Specific research description

1. Aim of research

In this research, the concept of integrated computational imaging system (ICIS) is explored. It is a philosophy that revolutionizes the imaging system design methodology. Conventionally, each imaging system is divided into 3 separate parts: optical front-end, optoelectronic sensor and post-detection signal processing. Each part is designed and optimized individually. However, under ICIS, these 3 parts are jointly optimized. Each part needs not be designed to its highest precision and quality. The decreased performance of one part can be complemented by another part, so that the performance of the whole imaging system is not affected, or even improved. This allows for more design flexibility and higher system capability. Functions that cannot be achieved by conventional imaging systems may also be accomplished by ICIS. The cost may be lowered as a result.

The aim of this research is to apply ICIS to the problem of die inspection. It is hoped that an imaging system can be designed, assembled and applied in the industrial process of ASM Pacific.

2. Wavefront Coding™ technology

As a starting point, we researched on a successful example of ICIS: Wavefront Coding™ [1]. It is a promising technology developed by Edward R. Dowski, Jr. and W. Thomas Cathey from the University of Colorado, U.S.A. They formed a company called CDM Optics for developing products using Wavefront Coding™ technology.

Basically, Wavefront Coding™ makes use of a piece of tailor-made lens called cubic phase mask and a digital image restoration filter (can be implemented in hardware or software) to achieve extended depth of field and aberration invariance. The cubic phase mask has a cubic surface, which creates a rectangularly separable cubic phase distribution over the exit pupil of an imaging system. The surface equation of cubic phase mask is given by
\[ z = \frac{t}{2R^3}(x^3 + y^3), \quad (1) \]

where \( t \) is the maximum thickness of cubic phase mask and \( R \) is the pupil radius. Its cross-section along \( y = 0 \) is shown in Fig. 1.

![Fig. 1 Cross-section of cubic phase mask along y = 0.](image)

If the mask refractive index is \( n \) and \( k \) is the wave number, the phase distribution over the mask is derived as follows:

\[
\Phi(x, y) = kn \left( \frac{t}{2} + z \right) + k \left[ t - \left( \frac{t}{2} + z \right) \right] \\
= k(n-1)z + k(n+1)\frac{t}{2} \\
= \frac{t}{2R^3}(n-1)(x^3 + y^3) + k(n+1)\frac{t}{2}.
\]

The pupil function is then given by:

\[
P(x, y) = e^{i\Phi(x, y)} = e^{ik(n+1)\frac{t}{2} + i\frac{t}{2k(n-1)}(x^3 + y^3)} \quad \text{for } \sqrt{x^2 + y^2} \leq R. \quad (2)
\]

Neglecting the constant phase change, the pupil function of cubic phase mask is:

\[
P(x, y) = e^{i\frac{\alpha}{n^2}(x^3 + y^3)}, \quad (3)
\]

where \( \alpha = \frac{t}{2}k(n-1) \). This parameter can control the depth of field achieved by cubic phase mask. In CDM Optics website (http://www.cdm-optics.com), a specification for the cubic phase mask was posted previously. It has been removed recently since CDM Optics does not manufacture cubic phase masks any more. Nevertheless, \( \alpha \) can be calculated by using the specification:
With these data, \( \alpha = 59.0478 \approx 60 \). The powerfulness of cubic phase mask lies on its ability to produce focus-invariant optical transfer functions (OTFs). Suppose we have a simple imaging environment shown below (Fig. 2).

![Diagram of imaging environment](image)

**Fig. 2 An imaging environment.** The object distance is to be varied around the ideal value.

From lens formula \( \left( \frac{1}{z_o} + \frac{1}{z_i} = \frac{1}{f} \right) \), the ideal object distance is 15 cm. When the object distance is different from the ideal value, defocus (or misfocus) is resulted. Without cubic phase mask, the OTF is seriously affected by the defocus. This is shown in Fig. 3, where the magnitudes of OTFs (often called modulation transfer functions (MTFs)) for the case of no phase mask are plotted. It is seen that when the object is in focus (blue curve), the MTF curve is smooth and decreases slowly towards zero. No zero point is present except at the cutoff frequency (approx. 72 mm\(^{-1}\)). However, when the object is out of focus by 0.5 cm (green curve) and 1 cm (red curve), the MTF curves decrease quickly and reach zeros at several frequencies. Those frequency components are not transmitted by the imaging system and no image restoration algorithm can recover them.
When cubic phase mask is installed and the object is in focus, the MTF values are comparatively lower than those for no phase mask (compare the blue curve in Fig. 3 and the blue curve in Fig. 4). The image is blurred even the object is in its in-focus position. Nevertheless, the MTF curves remain the same when the object is out of focus (green and red curves). The OTF is invariant to the object position shift. It implies that the severity of blurring is about the same towards near or far objects. This is the most important function of cubic phase mask. Once we get a blurred intermediate image from the image sensor, we can just feed the same OTF (e.g. the in-focus OTF) as input to an image restoration filter and obtain the final, clear image. This cannot be achieved by using spherical lens only, because the OTF changes as the object shifts. In short, a cubic phase mask must be used together with a digital restoration filter to produce the restored image.

From the view of information transfer, cubic phase masks sacrifice some of the object information to achieve focus invariance, and the lost information can be recovered, though not fully, by restoration filters.

![MTF of imaging system (L = 550 nm, phase mask: None, NA = 0.02, λ15)](image)

**Fig. 3** MTFs for the case of spherical lens only (no phase mask installed) at different object positions.
Cubic phase masks can be applied in many situations, to name a few, microscopy, microelectromechanical systems (MEMS), miniature cameras, barcode and fingerprint identification, die inspection, etc. For die inspection, cubic phase masks can find their use when the die consists of several layers. Fig. 5 shows a piece of die with dimensions 5 mm×5 mm. Its image was simulated with the same camera parameters shown in Fig. 2. Fig. 6 shows the images when spherical lens was used only. The image is blurred significantly when the object is out of focus. In Fig. 7, the images produced by spherical lens together with cubic phase mask are shown. They are the images before image restoration. The quality is worse than those produced without the cubic phase mask, but they are just temporary images and will be improved after digital restoration.

Fig. 8 shows the images when cubic phase mask and Wiener filter were added. Wiener filter is a type of restoration filter that minimizes the mean square error (SSE) between the uncorrupted image and the estimated image. It is given by the following expression in the frequency domain:

\[
W(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + S_{\eta}(u,v)/S_f(u,v)},
\]

where \(H(u,v)\) is the OTF of the imaging system, \(S_{\eta}(u,v)\) is the power spectrum of the noise and \(S_f(u,v)\) is the power spectrum of the undegraded image. Since the power spectra of the noise and undegraded image are often unknown, we approximate the ratio \(S_{\eta}(u,v)/S_f(u,v)\) by a constant \(K\). In our simulation, we assume that \(K = 0\), i.e. no noise is present.
Fig. 5 Object – a piece of die with blob defects.

