pyBSM: A Python package for modeling imaging systems

Daniel A. LeMaster*, Michael T. Eismann
Air Force Research Laboratory, 2241 Avionics Circle, Wright Patterson AFB, OH, USA 45433

ABSTRACT

There are components that are common to all electro-optical and infrared imaging system performance models. The purpose of the Python Based Sensor Model (pyBSM) is to provide open source access to these functions for other researchers to build upon. Specifically, pyBSM implements much of the capability found in the ERIM Image Based Sensor Model (IBSM) V2.0 along with some improvements. The paper also includes two use-case examples. First, performance of an airborne imaging system is modeled using the General Image Quality Equation (GIQE). The results are then decomposed into factors affecting noise and resolution. Second, pyBSM is paired with openCV to evaluate performance of an algorithm used to detect objects in an image.

Keywords: imaging systems, performance model

1. INTRODUCTION

Performance models appear at every point in the product cycle of an imaging system. The development team provides modeled results in a proposal, the manufacturer models sub-system cost versus performance, the acquisition agency models performance in their intended operating conditions, and the end-user models how to employ their equipment. Existing tools such as the Night Vision Integrated Performance Model (NV-IPM)[1], provide an excellent solution for some. Others develop their own tools for various reasons (e.g. to control of the code and ease of integration). The Python Based Sensor Model (pyBSM) falls somewhere between these two options. It provides all of the basic components for users that are interested in quick results while also providing complete access and flexibility within the code itself. Python is the environment of choice because it is widely available, free, and is supported by a large group of developers. Implementation in Python also allows for rapid integration into existing projects including those in other environments such as C, Java, and MATLAB. Additionally, users can mesh pyBSM output with a large array of Python modules such as matplotlib and openCV which are both used in this paper. pyBSM itself is built largely on top of the numpy module.

pyBSM implements the ERIM Image Based Sensor Model (IBSM)[2,16] along with an accumulation of improvements. Within the code, and in this paper, functions are cross-referenced with the corresponding IBSM equation number. Those that are new to sensor modeling are encouraged to consult other references such as Holst[3] or Driggers et al.[4].

The purpose of this paper is to introduce the model and provide inspiration for its use. A brief overview of the model (and a few details on how pyBSM and IBSM differ) are included in Section 2. Section 3 is demonstration of an airborne imaging system evaluated with the General Image Quality Equation (GIQE) and then decomposed in terms of noise and resolution contributions. Section 4 is another demonstration where pyBSM is combined with an openCV feature-based classifier to predict face and eye detection performance as a function of camera parameters and range.

1.1 How to get pyBSM

For now, pyBSM can be obtained by contacting the authors directly.

2. MODEL COMPONENTS

The primary components of pyBSM can be arranged in the following categories: radiative transfer (how radiation travels from the target to the sensor and everything it picks up along the way), detection (how this radiation is converted into a digital signal), optical transfer function (all things that affect representation of fine details in the scene), sampling, and scenario details such as range and platform motion. Each of these categories is introduced in this Section. IBSM Chapters 3 and 5 should be the primary theory of operations reference. There are also support functions in pyBSM that are not discussed but are documented within the code itself (as an example, calculation of the sensor nadir angle above

Long-Range Imaging II, edited by Eric J. Kelmelis, Proc. of SPIE Vol. 10204,
1020405 · © 2017 SPIE · CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2262561

the curved earth). Figure 1 summarizes some commonly used components of pyBSM though additional capabilities are included.

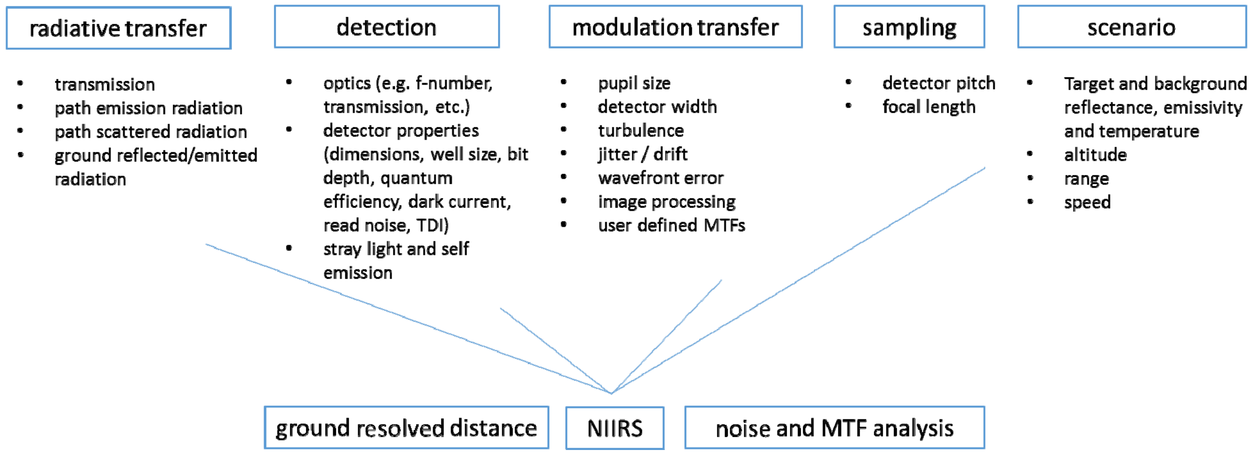

radiative transfer
• transmission
• path emission radiation
• path scattered radiation
• ground reflected/emitted radiation
detection
• optics (e.g. f-number, transmission, etc.)
• detector properties (dimensions, well size, bit depth, quantum efficiency, dark current, read noise, TDI)
• stray light and self emission
modulation transfer
• pupil size
• detector width
• turbulence
• jitter / drift
• wavefront error
• image processing
• user defined MTFs
sampling
• detector pitch
• focal length
scenario
• Target and background reflectance, emissivity and temperature
• altitude
• range
• speed
ground resolved distance
NIIRS
noise and MTF analysis

Figure 1. Commonly used components of pyBSM.

