1242.         {
1243.             case 1:     return tripleThetaCondition1Bounds(  theta1, theta2, thet
      a3, threshold);
1244.             case 2:     return tripleThetaCondition2Bounds(  theta1, theta2, thet
      a3, threshold);
1245.             case 3:     return tripleThetaCondition3Bounds(  theta1, theta2, thet
      a3, threshold);
1246.             case 4:     return tripleThetaCondition4Bounds(  theta1, theta2, thet
      a3, threshold);
1247.             case 5:     return tripleThetaCondition5Bounds(  theta1, theta2, thet
      a3, threshold);
1248.             case 6:     return tripleThetaCondition6Bounds(  theta1, theta2, thet
      a3, threshold);
1249.             case 7:     return tripleThetaCondition7Bounds(  theta1, theta2, thet
      a3, threshold);
1250.             case 8:     return tripleThetaCondition8Bounds(  theta1, theta2, thet
      a3, threshold);
1251.             case 9:     return tripleThetaCondition9Bounds(  theta1, theta2, thet
      a3, threshold);
1252.             case 10:    return tripleThetaCondition10Bounds( theta1, theta2, thet
      a3, threshold);
1253.             case 11:    return tripleThetaCondition11Bounds( theta1, theta2, thet
      a3, threshold);
1254.             case 12:    return tripleThetaCondition12Bounds( theta1, theta2, thet
      a3, threshold);
1255.             case 13:    return tripleThetaCondition13Bounds( theta1, theta2, thet
      a3, threshold);
1256.             default:    return NIL_BOUNDS;
1257.         }
1258.     }
1259.     public static BigDecimal[] tripleThetaNegativeConditionBounds(int n, BigDecim
      al theta1, BigDecimal theta2, BigDecimal theta3, BigDecimal threshold)
1260.     {
1261.         switch (n)
1262.         {
1263.             case 1:     return tripleThetaNegativeCondition1Bounds(  theta1, thet
      a2, theta3, threshold);
1264.             case 2:     return tripleThetaNegativeCondition2Bounds(  theta1, thet
      a2, theta3, threshold);
1265.             case 3:     return tripleThetaNegativeCondition3Bounds(  theta1, thet
      a2, theta3, threshold);
1266.             case 4:     return tripleThetaNegativeCondition4Bounds(  theta1, thet
      a2, theta3, threshold);
1267.             case 5:     return tripleThetaNegativeCondition5Bounds(  theta1, thet
      a2, theta3, threshold);
```

```
1268.             case 6:     return tripleThetaNegativeCondition6Bounds(  theta1, thet
      a2, theta3, threshold);
1269.             case 7:     return tripleThetaNegativeCondition7Bounds(  theta1, thet
      a2, theta3, threshold);
1270.             case 8:     return tripleThetaNegativeCondition8Bounds(  theta1, thet
      a2, theta3, threshold);
1271.             case 9:     return tripleThetaNegativeCondition9Bounds(  theta1, thet
      a2, theta3, threshold);
1272.             case 10:    return tripleThetaNegativeCondition10Bounds( theta1, thet
      a2, theta3, threshold);
1273.             case 11:    return tripleThetaNegativeCondition11Bounds( theta1, thet
      a2, theta3, threshold);
1274.             case 12:    return tripleThetaNegativeCondition12Bounds( theta1, thet
      a2, theta3, threshold);
1275.             case 13:    return tripleThetaNegativeCondition13Bounds( theta1, thet
      a2, theta3, threshold);
1276.             case 14:    return tripleThetaNegativeCondition14Bounds( theta1, thet
      a2, theta3, threshold);
1277.             default:    return NIL_BOUNDS;
1278.         }
1279.     }
1280. }
```

```java
/*
PROGRAM:    Tester.java
AUTHOR:     Harper O. W. Wallace
DATE:       26 Dec 2017

DESCRIPTION:
This script tests the conversions used to relate binary "0" or "1" at a
given transmittance threshold to angles. As described in
TransmittanceEncrypt.java, the relationships used for conversion and tested
here are not empirically true, but are only used for demonstration purposes;
indeed, they still capture the principle of the scheme.
*/


import java.math.BigDecimal;
import java.math.RoundingMode;

public class Tester
{
    private final static int NUM_EXCLUSIVE_TRIALS = 10000;

    private final static int NUM_SINGLE_THETA_CONDITIONS = 2;
    private final static int NUM_SINGLE_THETA_NEGATIVE_CONDITIONS = 3;
    private final static int NUM_DOUBLE_THETA_CONDITIONS = 9;
    private final static int NUM_DOUBLE_THETA_NEGATIVE_CONDITIONS = 9;
    private final static int NUM_TRIPLE_THETA_CONDITIONS = 13;
    private final static int NUM_TRIPLE_THETA_NEGATIVE_CONDITIONS = 14;