Fig. 6 Images from spherical lens: at $u = 15$ cm (left) and at $u = 13.5$ cm (right).
By comparing the right figures of Fig. 6 and Fig. 8, we can see that the die images are much improved. Some sinusoidal pattern is observed in the right figure of Fig. 8, which is an artifact of Wiener filter. However, the detailed features are still discernible. Therefore, by simulation, we have verified that the depth of field of a Wavefront Coded™ system is extended compared with the conventional systems. The philosophy of ICIS is also realized, as cubic phase mask cannot be used on its own but has to be used in conjunction with a restoration filter. It may be argued that extended depth of field may also be achieved by reducing the aperture size. However, the disadvantage is that the flux of light that enters the camera is much reduced, thereby reducing the overall image contrast.
Fig. 9 MTFs of imaging system with spherical aberration: No phase mask (left) and with cubic phase mask (right)

Fig. 10 MTFs of imaging system with coma: No phase mask (left) and with cubic phase mask (right)

Fig. 11 MTFs of imaging system with astigmatism: No phase mask (left) and with cubic phase mask (right)
Besides extended depth of field, the OTF of cubic phase mask is also invariant to aberration. In terms of geometrical optics, when aberration is present, the one-to-one correspondence between object and image points no longer exists. Fig. 9 - Fig. 11 show how the MTFs vary when three types of Seidel aberration: spherical aberration, coma and astigmatism are present. We can see that the MTF curves for no phase mask vary significantly for each type of Seidel aberration, but the variation is far less when cubic phase mask is used. The curves for cubic phase mask overlap one another. This observation implies that the same OTF can be fed into the restoration filter regardless of the aberrations present.

3. Other phase masks

Having seen the functions of cubic phase mask, we have to ask: Is cubic phase mask the optimized mask? Is it possible to find a mask achieving similar functions, yet easier to be manufactured? Some researchers [2] introduced the concept of pupil phase engineering (PPE), which focuses on the design of a more general pupil-phase mask that can lead to even better performance than the cubic mask in extending the depth of field. Strehl ratio and Fisher information are two proposed mathematical tools for obtaining the optimized phase masks. Other researchers [3] also proposed logarithmic masks and quartic (i.e. 4\textsuperscript{th} degree) masks for focus-invariant and spherical-aberration-invariant imaging. However, the pupil functions describing these phase masks are even more complicated than a cubic function and this increases the difficulty of manufacturing. We propose to use ambiguity function (AF) [1] and defocus transfer function (DTF) [4] for constructing the optimization criterion. However, there is not much progress with this approach.

4. Imaging system simulator – Virtual Camera

For this research, a program called Virtual Camera (VC) has been developed using MATLAB. This software simulates the functions of a digital camera with the addition of different types of phase masks (see Section for details). Fig. 12 shows the graphical user interface (GUI) of Virtual Camera. Users are allowed to set a number of parameters about the camera. All of them are in the leftmost panel with title Input. The object and image are shown in the middle part of the GUI. The output values are in the rightmost panel, and users cannot alter their values. Buttons for operating the software are at the bottom-right corner.

4.1. Quick user guide

Here are the steps for using VC:
1. The default object is a spoke target. If you do not want this object, press Load object button to load another object into VC. In fact, the object is also an image file, but it is treated as an object by VC. The accepted file formats are: 24-bit or 8-bit bitmap (*.bit), Graphics Interchange Format (GIF) (*.gif), Joint Photographic Experts
Group (JPEG) (*.jpg), Portable Network Graphics (PNG) (*.png) and Tagged Image File Format (TIFF) (*.tiff/*.tif). If the object is grayscale (illuminated by monochromatic light), the type of spatially incoherent light source is shown to be Monochromatic. Users are allowed to input the wavelength. If the object is RGB, the light source type is Polychromatic (RGB). Users are not allowed to configure the wavelength. The wavelengths of light source are taken to be 3 values: 675 nm (red light), 525 nm (green light) and 475 nm (blue light).

Fig. 12 GUI of Virtual Camera.

2. Input all the necessary values into the fields in panel Input.

- **Spatially incoherent light source** panel
  - **Wavelength** field: Mean wavelength of a monochromatic light source in nm (default: 550 nm)
- **Object (centered at (0,0) if no translation(s))** panel
- **Width** field: Object width in mm (default: 5 mm)
- **Height** field: Object height in mm (default: 5 mm)
- **Distance** field: Object distance from the entrance pupil of the imaging system in mm (default: 150 mm)
- **Center translation** panel (See Fig. 13 for the coordinate system)
  - \(x\): Object center translation along the \(x\)-direction in mm (default: 0 mm)
  - \(y\): Object center translation along the \(y\)-direction in mm (default: 0 mm)

![Coordinate system used in Virtual Camera.](image)

- **Imaging system** panel
  - **Focal length** field: Focal length of spherical lens in mm (default: 60 mm)
  - **Exit pupil radius** field: Exit pupil (or aperture) radius in mm (default: 1 mm)
  - **Phase mask** popup menu: Choose a phase mask for simulation (See the next section for more information)
    - *None*: No phase mask is inserted
    - *Annular*: Annular pupil
      - *Ratio of radii*: Ratio of inner radius \(R_i\) to outer radius \(R_o\) (default: 0.8)
    - *Cubic*: Cubic phase mask
      - *Refractive index*: Refractive index of the cubic phase mask (default: 1.4938)
      - *Max. thickness*: Maximum thickness \(t\) at the rim (default: 0.0208 mm)
    - *Triangular prism*: Triangular prism with isosceles triangle as the base over a circular pupil
      - *Refractive index*: Refractive index of the triangular prism (default: 1.4938)
      - *Max. thickness*: Height of the isosceles base (default: 0.0208 mm)
- **Cone**: Cone
  - **Refractive index**: Refractive index of the cone (default: 1.4938)
  - **Max. thickness**: Height of the cone (default: 0.0208 mm)

- **Quartic**: Quartic filter
  - **Alpha**: Parameter $\alpha$ as stated in [3] (default: 2.3562)

- **Logarithmic**: Logarithmic filter
  - **Beta0**: Parameter $\beta_0$ as stated in [3] (default: 0.7213)
  - **Beta**: Parameter $\beta$ as stated in [3] (default: 17.5929)

![Fig. 14 Different types of phase masks.](image-url)
Set aberration dialog box. Users can choose between Seidel aberrations and Zernike polynomials to represent the aberration function.