2.1 Radiative Transfer

Radiation arriving at the sensor aperture is a combination of target, background, and atmospheric contributions. To this end, pyBSM offers a pre-calculated radiative transfer database as a convenient starting point. The database was generated using MODTRAN 5.2.1 using the mid-latitude summer model with rural aerosols (both the 5 km and 23 km visibility cases) over a 0.3 to 14 µm spectral range. Full details are included in Appendix A. The database itself follows a convention very similar to MODTRAN *.7sc files. For a given sensor altitude, ground range to the target, and visibility condition, radiance at the aperture is given by:

$$\mathrm{L_{ap}}(\lambda) = \mathrm{L_{pt}}(\lambda) + [1.0 - \mathrm{r}(\lambda)]\mathrm{L_{bb}}(\lambda, T_{tgt})\tau(\lambda) + \mathrm{L_{ss}}(\lambda) + \mathrm{L_{tr}}(\lambda)\frac{\mathrm{r}(\lambda)}{\mathrm{r_{db}}} , \quad (1)$$

where L_{ap}, L_{pt}, L_{bb}, L_{ss}, and L_{tr} are radiance at the aperture, path thermal radiance, blackbody radiance, solar scattered radiance and target leaving radiance. The term, τ, is transmission along the path (TRANS), r is the target spectral directional hemispheric reflectance, r_{db} is the (spectrally constant) target reflectance used in the database, T_{tgt} is the target temperature, and λ is wavelength.

Equation 1 represents the case where an extended Lambertian target lies flat against a Lambertian background without influence from adjacent structures, clouds, etc. The database background was selected to be 'Mixed Forest' because its spectral reflectance lies somewhere in the middle of all the options supplied with MODTRAN and is a much more reasonable choice than assuming a constant background in terms of scattered light. Equation 1 can also be applied to calculate radiance due to the local background which is useful in contrast calculations. When used this way, the model can be visualized as a target surrounded by a local background which, in turn, is surrounded by the Mixed Forest background. In this way, the forest reflectance extends out to the horizon and only contributes to the path scattering terms.

The primary benefits of the radiative transfer database are speed and code portability. Especially at shorter wavelengths, it is orders of magnitude faster to use than running MODTRAN. Also, it is not necessary to have or to understand MODTRAN in order to model useful results. However, there are many cases where a pre-calculated database is insufficient. Leveraging the new JavaScript Object Notation (JSON) input format, a pyBSM interface to MODTRAN 6[5] is planned for the near future.

2.2 Detection

Radiation collected by the aperture passes through the optical system and forms an image at the Focal Plane Array (FPA). Along the way, additional radiation is added from sources of stray light and, in the infrared bands, radiation emitted by the sensor itself. The FPA transduces this radiation into a photocurrent which, along with dark current, is accumulated and digitized by the Read Out Integrated Circuit (ROIC). pyBSM varies slightly from IBSM in how aperture radiance is converted to into irradiance, E_{fpa}, at the FPA:

$$\mathrm{E}_{\mathrm{fpa}}(\lambda) = \mathrm{L}_{\mathrm{ap}}(\lambda)\frac{\tau_o(\lambda)}{\Omega} + \mathrm{E}_{\mathrm{stray}}(\lambda) + E_{se}(\lambda), \tag{2}$$

where τ_o is system optical transmission (including the effects of obscurations), Ω is throughput, *f* is focal length, *D* is the aperture diameter, E_{stray} represents stray light and E_{se} is sensor self-emission. The primary differences between Equation (2) and IBSM Equation 3-34 include maintaining the spectral nature of the calculation throughout, modifying the throughput so that it is applicable over a wider range of f-numbers[6], and explicitly calculating the self-emission terms in the infrared bands.

2.3 Noise

IBSM and pyBSM calculate noise in units of root-mean-squared photoelectrons. Photon related noise terms include radiation arriving at the sensor, stray light and any sensor self-emissions. Dark current contributes noise as well. Quantization noise is contained within IBSM Equation 3-47 but is broken out separately in pyBSM. Finally, a readout noise term is also included to account for the detector readout and multiplexer circuitry.

2.4 Optical Transfer Function

Strictly speaking, the Optical Transfer Function (OTF) is defined to be the Fourier transform of the optical point spread function[7]. Within IBSM and pyBSM, the term is used more broadly to include anything that affects image contrast as a function of spatial frequency. This broader definition is commonplace for Modulation Transfer Functions (MTF) and we have simply extended it to OTF as shorthand for the magnitude and phase of any such transfer function. Table 1 lists all of the OTFs available in pyBSM. Only a subset of these will be used in any given simulation.

Table 1. Optical Transfer Functions available in pyBSM

Name	IBSM reference	Purpose
Turbulence	3-3, 3-9	The long, short or blended exposure turbulence OTF.
Diffraction from a circular aperture	3-20	Obscured circular aperture diffraction OTF. If the obscuration is set to 0, the function will return the unobscured aperture result.
Defocus	3-25	Gaussian approximation for defocus. Valid near the optical axis.
Jitter	3-28	Blur due to random line-of-sight motion that occurs at high frequency, i.e. many small random changes in line-of-sight during a single integration time.
Drift	3-29	Blur due to constant angular line-of-sight motion during the integration time.
Wavefront	3-31	Blur due to small random wavefront errors in the pupil. Refer to the discussion on random phase screens in Goodman[7] for a full explanation.
User defined OTFs	Sections 3.2.6-7	Imports user defined OTFs in various input formats.
Detector	3-36	A simplified version of IBSM Equation 3-36. Blur due to the integrating effect of the detector size.
TDI	3-38	Blur due to a mismatch between the time-delay-integration clocking rate and the image motion.
Charge Transfer	3-39	Blur due to charge transfer efficiency losses in a CCD array.