// EXCLUSIVE TESTS

    public static void exclusiveTestAllConditions()
    {
        System.out.println();
        System.out.println("SINGLE THETA");
        exclusiveTestSingleThetaConditions();
        System.out.println();

        System.out.println("SINGLE THETA - NEGATIVE");
        exclusiveTestSingleThetaNegativeConditions();
        System.out.println();

        System.out.println("DOUBLE THETA");
        exclusiveTestDoubleThetaConditions();
        System.out.println();

        System.out.println("DOUBLE THETA - NEGATIVE");
        exclusiveTestDoubleThetaNegativeConditions();
        System.out.println();

        System.out.println("TRIPLE THETA");
        exclusiveTestTripleThetaConditions();
        System.out.println();

        System.out.println("TRIPLE THETA - NEGATIVE");
        exclusiveTestTripleThetaNegativeConditions();
        System.out.println();
    }

    // PASSED
    // MAKE SURE CONDITIONS EXCLUDE ALL INCORRECT ANGLES (ALL RESULTS OF CONDITIO
NS WILL SIGNAL)
```

```
61.       public static void exclusiveTestSingleThetaConditions()
62.       {
63.           BigDecimal theta;
64.           BigDecimal threshold;
65.           BigDecimal[] bounds;
66.           BigDecimal angle;
67.
68.           int signaled = 0;
69.
70.           for (int condition = 1; condition <= NUM_SINGLE_THETA_CONDITIONS; conditi
      on ++)
71.           {
72.               signaled = 0;
73.               for (int i = 0; i < NUM_EXCLUSIVE_TRIALS; i ++)
74.               {
75.                   do
76.                   {
77.                       theta = Conditions.random180();
78.                       threshold = Conditions.random();
79.
80.                       bounds = Conditions.singleThetaConditionBounds(condition, the
      ta, threshold);
81.                       angle = Conditions.randomAngleBetween(bounds);
82.
83.                   } while (Conditions.isNilAngle(angle));
84.
85.                   signaled += Conditions.willSignal(angle, theta, threshold) ? 1 :
      0;
86.
87.                   if (!Conditions.willSignal(angle, theta, threshold))
88.                       printError(angle, threshold, bounds, Conditions.signal(angle,
       theta), theta);
89.               }
90.               System.out.println(condition + " \t-
      > " + ((double)signaled / 100) + "%");
91.           }
92.       }
93.
94.       // PASSED
95.       // MAKE SURE CONDITIONS EXCLUDE ALL INCORRECT ANGLES (ALL RESULTS OF CONDITIO
      NS WILL SIGNAL)
96.       public static void exclusiveTestSingleThetaNegativeConditions()
97.       {
98.           BigDecimal theta;
99.           BigDecimal threshold;
100.          BigDecimal[] bounds;
101.          BigDecimal angle;
102.
103.          int notSignaled = 0;
104.
105.          for (int condition = 1; condition <= NUM_SINGLE_THETA_NEGATIVE_CONDITIONS
      ; condition ++)
106.          {
107.              notSignaled = 0;
108.              for (int i = 0; i < NUM_EXCLUSIVE_TRIALS; i ++)
109.              {
110.                  do
111.                  {
112.                      theta = Conditions.random180();
113.                      threshold = Conditions.random();
114.
```

```
                    bounds = Conditions.singleThetaNegativeConditionBounds(condit
ion, theta, threshold);
                    angle = Conditions.randomAngleBetween(bounds);

                } while (Conditions.isNilAngle(angle));

                notSignaled += Conditions.willSignal(angle, theta, threshold) ? 0
 : 1;

                if (Conditions.willSignal(angle, theta, threshold))
                    printError(angle, threshold, bounds, Conditions.signal(angle,
 theta), theta);
            }
            System.out.println(condition + " \t-
> " + ((double)notSignaled / 100) + "%");
        }
    }