- **Set aberration** button: Open the *Set aberration* dialog box
- **Set aberration** dialog box:
  - **Aberration type** popup menu: Choose whether the imaging system has aberration. If so, choose a way to expand the aberration function.
    - **None**: No aberration
    - **Seidel**: Express the aberration function in terms of Seidel aberrations
    - **Zernike**: Express the aberration function in terms of Zernike polynomials
  - (If **Seidel** is chosen in **Aberration type** popup menu) **Seidel aberration** popup menu: Choose Seidel aberrations
    - **Spherical aberration**: Spherical aberration
    - **Coma**: Coma
    - **Astigmatism**: Astigmatism
    - **Curvature of field**: Curvature of field
    - **Distortion**: Distortion
  - (If **Zernike** is chosen in **Aberration type** popup menu) **Zernike term** popup menu: Choose rms normalized Zernike polynomials. The polynomials are indexed from 1 to 37.
  - **Coefficient** field: Coefficient for the chosen Seidel aberration or Zernike polynomial. It is expressed in terms of the number of wavelengths (default: 0 waves).
  - **Add** button: Add aberration term
  - **Delete** button: Delete aberration term
  - **OK** button: Confirm the aberration configuration and close the dialog box
  - **Cancel** button: Close the dialog box without changing the aberration configuration

- **CCD sensor** panel
- **Distance** field: Distance between the CCD sensor and the exit pupil of the imaging system in mm (default: 100 mm)
- **Pixel width** field: Width of one pixel in micron (µm) (default: 7.4 micron)
- **Pixel height** field: Height of one pixel in micron (µm) (default: 7.4 micron)
- **Array width** field: Width of the whole CCD sensor in mm (default: 3.7888 mm)
- **Array height** field: Height of the whole CCD sensor in mm (default: 3.7888 mm)

- **Noise** panel
  - **Type** popup menu: Choose the type of noise to be added to the intermediate image BEFORE image restoration.
    - **Gaussian**: Gaussian white noise
      - **Mean** field: Mean of the Gaussian distribution in graylevels (default: 0 graylevel)
      - **Standard deviation** field: Standard deviation of the Gaussian distribution in graylevels (default: 7 graylevels)
    - **Poisson**: Poisson noise
    - **Salt & pepper**: Salt and pepper noise
      - **Density** field: Density of the white (salt) and black (pepper) dots. This affects approximately [density*(no. of pixels of image)] pixels (default: 5%).
    - **Speckle**: Speckle noise. It adds multiplicative noise to an image \( I \), using the equation \( J = I + n*I \), where \( n \) is a uniformly distributed random noise with mean 0 and variance \( V \).
      - **Variance** field: Value of variance \( V \) (default: 0.04)

- **Post-processing** panel
  - **Use Wiener filter** checkbox: Check this box when the intermediate image is to be post-processed by Wiener filter
  - **NSR** field: User-estimated noise-to-signal ratio. It is the approximation to the ratio between the noise power spectrum and image power spectrum as required by the definition of Wiener filter. Enabled when **Use Wiener filter** checkbox is checked.

- **Show MTF** checkbox: Check this box when you want to show the MTF of the imaging system. Whenever the Simulate image button is pressed, the MTF for the present imaging system configuration is computed and shown in a figure. New plots and old plots are shown together in the same figure.
- **Estimate blur** checkbox: Check this box when you want to estimate the image blurring in terms of number of pixels.

3. Press the **Simulate image** button to simulate the image. You can view the pixel information by placing the cursor over the Object and Image panels.
4. The output values are calculated using the given arguments. They are then shown in the *Output* panel. Let us explain them in order:
- **Object field width**: Width of object field in pixels
- **Object field height**: Height of object field in pixels
- **Ideal image distance from exit pupil**: Image distance calculated by the lens formula. This value may NOT be equal to the user-defined distance of the CCD sensor from the exit pupil
- **Magnification of in-focus image**: Linear magnification of in-focus image
- **f-number**: F-number, defined as (focal length)/(exit pupil diameter)
- **Numerical aperture (NA)**: Numerical aperture, defined as (exit pupil radius)/(distance of CCD sensor from exit pupil)
- **Field of view (FOV) in degrees panel**
  - **Horizontal**: Horizontal FOV in degree
  - **Vertical**: Vertical FOV in degree
  - **Diagonal**: Diagonal FOV in degree
- **Max. OPD due to defocus (in wavelengths) panel**
  - (If the light source type is *Monochromatic*) **Monochromatic light (x nm)**: Max. optical path difference (OPD) of monochromatic light with wavelength x nm due to defocus
  - (If the light source type is *Polychromatic (RGB)*) **Red light (675 nm)**: Max. OPD of red light (675 nm) due to defocus
  - (If the light source type is *Polychromatic (RGB)*) **Green light (525 nm)**: Max. OPD of green light (525 nm) due to defocus
  - (If the light source type is *Polychromatic (RGB)*) **Blue light (475 nm)**: Max. OPD of blue light (475 nm) due to defocus
- **Blurring (in no. of pixels) panel**
  - (If the light source type is *Monochromatic*) **Monochromatic light (x nm)**: Measures how blur the image of a monochromatic (x nm) point source is
  - (If the light source type is *Polychromatic (RGB)*) **Red light (675 nm)**: Measures how blur the image of a red (675 nm) point source is
  - (If the light source type is *Polychromatic (RGB)*) **Green light (525 nm)**: Measures how blur the image of a green (525 nm) point source is
  - (If the light source type is *Polychromatic (RGB)*) **Blue light (475 nm)**: Measures how blur the image of a blue (475 nm) point source is

5. Press the *Save image* button if you want to save the image. The image can be saved into *.bmp, *.gif, *.jpg, *.png and *.tif files.
6. Press the *Estimate blur* button if you want to estimate the image blur.
7. Press the *Close* button to close Virtual Camera.
4.2. Algorithm

The algorithm of Virtual Camera is mainly based on the theory of Fourier optics. Let us explain how this theory works in the software by viewing closely the most important function in the source code: `simImgButton_Callback(hObject, eventdata, handles)` (Please see Appendix for the code).