Efficiency		
Charge Diffusion	3-40	Blur due to the effects of minority carrier diffusion in a CCD sensor. This is useful for demonstrations but does not apply to modern detector structures.
Gaussian		A generic MTF with a Gaussian shape. The MTF is parameterized by a cutoff frequency, f_c. "Cutoff" occurs at about 4% modulation.

There are several OTF implementation details that differ from IBSM. First, the coherence diameter in IBSM 3-5 (used in calculation of the turbulence OTF) is generalized with the definition from Schmidt[8]. Also, pyBSM replaces the refractive index structure parameter profiles in IBSM 3-6 through 3-8 with the often-cited Hufnagel-Valley Turbulence profile[9]. IBSM includes boundary layer / aero-optic turbulence OTF that is not implemented in pyBSM due to its uncertain range of applicability.

The diffraction, wavefront, and turbulence OTFs are all functions of wavelength. In these cases, a composite OTF, *g*, is weighted using the unity normalized spectral photoelectron generation rate discussed in Section 2.2:

$$g(u,v) = \sum w(\lambda) g_\lambda (u,v). \tag{3}$$

where $w(\lambda)$ is the weighting function and g_λ is the OTF at wavelength λ.

A sharpening kernel OTF is also included in pyBSM. Below the Nyquist frequency, this OTF is represented by the discrete Fourier transform of the sharpening kernel. Calculation of the OTF above Nyquist follows from the periodic nature of the discrete space Fourier transform. (It may seem strange to care about what happens above Nyquist but image content at these spatial frequencies are affected by the sharpening filter even when they are aliased.)

2.5 Sampling

Sampling in pyBSM and IBSM follows the conventional geometric treatment in terms of instantaneous field-of-view. Like the OTFs, the Nyquist frequency is tracked in units of cycles/radian.

2.6 Scenario

The model also makes accommodation for sensor altitude, range to target, and speed as these things affect radiative transfer, turbulence, and so on. The geometry underpinning pyBSM calculations is based on a circular Earth model (radius of 6378.164 km).

2.7 Model Limitations

The pyBSM model does not explicitly accommodate space-varying blur functions though, in implementation, the user may run the model multiple times, parameterized by field location. Similarly, the model does not account for space-varying distortion either within the optical system itself or as introduced externally through turbulence. The detection model assumes a linear response to incident radiation within the range of the detector well. Furthermore, the noise model does not account for other sources of detector noise (e.g. 1/f, Johnson) that may be important in some cases. Finally, the model allows some things to occur without considering the impact of OTF. A good example of this is frame stacking to increase the ratio of signal-to-noise. Unmodeled blur will result if frame-to-frame registration is worse than some small fraction of a pixel. There may also be unmodeled parallax present between objects in adjacent frames. Pointing out these limitations also highlights one of the model's strengths. Because pyBSM is open-source, it is easy to expand the capability to meet new needs. Indeed, one of the motivations for releasing the code is to take advantage of the improvements that others in the user community can provide.

3. EXAMPLE: THE GENERAL IMAGE QUALITY EQUATION

The General Image Quality Equation maps sensor performance into task performance on the National Image Interpretability Rating Scale (NIIRS)[10]. There are multiple versions of the GIQE. The version included in IBSM is given by:

$$NIIRS = 11.81 + 3.32\log_{10}\left(\frac{RER}{GSD}\right) - 1.48EHO - \frac{G}{SNR}, \tag{4}$$

where the Relative Edge Response (RER) and Edge Height Overshoot (EHO) are functions of the system MTF, G is noise gain due to sharpening/enhancement in post processing, GSD is the Ground Sample Distance, and SNR is the contrast Signal-to-Noise Ratio. The expressions for calculating each of these terms can be found in the IBSM documentation.

In this example, pyBSM is used to calculate NIIRS performance as a function of range and atmospheric conditions for the full motion video imaging system described by Olson[11] (along with our own assumptions about a few missing pieces of information). Camera performance is summarized using the GIQE and then expanded upon in terms of noise and MTF contributions. The reader is encouraged to study functions *pybsm.niirs*, *pybsm.commonOTFs*, and *pybsm.photonDetectionSNR* for full details of the calculation.

3.1 Scenario and assumptions

In this scenario, the camera is flown at an altitude of 9 km with ground speed of 100 m/s. To fully flesh out the model, we will add to the specifications provided in the Olson paper. We assume a relative linear telescope obscuration of 0.4, a quantum efficiency curve consistent with a high performance silicon CCD, dark current density of 1 nA/cm^2, ¼ pixel root-mean-square high frequency jitter and 100 μradians/s drift. Finally, we place a 60% detector well fill limit on integration time. This choice is made because the GIQE standard target reflectance is 15% and we want to avoid the risk of saturating on more reflective (but unmodeled) objects in the scene. The 60% well fill limit is applied uniformly for simplicity in this example but it is actually too restrictive in the low visibility case at long range. Sensor details are included in Table 2. All other possible sensor parameters are assigned to a default value.

Table 2. Sensor parameters.

