N-Dimensional Phase Unwrapping

Release 0.00

Davis M. Vigneault,[1,2,3] Wen-Tung Wang, PhD,[2] Michael Tee,[1,2] David A. Bluemke, MD, PhD,[2] and J. Alison Noble, PhD[1]

May 15, 2015

[1]Institute of Biomedical Engineering, Department of Engineering Science, University of Oxford, Oxford, United Kingdom.

[2]Radiology and Imaging Sciences, Clinical Center, National Institutes of Health, Bethesda, MD, United States.

[3]Sackler School of Graduate Biomedical Sciences, Tufts University School of Medicine, Boston, MA, United States.

This work was supported by the National Institutes of Health Intramural Research Program, and the National Institutes of Health-Oxford Scholars Program.

Abstract

Although phase data can take on any value, it is generally only possible to measure phase as a *principle* value, i.e., wrapped within the range $(-\pi,\pi]$. Determining the unwrapped phase from its principle value is a topic of considerable interest in magnetic resonance imaging (MRI), as well as many non-medical disciplines. Despite their importance in image analysis, filters for manipulating phase information have not been incorporated into ITK. This article introduces the ITKPhase module, containing filters useful for understanding, analyzing, and unwrapping n-dimensional phase data, and also serves as a practical introduction to phase unwrapping.

Contents

1 Background

1.1 Theory

Data collected via magnetic resonance (MR) imaging is inherently complex-valued, containing both real and imaginary (or equivalently, phase and magnitude) components (Figure 1). Phase data is useful in a variety of applications, such as harmonic phase (HARP) analysis of tagged MR [11], susceptibility weighted imaging (SWI) [10], and phase contrast angiography/venography . Although phase at a given pixel can generally take on any value, it is impossible to distinguish between true (ϕ) and principal ($\hat{\phi}$) phase values separated by an arbitrary multiple k of 2π, such that the measured phase value is 'wrapped' within the range $(-\pi, \pi]$. In theory, true phase may be recovered from principle phase by adding an integer multiple k of 2π, in a process known as 'unwrapping' (Equation 1).

$$\phi = \hat{\phi} + k2\pi \tag{1}$$

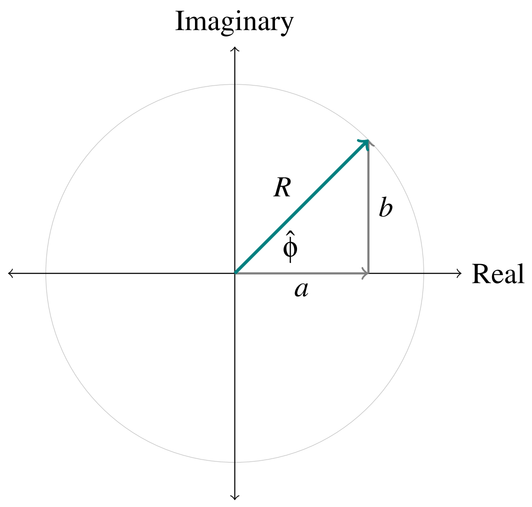


Figure 1: The Complex Plane. Here a complex number, z, is represented as a point on the complex plane. z can be represented equivalently in cartesian coordinates as the sum of a real and an imaginary number ($z = a + bi$) or in polar coordinates as a magnitude phase pair ($[R, \phi]$).

The reverse operation, by which principle phase is obtained from its true phase, is denoted by the wrapping operator W (Equation 2), which can be practically implimented with the four-quadrant arctangent function (`std::atan2(x,y)`).

$$\hat{\phi} = \arctan(\sin(\phi), \cos(\phi)) \equiv W\{\phi\} \tag{2}$$

Itoh [9] observed that the locally 'corrected' phase gradient (i.e., the phase gradient after appropriate addition of $k2\pi$ such that the difference between adjacent pixels is in the range $(-\pi, \pi]$) may be written in terms of the wrapping operator (Equation 3).[1]

[1] i is used in this submission both to refer to the mathematical constant meaning $\sqrt{-1}$, as previously, and, in this context, to pixel index along an arbitrary dimension ($0 < i < M - 1$, where M is the number of pixels along that dimension). The difference in usage should be clear from context. Note also that the above definition only applies when both i and $i - 1$ are completely within the image; otherwise, $\Delta\phi_i \equiv 0$.