4.2.1. Derived parameters

The first part of the function is to collect all the user-defined information about the imaging system. Fig. 16 shows some of the user-defined parameters. The parameters that are to be calculated are followed by a question mark (?). The ideal image distance $z_i$ is calculated by the lens formula:

$$\frac{1}{z_o} + \frac{1}{z_i} = \frac{1}{f}. \quad (5)$$

Then the magnification of the in-focus image is

$$M = -\frac{z_i}{z_o}. \quad (6)$$

On the left hand side of Fig. 16, there is a region called object field. It is the region that can be viewed by the CCD sensor. By similar triangles, the width and height of object field are given by

$$w_f = \frac{w_{CCD}}{(z_{CCD}/z_o)}, \quad h_f = \frac{h_{CCD}}{(z_{CCD}/z_o)}, \quad (7)$$

where $w_{CCD}$ and $h_{CCD}$ are the width and height of CCD sensor array, $z_{CCD}$ is the user-defined distance between the CCD sensor and the exit pupil, $z_o$ is the distance between the object plane and the entrance pupil.
Fig. 16 Object, object field, CCD sensor and image.
The FOV of the imaging system is defined by the horizontal FOV \((FOV_h)\), vertical FOV \((FOV_v)\) and diagonal FOV \((FOV_d)\). By considering the right-angled triangles in Fig. 16, the field of views are given by

\[
FOV_h = 2 \left( \tan^{-1} \frac{w_{\text{CCD}}/2}{z_{\text{CCD}}} \right) \left( \frac{180}{\pi} \right) \text{ degrees},
\]

\[
FOV_v = 2 \left( \tan^{-1} \frac{h_{\text{CCD}}/2}{z_{\text{CCD}}} \right) \left( \frac{180}{\pi} \right) \text{ degrees},
\]

\[
FOV_d = 2 \left( \tan^{-1} \frac{\sqrt{w_{\text{CCD}}^2 + h_{\text{CCD}}^2}/2}{z_{\text{CCD}}} \right) \left( \frac{180}{\pi} \right) \text{ degrees}.
\]

(8)

4.2.2. Overlapping between object and object field

Next, we have to determine how the object and the object field overlap. This is necessary because the object may not be centered, and the object field may not be large enough to cover the whole object. We determine the overlapping by considering the corner coordinates of the overlapping region. Let the object corners be \((x_1, y_1)\), \((x_2, y_1)\), \((x_1, y_2)\), \((x_2, y_2)\) and the object field corners be \((x_3, y_3)\), \((x_4, y_3)\), \((x_3, y_4)\), \((x_4, y_4)\), where

\[
x_1 = -\frac{w_o}{2} + x_c, \quad x_2 = \frac{w_o}{2} + x_c, \quad x_3 = -\frac{w_f}{2}, \quad x_4 = \frac{w_f}{2}.
\]

\[
y_1 = -\frac{h_o}{2} + y_c, \quad y_2 = \frac{h_o}{2} + y_c, \quad y_3 = -\frac{h_f}{2}, \quad y_4 = \frac{h_f}{2}, \quad \text{and}
\]

\((x_c, y_c)\) are the object center coordinates. There are altogether 6 cases of overlapping for each coordinate. After that, we can find out the rows and columns of the object that should be extracted to form the cropped object. Let the object be a matrix with \(N_{yo}\) rows and \(N_{xo}\) columns. The extracted columns are shown at the bottom of each case in Fig. 17, and the extracted rows are shown in Fig. 18. Note that \(a:b\) means that the extraction ranges from row \(a\) (or column \(a\)) to row \(b\) (or column \(b\)). The first row or column has an index of 1 instead of 0, following the convention of MATLAB. The function \texttt{round()} rounds a floating point number to an integer.
Fig. 17 Determining the object columns to be extracted.
Case 1

\[ y_1 < y_2 \leq y_3 < y_4 \]

No overlap

Extracted rows \( N_{yo} \): 0

Case 2

\[ y_1 < y_3 < y_2 < y_4 \]

Overlap

\[ 1: \text{round} \left( \frac{y_2 - y_3}{h_o} N_{yo} \right) \]

Case 3

\[ y_3 < y_1 < y_2 < y_4 \]

Overlap

\[ 1: N_{yo} \]

Case 4

\[ y_3 < y_1 < y_4 < y_2 \]

Overlap

\[ N_{yo} - \text{round} \left( \frac{y_4 - y_1}{h_o} N_{yo} \right) + 1: N_{yo} \]

Case 5

\[ y_3 < y_4 \leq y_1 < y_2 \]

No overlap

Case 6

\[ y_1 < y_3 < y_4 < y_2 \]

Overlap

\[ \text{round} \left( \frac{y_2 - y_4}{h_o} N_{yo} \right) \]

\[ : N_{yo} - \text{round} \left( \frac{y_3 - y_1}{h_o} N_{yo} \right) + 1 \]

Fig. 18 Determining the object rows to be extracted.
4.2.3. Resizing and zero padding

After cropping the object, we have to consider how the cropped object is fit into the object field. In this software, the object field is represented by a matrix whose size is the same as that of the CCD array. This is to facilitate the Fast Fourier transform (FFT) operation for computing the image. Suppose the object field matrix has $N_{yf} \times N_{xf}$ rows and columns. In order to fit the $N_{yc} \times N_{xc}$ cropped object into the $N_{yf} \times N_{xf}$ object field correctly, the cropped object matrix must be resampled and resized to a size of $N_{yc} \times N_{xc}$'. At the same time, we have to determine the number of zeros that must be filled into the object field matrix. These zeros represent the dark background around the cropped object. Let $N_{right}$ and $N_{left}$ be the number of zeros that must be filled on the right side and left side of the cropped object respectively, and $N_{above}$ and $N_{below}$ be the number of zeros that must be padded above and below the cropped object respectively. We revisit the cases in Fig. 17 and Fig. 18. Since overlapping does not occur in Case 1 and Case 5, we only need to consider Cases 2, 3, 4 and 6. This is shown in Fig. 19 and Fig. 20.

The MATLAB function that performs resampling is resample($X$, $P$, $Q$). If $X$ is a matrix, it resamples the columns of $X$ at $P/Q$ times the original sampling rate. For example, to resample the columns of $X$ from length 512 to length 256, we should use resample($X$, 256, 512). To resample the whole matrix, we have to resample along the columns first, take the transpose of the resulting matrix, resample along the rows, and take the transpose again. For details, please refer to the code in Appendix.

4.2.4. Image generation

At this point onwards, we shall treat the object field as the actual object seen by the imaging system. To generate the image, we make use of Fourier optics. Here is a very brief account of the theory behind. Let $I_o(\xi, \eta)$ be the object intensity, $I_g(\xi, \eta)$ be the ideal image intensity and $I_i(u,v)$ be the actual image intensity. $I_g(\xi, \eta)$ is the geometrical-optics prediction of the image for a perfect imaging system, given by

$$I_g(\xi, \eta) = \frac{1}{M^2} I_o\left(\frac{\xi}{M}, \frac{\eta}{M}\right),$$