Name	Value	Name	Value
Focal length (m)	4.0	Detector well (electrons)	96000
Aperture (m)	0.275	Bit depth (bits)	11.9
Optics transmission (unitless)	0.5	Quantum Efficiency (500 to 660 nm)	0.80
Relative linear obscuration	0.4	Quantum Efficiency (550 to 850 nm)	0.71
Detector pitch and width (m)	8.0×10^{-6}	Jitter (rms radians)	5×10^{-7}
Dark current density(A/m^2)	1.0×10^{-5}	Drift (radians/s)	1×10^{-4}
Read noise (rms electrons)	25	Maximum allowed integration time (s)	.03

3.2 Results

Figure 2 shows NIIRS as a function of range for the 5 km and 23 km visibility atmospheres in each of the red (500 nm to 660 nm) and near infrared (550 nm to 850 nm) bands. Using the NIIRS scale, the model predicts that an image analyst will be able to "detect individual spikes in railroad ties" (NIIRS 9) when looking at nadir. At the farthest range, the analyst will be able to "identify individual rail cars by type" (NIIRS 5). Not surprisingly, performance degrades rapidly in the low visibility case, especially in the red band where there is a natural penalty from more scattered light.

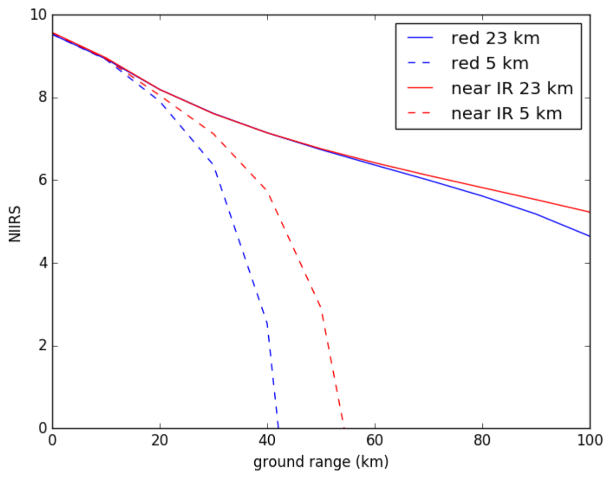


Figure 2. Modeled NIIRS as a function of range and ground visibility (5 km or 23 km).

Figure 2 is useful on its own, but much additional detail is available in the intermediate products of the pyBSM calculation. As an example, SNR as a function of range is shown Figure 3. Without noise gain due to sharpening, Equation 4 shows that the GIQE penalty is small until contrast signal-to-noise falls below 10 or so. Consequently, SNR is not appreciably degrading performance until the longest ranges for the high visibility case. In the low visibility case, NIIRS plummets at comparatively short range because the SNR term dominates.

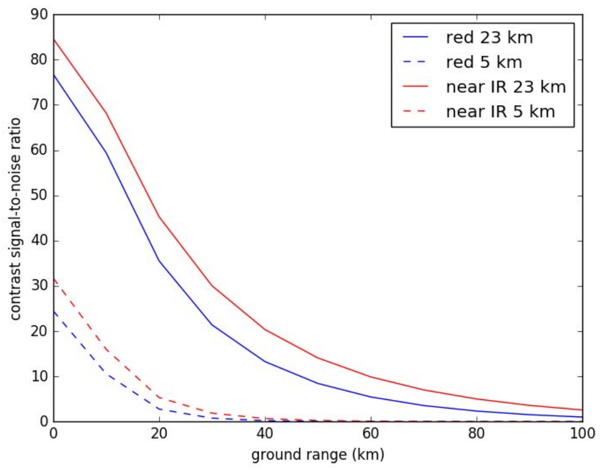
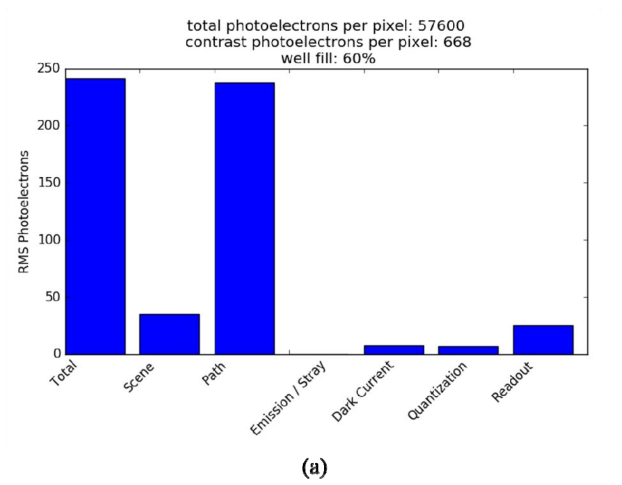


Figure 3. Modeled SNR as a function of range and ground visibility (5 km and 23 km).

Figure 4 provides insight into what is contributing to the noise terms in the contrast SNR calculation for the low visibility case at 20 km ground range. As should be expected, the near infrared out-performs the red band at long range due to superior haze penetration. This is understood by comparing the path radiance contributions. In both cases, the path radiance term is the largest contributor to well fill and noise but, in the near infrared case, this noise is appreciably lower and the contrast photoelectron count is higher.