    // PASSED
    // MAKE SURE CONDITIONS EXCLUDE ALL INCORRECT ANGLES (ALL RESULTS OF CONDITIO
NS WILL SIGNAL)
    public static void exclusiveTestDoubleThetaConditions()
    {
        BigDecimal theta1;
        BigDecimal theta2;
        BigDecimal threshold;
        BigDecimal[] bounds;
        BigDecimal angle;

        int signaled = 0;

        for (int condition = 1; condition <= NUM_DOUBLE_THETA_CONDITIONS; conditi
on ++)
        {
            signaled = 0;
            for (int i = 0; i < NUM_EXCLUSIVE_TRIALS; i ++)
            {
                do
                {
                    theta1 = Conditions.random180();
                    theta2 = Conditions.random180();
                    threshold = Conditions.random();

                    bounds = Conditions.doubleThetaConditionBounds(condition, the
ta1, theta2, threshold);
                    angle = Conditions.randomAngleBetween(bounds);

                } while (Conditions.isNilAngle(angle));

                signaled += Conditions.willSignal(angle, theta1, theta2, threshol
d) ? 1 : 0;

                if (!Conditions.willSignal(angle, theta1, theta2, threshold))
                    printError(angle, threshold, bounds, Conditions.signal(angle,
 theta1, theta2), theta1, theta2);
            }
            System.out.println(condition + " \t-
> " + ((double)signaled / 100) + "%");
        }
    }

```

```
166.     // PASSED
167.     // MAKE SURE CONDITIONS EXCLUDE ALL INCORRECT ANGLES (ALL RESULTS OF CONDITIO
     NS WILL SIGNAL)
168.     public static void exclusiveTestDoubleThetaNegativeConditions()
169.     {
170.         BigDecimal theta1;
171.         BigDecimal theta2;
172.         BigDecimal threshold;
173.         BigDecimal[] bounds;
174.         BigDecimal angle;
175.
176.         int notSignaled = 0;
177.
178.         // NOTE: CHANGED CONDITION LIMITS
179.         for (int condition = 1; condition <= NUM_DOUBLE_THETA_NEGATIVE_CONDITIONS
     ; condition ++)
180.         {
181.             notSignaled = 0;
182.             for (int i = 0; i < NUM_EXCLUSIVE_TRIALS; i ++)
183.             {
184.                 do
185.                 {
186.                     theta1 = Conditions.random180();
187.                     theta2 = Conditions.random180();
188.                     threshold = Conditions.random();
189.
190.                     bounds = Conditions.doubleThetaNegativeConditionBounds(condit
     ion, theta1, theta2, threshold);
191.                     angle = Conditions.randomAngleBetween(bounds);
192.
193.                 } while (Conditions.isNilAngle(angle));
194.
195.                 notSignaled += Conditions.willSignal(angle, theta1, theta2, thres
     hold) ? 0 : 1;
196.
197.                 if (Conditions.willSignal(angle, theta1, theta2, threshold))
198.                     printError(angle, threshold, bounds, Conditions.signal(angle,
      theta1, theta2), theta1, theta2);
199.             }
200.             System.out.println(condition + " \t-
     > " + ((double)notSignaled / 100) + "%");
201.         }
202.     }
203.
204.     // PASSED
205.     // MAKE SURE CONDITIONS EXCLUDE ALL INCORRECT ANGLES (ALL RESULTS OF CONDITIO
     NS WILL SIGNAL)
206.     public static void exclusiveTestTripleThetaConditions()
207.     {
208.         BigDecimal theta1;
209.         BigDecimal theta2;
210.         BigDecimal theta3;
211.         BigDecimal threshold;
212.         BigDecimal[] bounds;
213.         BigDecimal angle;
214.
215.         int signaled = 0;
216.
217.         // NOTE: CHANGED CONDITION LIMITS
218.         for (int condition = 1; condition <= NUM_TRIPLE_THETA_CONDITIONS; conditi
     on ++)
```