(9)

where $M$ is the linear magnification. This equation implies that the ideal image intensity is just the scaled version of the object intensity. The actual image is found by Fourier transforming the ideal image first, multiplying the spectrum by the optical transfer function (OTF) and finally inverse Fourier transforming the result. This is shown in the following equation:

$$I_i(u,v) = F^{-1}\{H(f_x, f_y)G_g(f_x, f_y)\}$$

(10)
Case 2

Resize the number of columns from $N_{xc}$ to $N_{xc}'$:

$$N_{xc}' = \text{round} \left( \frac{x_2 - x_3}{w_f} N_{sf} \right)$$

Zero padding:

$$N_{right} = N_{sf} - N_{xc}'$$

Case 3

$$x_1 < x_3 < x_2 < x_4$$

$$N_{xc}' = \text{round} \left( \frac{w_e}{w_f} N_{sf} \right)$$

Case 4

$$x_3 < x_1 < x_2 < x_4$$

$$N_{left} = \text{round} \left( \frac{x_1 - x_3}{w_f} N_{sf} \right)$$

$$N_{right} = N_{sf} - N_{xc}' - N_{left}$$

Case 6

$$x_1 < x_3 < x_4 < x_2$$

$$N_{xc}' = N_{sf}$$

$$N_{left} = N_{sf} - N_{xc}'$$

Fig. 19 Resizing and zero padding in the x-direction.
Fig. 20 Resizing and zero padding in the y-direction.
where $F^{-1}$ represents the inverse Fourier transform, $H(f_x, f_y)$ is the OTF and $G_g(f_x, f_y)$ is the Fourier spectrum of ideal image. Since OTF is completely determined by the characteristics of the imaging system and independent of the object and image, if we know the values of OTF, then we can compute the image for any object. OTF is given by

$$
H (f_x, f_y) = \frac{\iint P \left( x + \frac{\lambda z_i f_x}{2}, y + \frac{\lambda z_i f_y}{2} \right) P^* \left( x - \frac{\lambda z_i f_x}{2}, y - \frac{\lambda z_i f_y}{2} \right) dx dy}{\iint P (x, y) dx dy},
$$

where $P(x, y)$ is the pupil function, $\lambda$ is the wavelength of the light source and $z_i$ is the ideal image distance. Pupil function describes the amplitude and phase change encountered by an electromagnetic (EM) wave when it passes through the exit pupil. Since the OTF calculation is done in the computer, the pupil function has to be discretized. How should the pupil function be sampled so as to obtain the correct discrete OTF?

$$
\Delta \xi = \frac{w_o}{N_x}, \quad \Delta \eta = \frac{h_o}{N_y}.
$$

According to (5), the ideal image should have a width of $w_g = w_f |M|$ and a height of $h_g = h_f |M|$. The sampling periods are:

$$
\Delta \xi = \frac{w_g}{N_x} = \frac{w_f |M|}{N_x}, \quad \Delta \eta = \frac{h_g}{N_y} = \frac{h_f |M|}{N_y}.
$$

Fig. 21 Transforming object to ideal image.

We start by considering an object with actual dimensions $w_o \times h_o$. It is also a matrix of size $N_x$ by $N_y$. The sampling periods along the vertical and horizontal directions are
The ideal image is then Fourier-transformed into its spectrum. In digital implementation, the discrete Fourier transform (DFT) is used and it is performed by the fast Fourier transform (FFT) algorithm. This results in a spectrum in which the frequency components are spaced by \( \frac{1}{N_x \Delta \xi} = \frac{1}{w_x |M|} \) in the horizontal direction and \( \frac{1}{N_y \Delta \eta} = \frac{1}{h_y |M|} \) in the vertical direction (Fig. 22). According to (10), this spectrum is multiplied with the OTF to form the actual image spectrum. Therefore the frequency component spacing of the OTF and the actual image spectrum must be the same as the ideal image spectrum. The actual image spectrum is then converted to the actual image by taking inverse DFT (Fig. 23).
The frequency component spacing of the OTF constraints the sampling of the pupil function, because OTF and pupil function are closely related. Let’s look at Fig. 24. The OTF matrix $H$ that is to be multiplied by the ideal image spectrum is a submatrix of a larger OTF matrix $J$. $J$ has a size of $N_{yp}$ by $N_{xp}$. The reason for $N_{yp}$ and $N_{xp}$ will be explained later. In functional form, $H$ and $J$ are related by

$$ H(f_x, f_y) = J(\lambda z_i f_x, \lambda z_i f_y). \quad (14) $$

Thus, the horizontal and vertical spacings between two adjacent elements of $J$ are $\lambda z_i / (N_x \Delta f_x) = \lambda z_i / (w_f |M|)$ and $\lambda z_i / (N_y \Delta f_y) = \lambda z_i / (h_f |M|)$, respectively.

With regard to Eq. 14, $J$ is actually the autocorrelation of the pupil function $P$. By autocorrelation theorem, $J$ can be calculated by first taking Fourier transform on $P$, squaring the magnitude of the spectrum, and then taking inverse Fourier transform. That is to say,

$$ J(x, y) = F^{-1}\left\{\left|F\{P(x, y)\}\right|^2\right\} = F^{-1}\left\{|\tilde{P}(f, g)|\right\}, \quad (15) $$

where $\tilde{P}$ is the Fourier spectrum of $P$. Since the frequency component spacings for $J$ are known, we can trace backwards and find the sampling periods for $P$. The horizontal and vertical sampling periods for $P$ are found to be equal to the horizontal and vertical spacings between two adjacent elements of $J$, i.e. $\lambda z_i / (w_f |M|)$ and $\lambda z_i / (h_f |M|)$.

The numbers $N_{xp}$ and $N_{yp}$ are determined as follows. The actual pupil function is situated at the middle of matrix $P$. It is noted that in order to obtain the correct OTF, i.e. performing the autocorrelation correctly, the half-width (i.e. in actual dimension) of the matrix $P$ must be at least twice the pupil radius. It is illustrated in Fig. 25. This is translated to the following criterion:

$$ \frac{1}{2} \frac{N_{xp} \lambda z_i}{w_f |M|} \geq 2R \Rightarrow N_{xp} \geq \frac{4R w_f |M|}{\lambda z_i} \quad (16) $$

$$ \frac{1}{2} \frac{N_{yp} \lambda z_i}{h_f |M|} \geq 2R \Rightarrow N_{yp} \geq \frac{4R h_f |M|}{\lambda z_i} $$

These two lower bounds for $N_{xp}$ and $N_{yp}$ are implemented in the red lines of the code in Appendix.

Besides imposing lower bounds for $N_{xp}$ and $N_{yp}$, upper bounds are also imposed. The reason is that if both numbers were too large, the matrix $P$ would be so large that the DFT of $P$ becomes computationally infeasible. For example, the DFT of $P$ with a size of $2000 \times 2000$ could hang a computer up! Therefore in the program, $N_{xp}$ and $N_{yp}$ are limited to under 1500 (see the blue lines in the code).
Fig. 24 Sampling of pupil function $P$.
Fig. 25 To ensure correct autocorrelation of the pupil function, the half width of $P$ must be at least twice of the pupil radius $R$.