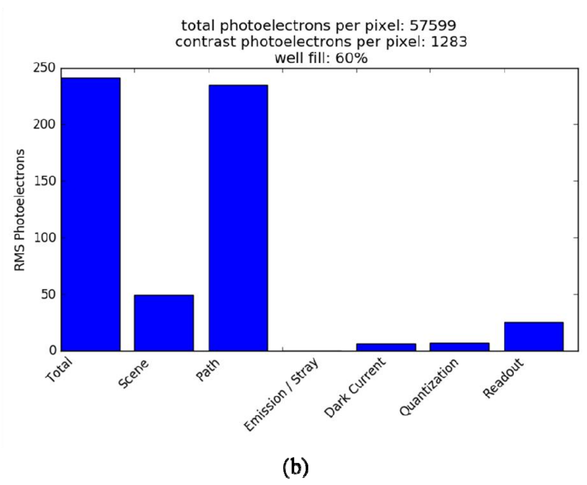
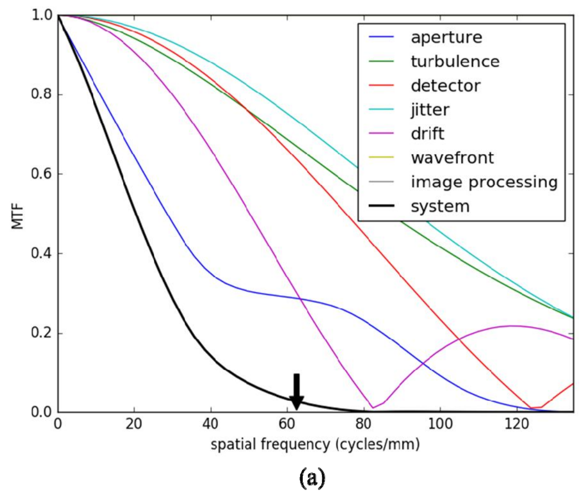


Figure 4. A breakdown of contrast SNR components at 20 km ground range in the low visibility case. Figure (a) is the red band and (b) is the near infrared band. Total noise is the root sum square of the component noise sources.

It is also worthwhile to consider how changing integration times affect MTF. Figure 5 is a comparison of MTF components at nadir and at 100 km ground range in the red band for the high visibility case. The MTF plots show that, at nadir, the drift MTF is the second largest contributor to image degradation and turbulence effects are weak. At 100 km ground range, where the integration time was reduced, the drift MTF is improved but turbulence dominates.



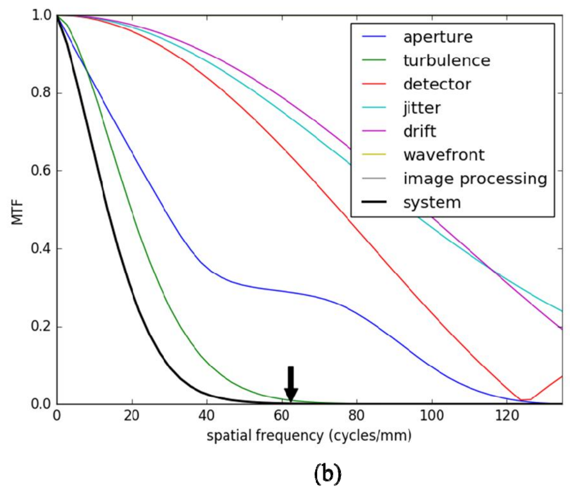


Figure 5. A breakdown of MTF components in the image plane. Figure (a) is at 0 km ground range and (b) is at 100 km. The arrow indicates the Nyquist frequency.

The interested reader may wish to take these analyses further. For instance, SNR could be traded for improved drift MTF at nadir by reducing the maximum allowable integration time. Ranges corresponding to high SNR may also benefit from applying a sharpening filter (essentially trading noise gain for MTF). In a similar way, it also straightforward to model trades in the optics, focal plane array, and stabilization system. Let this section serve as an introduction to the many possibilities.

4. EXAMPLE: OBJECT DETECTION

This second example highlights how pyBSM can be readily extended to applications beyond point models like the GIQE. Here, pyBSM is used to simulate imagery. This imagery is then fed into the openCV[12] Harr feature-based cascade classifier and scored for probability of detection and false alarm rate. In this case, the classifier has been trained to find human faces and eyes. The interested reader may compare this to a more extensive study by Howell et al[13], that uses the NV-IPM image generation tool to study the effectiveness of the PITT-PATT algorithm for face identification.

The Yale Face Database[14], which consists of 15 human subjects each with 11 different facial expressions, serves as the target dataset. pyBSM is used to degrade image quality for each face as a function of range, drift, etc. The simulated camera specifications are selected to be in the range of typical smart phone performance (focal length is 4.1mm, pupil diameter is 1.86mm, detector pitch 1.5 μm) though MTF effects of color filter array demosaicing are not included in order to keep the example simple. With fast optics, it is unrealistic to assume that the system MTF is ideal (e.g. diffraction plus detector MTF). As a substitute, a Gaussian MTF with cutoff at 0.7 cycles/pixel is assumed as a baseline. For each range, each image, *o*, in the face database is degraded to form the image, *i*, in the following way:

$$i = R[h \otimes o], \tag{5}$$

where R is the resampling operator, $\otimes$ denotes convolution, and the blur kernel, h, comes from

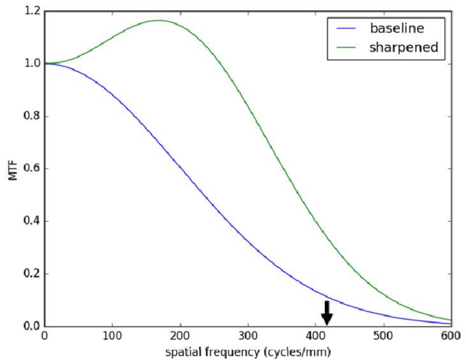
$$h = R[[F[g]]], \tag{6}$$

where F is the discrete Fourier transform and g is the transfer function generated from pyBSM. The resampling operator in Equation 6 is used to match the sampling of the blur kernel to the native sampling of o. It is the resampling in Equation 5 that accounts for the changes in projected sampling with range.