```
219.         {
220.             signaled = 0;
221.             for (int i = 0; i < NUM_EXCLUSIVE_TRIALS; i ++)
222.             {
223.                 do
224.                 {
225.                     theta1 = Conditions.random180();
226.                     theta2 = Conditions.random180();
227.                     theta3 = Conditions.random180();
228.                     threshold = Conditions.random();
229.
230.                     bounds = Conditions.tripleThetaConditionBounds(condition, the
    ta1, theta2, theta3, threshold);
231.                     angle = Conditions.randomAngleBetween(bounds);
232.
233.                 } while (Conditions.isNilAngle(angle));
234.
235.                 signaled += Conditions.willSignal(angle, theta1, theta2, theta3,
    threshold) ? 1 : 0;
236.
237.                 if (!Conditions.willSignal(angle, theta1, theta2, theta3, thresho
    ld))
238.                     printError(angle, threshold, bounds, Conditions.signal(angle,
     theta1, theta2, theta3), theta1, theta2, theta3);
239.             }
240.             System.out.println(condition + " \t-
    > " + ((double)signaled / 100) + "%");
241.         }
242.     }
243.
244.     // PASSED
245.     // MAKE SURE CONDITIONS EXCLUDE ALL INCORRECT ANGLES (ALL RESULTS OF CONDITIO
    NS WILL SIGNAL)
246.     public static void exclusiveTestTripleThetaNegativeConditions()
247.     {
248.         BigDecimal theta1;
249.         BigDecimal theta2;
250.         BigDecimal theta3;
251.         BigDecimal threshold;
252.         BigDecimal[] bounds;
253.         BigDecimal angle;
254.
255.         int notSignaled = 0;
256.
257.         // NOTE: CHANGED CONDITION LIMITS
258.         for (int condition = 1; condition <= NUM_TRIPLE_THETA_NEGATIVE_CONDITIONS
    ; condition ++)
259.         {
260.             notSignaled = 0;
261.             for (int i = 0; i < NUM_EXCLUSIVE_TRIALS; i ++)
262.             {
263.                 do
264.                 {
265.                     theta1 = Conditions.random180();
266.                     theta2 = Conditions.random180();
267.                     theta3 = Conditions.random180();
268.                     threshold = Conditions.random();
269.
270.                     bounds = Conditions.tripleThetaNegativeConditionBounds(condit
    ion, theta1, theta2, theta3, threshold);
271.                     angle = Conditions.randomAngleBetween(bounds);
```

```
272.
273.                 } while (Conditions.isNilAngle(angle));
274.
275.                 notSignaled += Conditions.willSignal(angle, theta1, theta2, theta
    3, threshold) ? 0 : 1;
276.
277.                 if (Conditions.willSignal(angle, theta1, theta2, theta3, threshol
    d))
278.                     printError(angle, threshold, bounds, Conditions.signal(angle,
     theta1, theta2, theta3), theta1, theta2, theta3);
279.             }
280.             System.out.println(condition + " \t-
    > " + ((double)notSignaled / 100) + "%");
281.         }
282.     }
283.
284.     public static void printError(BigDecimal angle, BigDecimal threshold, BigDeci
    mal[] bounds, BigDecimal signal, BigDecimal...thetas)
285.     {
286.         String thetaString = "";
287.         for (int i = 0; i < thetas.length; i ++)
288.             thetaString += round(thetas[i]) + (i == thetas.length - 1 ? "" : ", "
    );
289.
290.         System.out.println("\tFAILED");
291.         System.out.println("\t" + round(angle) + ", " + thetaString + " (" + roun
    d(threshold.multiply(new BigDecimal(100))) + "%)");
292.         System.out.println("\t" + round(bounds[0]) + " < " + round(angle) + " < "
     + round(bounds[1]));
293.         System.out.println("\t" + round(signal.multiply(new BigDecimal(100))) + "
    %");
294.     }
295.
296. // INCLUSIVE TESTS
297.
298.     /*
299.     // MAKE SURE CONDITIONS INCLUDE ALL CORRECT ANGLES (ALL POTENTIAL SIGNAL ANGL
    ES ARE COVERED BY CONDITIONS)
300.     public static void inclusiveTestSingleThetaConditions()
301.     {
302.         for (int theta = 0; theta < 180; theta ++)
303.         {
304.             for (int T = 1; T < 10; T ++)
305.             {
306.                 double threshold = (double)(T) / 10;
307.
308.                 for (int angle = 0; angle < 180; angle ++)
309.                 {
310.                     boolean willSignal = Encrypter.willSignal(angle, theta, thres
    hold);
311.
312.                     boolean angleExistsInConditions = false;
313.                     for (int condition = 1; condition <= NUM_SINGLE_THETA_CONDITI
    ONS; condition ++)
314.                     {
315.                         if (angleExistsInConditions)
316.                             break;
317.
318.                         double[] bounds = Conditions.singleThetaConditionBounds(c
    ondition, theta, threshold);
```

```
                        angleExistsInConditions = ((bounds[0] < angle && angle < bounds[1]) || (bounds[0] + 180 < angle && angle < bounds[1] + 180));

                        if (!willSignal && angleExistsInConditions)
                        {
                            System.out.println("C" + condition + " : \t" + bounds[0] + " < " + angle + " < " + bounds[1]);
                        }
                    }