We have just gone through the core part of the function `simImgButton_Callback()`. It is also the main function of the whole software. It is hoped that by understanding this section, a developer can have a clear picture of how this software works and edit the source code according to his/her need. For the other functions, I am not going to explain in detail. Please refer to the comment in the source code for more explanations.

5. Conclusion

This research explores the concept of integrated computational imaging system by investigating the combined use of pupil phase masks and digital restoration filters to achieve extended depth of field and aberration invariance. A simulation software known as Virtual Camera is developed to simulate a spatially-invariant imaging system installed with phase masks. It is demonstrated that cubic phase masks can increase the depth of field significantly without decreasing the exit pupil size and light flux. To probe further, we seek for a phase mask that may perform better than a cubic mask and/or have higher manufacturability. However, the progress is still slow along this line. It is hoped that the future researchers can have a breakthrough in this problem.

This report also describes the usage and algorithm of Virtual Camera. This is served as a reference for researchers who are interested in modifying the software to suit his/her own use.
Appendix

% --- Executes on button press in simImgButton.
function simImgButton_Callback(hObject, eventdata, handles)
% hObject    handle to simImgButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Check errors in data fields before simulation
msg = CheckErrors(handles);
if ~strcmp(msg, '')
    msg = [sprintf('Error exists in the input arguments. No image is simulated.\n') msg];
    errordlg(msg, 'Bad input', 'modal')
    return
end
[obj.Ny obj.Nx numColors] = size(handles.obj.values);

% Extract information from various input text boxes
% Parameters for the light source
source.type = get(handles.sourceTypeEdit, 'String');          % Source type
if numColors == 1
    source.wavelength = str2double(get(handles.wavelengthEdit, 'String'))*1e-6;
else
    source.wavelength = [675e-6 525e-6 475e-6];               % Average wavelengths of
red, green and blue light
end

% Parameters for the object plane
obj.width = str2double(get(handles.woEdit, 'String'));        % Object width
obj.height = str2double(get(handles.hoEdit, 'String'));       % Object height
obj.zo = str2double(get(handles.zoEdit, 'String'));           % Object distance from
entrance pupil
obj.xcen = str2double(get(handles.xcenEdit, 'String'));       % X-coordinate of object
center
obj.ycen = str2double(get(handles.ycenEdit, 'String'));       % Y-coordinate of object
center

% Parameters for the imaging system, including the pupil
lens.f = str2double(get(handles.fEdit, 'String'));            % Focal length
pupil.radius = str2double(get(handles.radiusEdit, 'String')); % Pupil radius
val = get(handles.pupilTypePopup, 'Value');                   % Pupil type
list = get(handles.pupilTypePopup, 'String');
pupil.type = list(val);
if strcmp(pupil.type, 'Annular')
pupil.rratio = str2double(get(handles.pupilP1Edit, 'String')); % Ratio of radii
end
if strcmp(pupil.type, 'Cubic') || strcmp(pupil.type, 'Triangular prism') ...
    || strcmp(pupil.type, 'Cone')
pupil.n = str2double(get(handles.pupilP1Edit, 'String'));     % Refractive index
pupil.maxt = str2double(get(handles.pupilP2Edit, 'String'));   % Maximum thickness
end
if strcmp(pupil.type, 'Quartic')
pupil.alpha = str2double(get(handles.pupilP1Edit, 'String')); % Alpha value
end
if strcmp(pupil.type, 'Logarithmic')
pupil.beta0 = str2double(get(handles.pupilP1Edit, 'String')); % Beta0 value
pupil.beta = str2double(get(handles.pupilP2Edit, 'String'));   % Beta value
end
pupil.aber = handles.aber;

% Parameters for the image plane, including the CCD sensor array
img.zccd = str2double(get(handles.zccdEdit, 'String'));
img.boundaries = cell(1, numColors);
img.Nblur = zeros(1, numColors);
sensor.wp = str2double(get(handles.wpEdit, 'String'))*1e-3; % Pixel width
sensor.hp = str2double(get(handles.hpEdit, 'String'))*1e-3; % Pixel height
sensor.wccd = str2double(get(handles.wccdEdit, 'String'));  % CCD array width
sensor.hccd = str2double(get(handles.hccdEdit, 'String'));  % CCD array height
sensor.diagccd = sqrt(sensor.wccd.^2 + sensor.hccd.^2);      % CCD array diagonal length
sensor.Nx = round(sensor.wccd/sensor.wp);                    % No. of pixels in x-direction
sensor.Ny = round(sensor.hccd/sensor.hp);                    % No. of pixels in y-direction

% Parameters for the noise
val = get(handles.noiseTypePopup, 'Value');                  % Noise type
list = get(handles.noiseTypePopup, 'String');
noise.type = list{val};
switch noise.type
    case 'None'
    case 'Gaussian'
        noise.mean = str2double(get(handles.noiseP1Edit, 'String'))./256;
        noise.var = (str2double(get(handles.noiseP2Edit, 'String'))./256).^2;
    case 'Poisson'
    case 'Salt & pepper'
        noise.density = str2double(get(handles.noiseP1Edit, 'String'))./100;
    case 'Speckle'
        noise.var = str2double(get(handles.noiseP1Edit, 'String'));
    otherwise
end

% Begin simulation
img.zi = obj.zo.*lens.f./(obj.zo - lens.f);
if img.zi <= 0
    % Since the image captured by an imaging system must be real, a
    % warning is issued when the image is virtual.
    errordlg(sprintf(['The image is virtual. No real image is formed on the screen
' ... 'Please change the object distance and/or image distance and/or focal
length']), ...
    'Warning', 'modal')
end
mag = -img.zi/obj.zo;
pupil.NA = pupil.radius/img.zccd;
pupil.opdMax = 1/2*(1/img.zi - 1/img.zccd)*pupil.radius.^2;
img.width = obj.width*abs(mag);
img.height = obj.height*abs(mag);

% Determine how the object field (the rectangular region seen by the
% imaging system on the object plane) and the object overlap
% The overlapping depends on the coordinates of the object and object field
% boundaries
obj.wf = sensor.wccd/(img.zccd/obj.zo);  % Field width
obj.hf = sensor.hccd/(img.zccd/obj.zo);  % Field height
xbounds = [-obj.width/2+obj.xcen obj.width/2+obj.xcen -obj.wf/2 obj.wf/2];  % X-coordinates of the boundaries of object and field
ybounds = [-obj.height/2+obj.ycen obj.height/2+obj.ycen -obj.hf/2 obj.hf/2];  % Y-coordinates of the boundaries of object and field
[xboundsSorted xorder] = sort(xbounds);
[yboundsSorted yorder] = sort(ybounds);