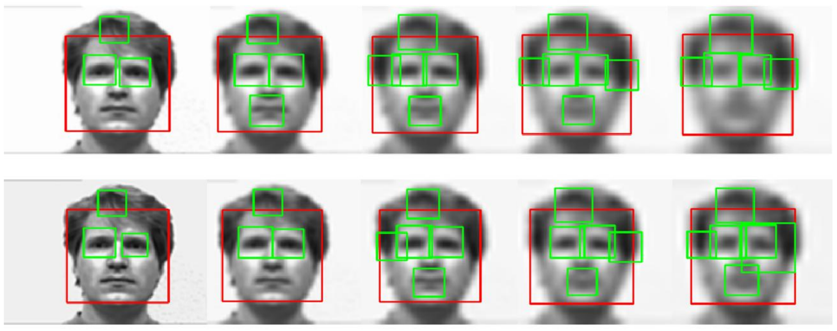
The Harr classifier in openCV is based on the work of Viola and Jones[15]. There are a number of classifier options that affect algorithm performance. For instance, the algorithm iterates over multiple scale sizes to account for differences in geometry between the training data and the image under test (for our test, we set the scale factor between iterations to 1.05). There is also a parameter that specifies what minimum number of neighboring pixels must be included for a detection to be achieved (we use 3 neighboring pixels). In Python, see the help on *cv2.CascadeClassifier* for details and additional options. The present goal is to demonstrate simple use cases for pyBSM but, if the goal were to fully evaluate classifier performance, these parameters would be varied as part of the trade space as well.

Success or failure of the classifier is scored against the true locations of the head and eye features. The true eye and head centers were determined by hand for each true image. Successful detection is defined in any region that includes the true target center pixel. A false alarm is scored to be any detection that does not include the target center pixel. In the case of overlapping detections, only one is scored as a detection and the others are considered false alarms (this avoids reporting faces with three or more eyes). Probability of detection is estimated to be the ratio of true detections to total possible detections. The false alarm rate is defined to be the ratio of false alarms over total number of images (i.e. false alarms per frame).

The first example compares classification as a function of range with and without sharpening. The minimum range used in the simulation is 10 meters to minimize sampling and blurring effects from in the "true" face images. pyBSM is used to ingest the baseline camera MTF and to generate the transfer function for a 3x3 sharpening kernel. Figure 6 shows the baseline camera MTF and the MTF after sharpening along with some example detection results. Figure 7 is the corresponding classifier probabilities of detection and false alarm rates as a function of range for the unsharpened and sharpened cases.

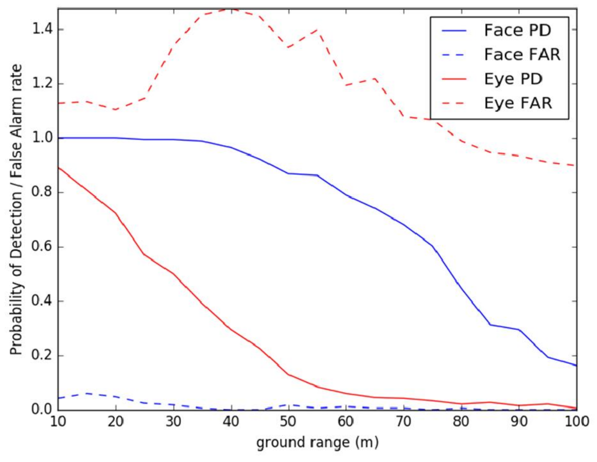


(a)

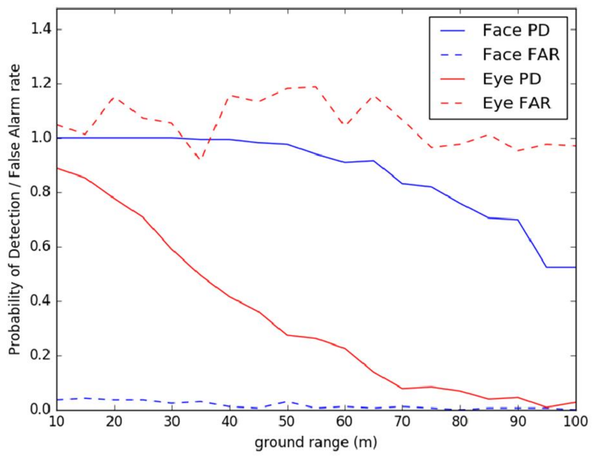


(b)

Figure 6. (a) pyBSM generated MTFs for the face and eye detection problem. The arrow identifies the Nyquist sampling frequency. (b) Top row: unsharpened detection result at 10 through 50 m ranges (10 m steps). Bottom row: the sharpened detection results.



(a)



(b)

Figure 7. Face and eye classifier performance as a function of range. (a) baseline camera MTF and (b) with sharpening. In the legend, PD is probability of detection and FAR is false alarm rate (false alarms per frame).

The results in Figure 7 are intended as a simplified demonstration. Within this limited range of applicability, it is useful to draw some conclusions. Despite exacerbated aliasing, sharpening improves probability of face detection without appreciably affecting false alarms. This conclusion may not hold against a more realistic cluttered background. It is also interesting to note that eye detection degrades with range but false alarm rates do not correlate. Eye detections and false alarm rates are improved somewhat with sharpening but are unacceptably high overall. (On average, there is more than one false eye in each image!) It is very likely that this false alarm rate could be made more manageable through common-sense mitigation steps. As previously stated, the goal here is simply to demonstrate the process rather than to optimize performance.