                    String angleText = angle + " :\t" + theta + " (" + threshold + "%)";
                    if (willSignal && !angleExistsInConditions)
                        System.out.println(angleText + "  \t-> CORRECT ANGLE DOES NOT EXIST IN CONDITIONS");
                    else if (!willSignal && angleExistsInConditions)
                        System.out.println(angleText + "  \t-> INCORRECT ANGLE EXISTS IN CONDITIONS");
                }
            }
        }

    // MAKE SURE CONDITIONS INCLUDE ALL CORRECT ANGLES (ALL POTENTIAL SIGNAL ANGLES ARE COVERED BY CONDITIONS)
    public static void inclusiveTestDoubleThetaConditions()
    {
        for (int t2 = 0; t2 < 180; t2 ++)
        {
            for (int t1 = t2 + 1; t1 < 180; t1 ++)
            {
                for (int T = 1; T < 10; T ++)
                {
                    BigDecimal theta1 = new BigDecimal(t1);
                    BigDecimal theta2 = new BigDecimal(t2);
                    BigDecimal threshold = new BigDecimal((double)(T) / 10);

                    for (int angle = 0; angle < 180; angle ++)
                    {
                        boolean willSignal = Encrypter.willSignal(angle, theta1, theta2, threshold);

                        boolean angleExistsInConditions = false;
                        for (int condition = 1; condition <= NUM_DOUBLE_THETA_CONDITIONS; condition ++)
                        {
                            double[] bounds = Conditions.doubleThetaConditionBounds(condition, theta1, theta2, threshold);
                            angleExistsInConditions = (!willSignal && ((bounds[0] < angle && angle < bounds[1]) || (bounds[0] + 180 < angle && angle <
                                                    bounds[1] + 180)) ||
                                                            willSignal && ((bounds[0] <= angle && angle <= bounds[1]) || (bounds[0] + 180 <= angle && angle <=
                                                     bounds[1] + 180)));

                            if (angleExistsInConditions)
                                break;
                        }

```

```
                        String angleText = angle + " :\t" + theta1 + ", " + theta
2 + " (" + (10*T) + "%)";
                        if (willSignal && !angleExistsInConditions)
                        {
                            System.out.println(angleText + "  \t-
> CORRECT ANGLE DOES NOT EXIST IN CONDITIONS");
                            System.out.println(Encrypter.signal(angle, theta1, th
eta2) + " > " + threshold);
                        }
                        else if (!willSignal && angleExistsInConditions)
                            System.out.println(angleText + "  \t-
> INCORRECT ANGLE EXISTS IN CONDITIONS");
                    }
                }
            }
        }
    }
    */
    /*
    // MAKE SURE CONDITIONS INCLUDE ALL CORRECT ANGLES (ALL POTENTIAL SIGNAL ANGL
ES ARE COVERED BY CONDITIONS)
    public static void inclusiveTestTripleThetaConditions()
    {
        for (int theta3 = 0; theta3 < 180; theta3 ++)
        {
            for (int theta2 = theta3 + 1; theta2 < 180; theta2 ++)
            {
                for (int theta1 = theta2 + 1; theta1 < 180; theta1 ++)
                {
                    for (int threshold = 1; threshold < 10; threshold ++)
                    {
                        for (int angle = 0; angle < 180; angle ++)
                        {
                            boolean willSignal = Encrypter.willSignal(angle, thet
a1, theta2, theta3, threshold);

                            boolean angleExistsInConditions = false;
                            for (int condition = 1; condition <= NUM_TRIPLE_THETA
_CONDITIONS; condition ++)
                            {
                                if (angleExistsInConditions)
                                    break;

                                double[] bounds = Conditions.tripleThetaCondition
Bounds(condition, theta1, theta2, theta3, (double)(threshold) / 10);
                                angleExistsInConditions = (bounds[0] < angle && a
ngle < bounds[1]);

                                if (!willSignal && angleExistsInConditions)
                                    System.out.println(bounds[0] + " < " + angle
+ " < " + bounds[1]);
                            }

                            String angleText = angle + " :\t" + theta1 + ", " + t
heta2 + ", " + theta3 + " (" + threshold + ")";
                            if (willSignal && !angleExistsInConditions)
                            {
                                System.out.println(angleText + "  \t-
> CORRECT ANGLE DOES NOT EXIST IN CONDITIONS");
                                System.out.println();
                            }
```

```
                                else if (!willSignal && angleExistsInConditions)
                                    System.out.println(angleText + "  \t-
    > INCORRECT ANGLE EXISTS IN CONDITIONS");
                            }
                        }
                    }
                }
            }*/

// HELPERS

    public static BigDecimal round(BigDecimal a)
    {
        return a.setScale(2, RoundingMode.HALF_UP);
    }
}
```