% padsize: Specify the number of zeros padded around the object in
% order to fill up the whole object field
% Format: [left right above below]
padsize = [0 0 0 0];
rowBegin = 0; rowEnd = 0; colBegin = 0; colEnd = 0;
if ~any(xorder - [1 2 3 4])
    % No intersection
elseif ~any(xorder - [1 3 2 4])
    colBegin = obj.Nx - round((xbounds(2)-xbounds(3))/obj.width*obj.Nx) + 1;
    colEnd = obj.Nx;
    Nxc = round((xbounds(2)-xbounds(3))/obj.wf*sensor.Nx);
    padsize = padsize + [0 sensor.Nx-Nxc 0 0];
elseif ~any(xorder - [3 1 2 4])
    colBegin = 1;
    colEnd = obj.Nx;
Nxc = round(obj.width/obj.wf*sensor.Nx);
Nleft = round((xbounds(1)-xbounds(3))/obj.wf*sensor.Nx);
Nright = sensor.Nx - Nxc - Nleft;
padsize = padsize + [Nleft Nright 0 0];
elseif ~any(xorder - [3 1 4 2])
colBegin = 1;
colEnd = round((xbounds(4)-xbounds(1))/obj.width*obj.Nx);
Nxc = round((xbounds(4)-xbounds(1))/obj.wf*sensor.Nx);
padsize = padsize + [sensor.Nx-Nxc 0 0 0];
elseif ~any(xorder - [3 4 1 2])
% No intersection
elseif ~any(xorder - [1 3 4 2])
colBegin = round((xbounds(3)-xbounds(1))/obj.width*obj.Nx);
colEnd = obj.Nx - round((xbounds(2)-xbounds(4))/obj.width*obj.Nx) + 1;
Nxc = sensor.Nx;
else
errordlg('Wrong overlapping in x-direction');
end

if ~any(yorder - [1 2 3 4])
% No intersection
elseif ~any(yorder - [1 3 2 4])
rowBegin = 1;
rowEnd = round((ybounds(2)-ybounds(3))/obj.height*obj.Ny);
Nyc = round((ybounds(2)-ybounds(3))/obj.hf*sensor.Ny);
padsize = padsize + [0 0 sensor.Ny-Nyc 0];
elseif ~any(yorder - [3 1 2 4])
rowBegin = 1;
rowEnd = obj.Ny;
Nyc = round(obj.height/obj.hf*sensor.Ny);
Nbelow = round((ybounds(1)-ybounds(3))/obj.hf*sensor.Ny);
Nabove = sensor.Ny - Nyc - Nbelow;
padsize = padsize + [0 0 0 Nabove Nbelow];
elseif ~any(yorder - [3 4 1 2])
rowBegin = obj.Ny - round((ybounds(4)-ybounds(1))/obj.height*obj.Ny) + 1;
rowEnd = obj.Ny;
Nyc = round((ybounds(4)-ybounds(1))/obj.hf*sensor.Ny);
padsize = padsize + [0 0 0 sensor.Ny-Nyc];
elseif ~any(yorder - [3 1 4 2])
rowBegin = round((ybounds(2)-ybounds(4))/obj.height*obj.Ny);
rowEnd = obj.Ny - round((ybounds(3)-ybounds(1))/obj.height*obj.Ny) + 1;
Nyc = sensor.Ny;
else
errordlg('Wrong overlapping in y-direction');
end

if all([rowBegin, rowEnd, colBegin, colEnd])
for m = 1:numColors
    croppedObj = resample(handles.obj.values(rowBegin:rowEnd, colBegin:colEnd, m), Nyc, rowEnd-rowBegin+1);
    croppedObj = croppedObj';
    croppedObj = resample(croppedObj, Nxc, colEnd-colBegin+1);
    croppedObj = croppedObj';
    % Pad zeros around the cropped object to represent the dark region in the field
    obj.values(:,:,m) = PadZero(croppedObj, padsize);
end

% Calculate the number of samples for the pupil function in order
% to obtain the correct OTF. If the number of samples is too large,
% then a warning is issued.
reqNx = ceil(4*obj.wf*abs(mag)*pupil.radius/(min(source.wavelength)*img.zi));
reqNy = ceil(4*obj.hf*abs(mag)*pupil.radius/(min(source.wavelength)*img.zi));
pupil.Nx = max(reqNx, sensor.Nx);
pupil.Ny = max(reqNy, sensor.Ny);
if pupil.Nx > 1500 && pupil.Ny > 1500
    msg = sprintf('The number of samples for the pupil function is too large.
');
else
    obj.values = zeros(sensor.Ny, sensor.Nx, numColors);
end

if all([rowBegin, rowEnd, colBegin, colEnd])
for m = 1:numColors
    croppedObj = resample(handles.obj.values(rowBegin:rowEnd, colBegin:colEnd, m), Nyc, rowEnd-rowBegin+1);
    croppedObj = croppedObj';
    croppedObj = resample(croppedObj, Nxc, colEnd-colBegin+1);
    croppedObj = croppedObj';
    % Pad zeros around the cropped object to represent the dark region in the field
    obj.values(:,:,m) = PadZero(croppedObj, padsize);
end

% Crop the object if only part of the object overlaps with the field
% obj.values = zeros(sensor.Ny, sensor.Nx, numColors);
if all([rowBegin, rowEnd, colBegin, colEnd])
for m = 1:numColors
    croppedObj = resample(handles.obj.values(rowBegin:rowEnd, colBegin:colEnd, m), Nyc, rowEnd-rowBegin+1);
    croppedObj = croppedObj';
    croppedObj = resample(croppedObj, Nxc, colEnd-colBegin+1);
    croppedObj = croppedObj';
    % Pad zeros around the cropped object to represent the dark region in the field
    obj.values(:,:,m) = PadZero(croppedObj, padsize);
end

% Calculate the number of samples for the pupil function in order
% to obtain the correct OTF. If the number of samples is too large,
% then a warning is issued.
reqNx = ceil(4*obj.wf*abs(mag)*pupil.radius/(min(source.wavelength)*img.zi));
reqNy = ceil(4*obj.hf*abs(mag)*pupil.radius/(min(source.wavelength)*img.zi));
pupil.Nx = max(reqNx, sensor.Nx);
pupil.Ny = max(reqNy, sensor.Ny);
if pupil.Nx > 1500 && pupil.Ny > 1500
    msg = sprintf('The number of samples for the pupil function is too large.
');
'The computer may run out of memory. Please either:
1. decrease the object width
2. decrease the exit pupil radius
3. increase the wavelength
4. increase the object distance'}
% Show MTF if the "Show MTF" checkbox is checked
if (get(handles.showMTFChkbox, 'Value') == get(handles.showMTFChkbox, 'Max'))
    mtf(:,:,m) = abs(otf(1, 1:round(pupil.Nx/2), m));
    dfr = 1/(sensor.Nx*sensor.wp);
    fr = 0:dfr:(pupil.Nx - 1)/2*dfr;
    figure(handles.mtfFig(m));
    set(handles.mtfFig(m), 'Visible', 'on', 'Name', 'Modulation Transfer Function');
    hold all, plot(fr, mtf(:,:,m)), hold off;
    xlabel('Radial frequency coordinate (mm^{-1})')
    ylabel('Normalized magnitude')
    title(['MTF of imaging system ' '\la' \num2str(source.wavelength(m)*1e6) ... ' nm, phase mask: ' pupil.type'])
end