The next example explores the effect of blur due to line-of-sight drift on classifier performance. The range to target is fixed a 10 m and drift is allowed to vary between 0 and 40 instantaneous fields-of-view in the horizontal and vertical directions. Figure 8 shows two-dimensional MTFs and resulting images for three drift values in each direction. Figure 9 is the corresponding classifier results. Within the limited scope of the example, it is interesting to find that (a) the detector has a small preference for vertical drift and (b) face detection is particularly impervious to drift, at least against this clutter-free background.

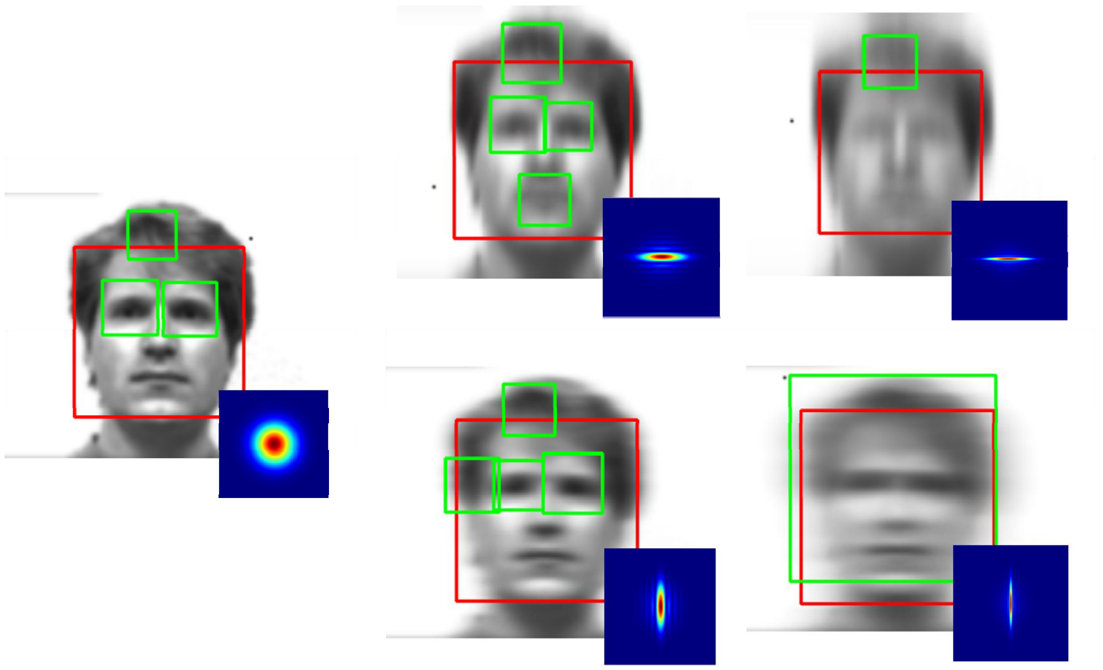
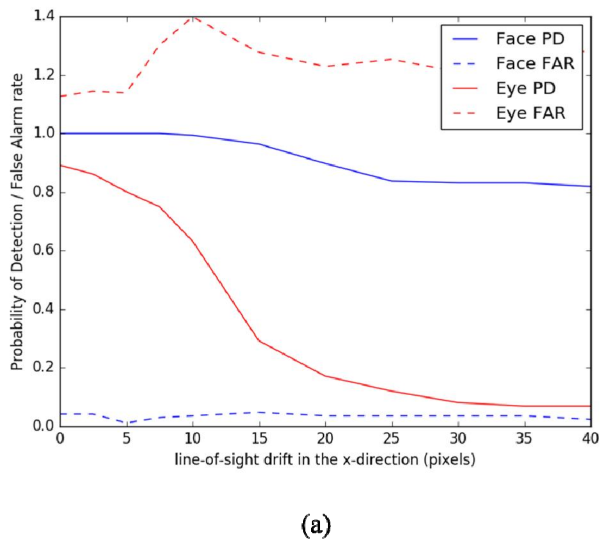


Figure 8. Example images degraded by horizontal or vertical drift with inset MTFs. Red boxes show regions identified as faces by the classifier. Green boxes indicate regions classified as eyes.



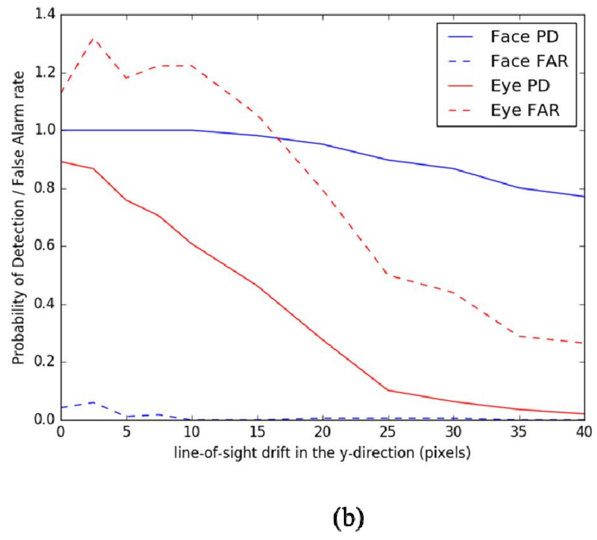


Figure 9. Face and eye classifier performance as a function of line-of-sight drift. (a) horizontal drift (b) vertical drift. PD is probability of detection and FAR is false alarm rate (false alarms per frame).