% Estimate image blurring if the "Estimate blur" checkbox is checked
if (get(handles.estBlurChkbox, 'Value') == get(handles.estBlurChkbox, 'Max'))
    % Create a point object whose size is the same as the original object
    pointObj = zeros(size(obj.values(:,:,m)));
    center = [round(size(pointObj,1)/2) round(size(pointObj,2)/2)];
    pointObj(center(1), center(2)) = 255;
    objSpec = fft2(pointObj);
    croppedOtf = [otf(1:ceil(sensor.Ny/2), 1:ceil(sensor.Nx/2), m) ... 
                  otf(1:ceil(sensor.Ny/2), pupil.Nx-a:pupil.Nx, m); ... 
    imgSpec = objSpec.*croppedOtf;
    blurredImg = abs(ifft2(imgSpec));
    if any(any(blurredImg))
        blurredImg = Normalize(blurredImg);
    end

    % Pixels where intensity > 0.1 are counted as image pixels
    % 0.1 is only set arbitrarily
    binImg = (blurredImg > 0.1);
    img.Nblur(m) = sum(sum(binImg));
    img.boundaries{m} = bwboundaries(binImg, 'noholes'); % Find the boundary of blurred image
end

% Show image
imagesc(uint8(round(handles.img.values.*256)), 'Parent', handles.imgAxes);
set(handles.imgAxes, 'Visible', 'off');
if (get(handles.estBlurChkbox, 'Value') == get(handles.estBlurChkbox, 'Max'))
    set(handles.imgAxes, 'NextPlot', 'add');
    for m = 1:numColors
        for k = 1:length(img.boundaries{m})
            boundCoors = img.boundaries{m}{k};
            switch m
                case 1
                    plot(boundCoors(:,2), boundCoors(:,1), 'r', 'Parent', handles.imgAxes);
                case 2
                    plot(boundCoors(:,2), boundCoors(:,1), 'g', 'Parent', handles.imgAxes);
                case 3
                    plot(boundCoors(:,2), boundCoors(:,1), 'b', 'Parent', handles.imgAxes);
            end
        end
    end
    set(handles.imgAxes, 'Visible', 'off', 'NextPlot', 'replace');
else
    set(handles.imgAxes, 'Visible', 'off', 'NextPlot', 'revert');
end
if (strcmp(source.type, 'Monochromatic'))
    set(handles.blurRedText, 'String', sprintf('Monochromatic light (%s nm):', ... 
              get(handles.wavelengthEdit, 'String')));
    set(handles.blurRedEdit, 'String', num2str(img.Nblur));
else
    set(handles.blurRedEdit, 'String', num2str(img.Nblur(1)));
```matlab
set(handles.blurGreenEdit, 'String', num2str(img.Nblur(2)));
set(handles.blurBlueEdit, 'String', num2str(img.Nblur(3)));
end
set(handles.clearMarkingsButton, 'Enable', 'on');
end

% Output all numbers
set(handles.wfEdit, 'String', num2str(obj.wf)); % Object field width
set(handles.hfEdit, 'String', num2str(obj.hf)); % Object field height
set(handles.ziEdit, 'String', num2str(img.zi)); % Ideal image distance
set(handles.magEdit, 'String', num2str(mag)); % Magnification
set(handles.numberEdit, 'String', num2str(lens.f/(2*pupil.radius)));
% F-number
set(handles.NAEdit, 'String', num2str(pupil.NA)); % NA
set(handles.horiFOVEdit, 'String', num2str(round(2*atan2(sensor.wccd/2, img.zccd)*180/pi)));
% Horizontal FOV
set(handles.vertFOVEdit, 'String', num2str(round(2*atan2(sensor.hccd/2, img.zccd)*180/pi)));
% Vertical FOV
set(handles.diagFOVEdit, 'String', num2str(round(2*atan2(sensor.diagccd/2, img.zccd)*180/pi)));
% Diagonal FOV

normWm = pupil.opdMax./source.wavelength; % Normalize max. OPDs by wavelengths
if (strcmp(source.type, 'Monochromatic'))
    set(handles.wmRedText, 'String', sprintf('Monochromatic
light (%s nm):', ...
        get(handles.wavelengthEdit, 'String')));
    set(handles.wmRedEdit, 'String', num2str(normWm)); % Max. OPD for red light
else
    set(handles.wmRedEdit, 'String', num2str(normWm(1))); % Max. OPD for red light
    set(handles.wmGreenEdit, 'String', num2str(normWm(2))); % Max. OPD for green light
    set(handles.wmBlueEdit, 'String', num2str(normWm(3))); % Max. OPD for blue light
end

% Enable the other buttons
set(handles.clearImgButton, 'Enable', 'on');
set(handles.saveImgButton, 'Enable', 'on');

% Create the pixel information tool for the object
handles.objInfo = impixelinfo(handles.objAxes);
set(handles.objInfo, 'Units', 'pixels');
objInfoPos = get(handles.objInfo, 'Position');
opUnits = get(handles.objPanel, 'Units');
set(handles.objPanel, 'Units', 'pixels');
opPos = get(handles.objPanel, 'Position');
objInfoPos(1) = opPos(1)+2;
objInfoPos(2) = opPos(2)+2;
set(handles.objInfo, 'Position', objInfoPos);
set(handles.objPanel, 'Units', opUnits);

% Create the pixel information tool for the image
handles.imgInfo = impixelinfo(handles.imgAxes);
set(handles.imgInfo, 'Units', 'pixels');
imgInfoPos = get(handles.imgInfo, 'Position');
ipUnits = get(handles.imgPanel, 'Units');
set(handles.imgPanel, 'Units', 'pixels');
ipPos = get(handles.imgPanel, 'Position');
imgInfoPos(1) = ipPos(1)+2;
imgInfoPos(2) = ipPos(2)+2;
set(handles.imgInfo, 'Position', imgInfoPos);
set(handles.imgInfo, 'Units', 'normalized');
set(handles.imgPanel, 'Units', 'normalized');

% Update handles
guidata(hObject, handles);
```
Reference