As with the GIQE example, pyBSM can be utilized for many additional variations on this experiment. As was shown in the Howell paper, an obvious next step would be to apply the noise model and parameterize classification as a function of ambient illumination.

5. CONCLUSION

The Python Based Image Model (pyBSM) is an open-source tool for predicting electro-optical imaging system performance. Python is the environment of choice because it is widely available, free, can readily interface with most other programming languages, and is supported by a large group of developers. By distributing pyBSM, we hope to reduce model development costs for both government and industry. We also hope that the open code philosophy will engender confidence, simplify comparisons, and promote use in education.

Two examples are provided in this paper. The first models NIIRS performance as a function of range and visibility for an airborne imaging system. pyBSM is then used to break down this performance in terms of noise and MTF contributions. Everything needed to run this first example is included with pyBSM. The second example combines pyBSM, openCV, and a database of test imagery to probe performance of an image-based classifier as a function of range and other MTF considerations. These examples, though neither exhaustive nor even complete, are intended to illustrate the possibilities for putting pyBSM to use.

APPENDIX A: ATMOSPHERE DATABASE

Database altitudes include (km): .002 .0325 .075 .15 .225 .5, 1 to 12 in 1 km steps, and 14 to 20 in 2 km steps. The following ground ranges are included in the database at each altitude until the ground range exceeds the distance to the spherical earth horizon: 0 .1 .5 1 to 20 in 1 km steps, 22 to 80 in 2 km steps, and 85 to 300 in 5 km steps. The database contains two IHAZE options 1 - rural extinction, 23 km visibility and 2 - rural extinction, 5 km visibility.

Each file in the database contains the following columns from the MODTRAN *.7sc file (in this order): TRANS, PTH THRML, SURF EMIS, SOL SCAT, GRND RFLT. Each of these columns is calculated between the wavelength range of 0.3 to 14 um in .01 um steps (including the end points, e.g. .3, .31, ... 14) . Therefore, each database file is 1371 rows by 5 columns. The units for all radiance columns are: W/(sr cm^2 um).

For all model runs, the following MODTRAN parameters are fixed (only the most salient shown):

MODTRAN designation	Value	Purpose
GNDALT	0.0	Ground altitude (km MSL)
H2	0.0	Target altitude (km MSL)
PARM1	90	Azimuth between sensor line of sight and sun (degrees)
PARM2	50	Solar zenith angle (degrees)
DV	.01	Spectral sampling (um)
FWHM	.02	Full width half max (um)
VIS	0	Visibility (0 means that visibility is inherited from IHAZE)
MODEL	2	Mid-Latitude Summer (45 degrees North Latitude)
ICLD	0	No clouds
IDAY	236	Day of year (August 23rd)
IMULT	1	Multiple scattering (1=yes, 0=no)
RAINRT	0	No rain

TPTEMP	299	Target temperature (K)
AATEMP	297	Background temperature (K)
SURREFt	constant, 15%	Target reflectance
SURREFb	'Mixed Forest'	Background reflectance

REFERENCES

[1] "Night Vision Integrated Performance Model - CERDEC – Army", 18 March 2015, <http://http://www.cerdec.army.mil/inside_cerdec/nvesd/integrated_performance_model/> (14 February 2017)

[2] Eismann, M. T., S. D. Ingle, and M. Slyz., [Utility Analysis of High-Resolution Multispectral Imagery. Volume 4. Image Based Sensor Model (IBSM) Version 2.0 Technical Description.], No. ERIM-253885-4-F-VOL-4. Environmental Research Institute of Michigan, Ann Arbor, 1996.

[3] Holst, G., [Electro-Optical Imaging System Performance: Staring Arrays], SPIE Press and JCD Publishing, 2017.

[4] Driggers, R.., Friedman, M. and Nichols, J. [Introduction to infrared and electro-optical systems]. Artech House, 2012.

[5] Berk, A., P. Conforti, R. Kennett, T. Perkins, F. Hawes, and J. van den Bosch, "MODTRAN6: a major upgrade of the MODTRAN radiative transfer code," Proc. SPIE 9088, (June 13, 2014)

[6] Schott, J., [Remote Sensing: The Image Chain Approach], Oxford University Press, 1997

[7] Goodman, J., [Introduction to Fourier optics], Roberts and Company Publishers, 2005.

[8] Schmidt, J., [Numerical simulation of optical wave propagation with examples in MATLAB]. Bellingham, WA: SPIE, 2010.

[9] Roggemann, M., Welsh, B., Hunt, B., [Imaging through turbulence] CRC press, 1996.

[10] Leachtenauer, Jon C., et al., "General image-quality equation: GIQE." Applied optics 36.32 (1997): 8322-8328.

[11] Olson, C., et al., "Optimizing resolution efficiency for long-range EOIR multispectral imaging sensors." Proc. SPIE 9846, (2016).

[12] "OpenCV", <http://opencv.org> (14 February 2017)

[13] Christopher Howell, Hee-Sue Choi and Joseph Reynolds, "Face Acquisition Camera Design Using the NV-IPM Image Generation Tool," Proc. of SPIE, Vol 9452, (2015)

[14] Belhumeur, P., et al., "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection," IEEE Transactions on Pattern Analysis and Machine Intelligence, 711-720 (1997).

[15] Viola, P., and Jones, M., "Rapid object detection using a boosted cascade of simple features." Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. Vol. 1. IEEE, 2001.

[16] W. F. Green, M. T. Eismann, and S. D. Ingle, "System Performance Modeling using the Khoros 4GL (4th Generation Language) and Cantata VPE (Visual Programming Environment)," Proc SPIE. 2829, (1996).