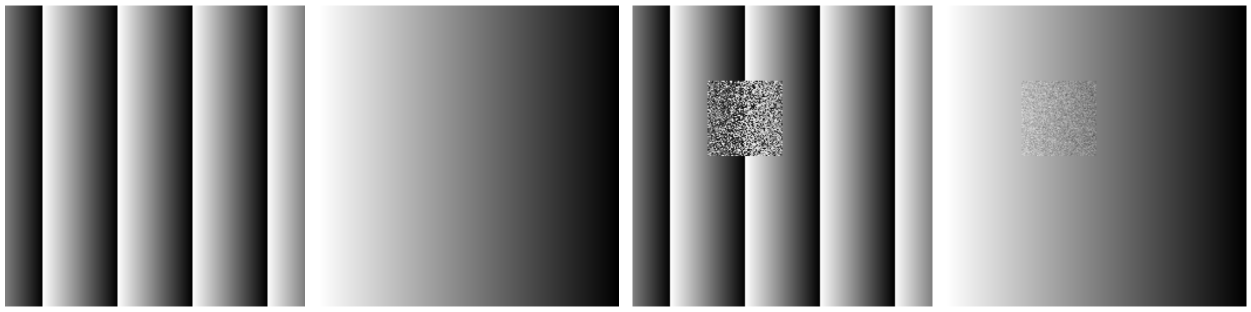


Figure 2: Simulated phase images, created by the `itk::PhaseExamplesImageSource` class. Wrapped phase ramp (2a); wrapped phase ramp with noise patch (2b); unwrapped phase ramp (2c); unwrapped phase ramp with noise patch (2d).

$$\Delta\phi_i \equiv W\left\{\hat{\phi}_i - \phi_{i-1}\right\} \tag{3}$$

A target pixel (ϕ_i) may be 'unwrapped' relative to an adjacent reference pixel (ϕ_{i-1}) such that their difference is in the range $(-\pi,\pi]$. This operation can be written in terms of the wrapped phase gradient (Equation 4).

$$\phi_i = \phi_{i-1} + \Delta\phi_i \tag{4}$$

1.2 Introduction to the ITKPhase Module

The presented module provides the unary wrap operator as a functor, `itk::Functor::WrapPhaseFunctor` (defined in `itkWrapPhaseFunctor.h`). The following snippet demonstrates the basic use of the class.[2]

```
itk::Functor::WrapPhaseFunctor< double > wrapFunc;

std::cout << wrapFunc( 3 ) << std::endl; // 3
std::cout << wrapFunc( 0 ) << std::endl; // 0
std::cout << wrapFunc( -3 ) << std::endl; // -3
std::cout << wrapFunc( 1 + vnl_math::pi ) << std::endl; // -2.14159
std::cout << wrapFunc( -vnl_math::pi - 1 ) << std::endl; // 2.14159
```

`itk::PhaseImageToImageFilter` (defined in `itkPhaseImageToImageFilter.h`) inherits from `itk::ImageToImageFilter` and serves as the base class for most classes in this module.

This class provides two methods, `Wrap(pixel)` and `Unwrap(target,relativeToReference)`. The first takes one argument and provides an interface to `WrapPhaseFunctor` (see Equation 2). The second makes use of the first to unwrap one pixel relative to another (see Equation 4).

For convenience, the ITKPhase module also includes `itk::PhaseExamplesImageSource` (defined in `itkPhaseExamplesImageSource.h`), which provides simple simulated phase examples for demonstrating the functionality of the module's classes.

[2]The constant `vnl_math::pi` is defined in the `vnl/vnl_math.h` header file. `std::cout` and `std::endl` are defined in the `<iostream>` header file.

By default, the class outputs a simple, wrapped phase ramp (Figure 2, far left).[3]

```
typedef itk::PhaseExamplesImageSource< WorkImageType > ExampleType;
ExampleType::Pointer phase = ExampleType::New();
phase->Update();
```

A patch of additive gaussian noise can be added by calling `phase->SetNoise(true)` (Figure 2, center right), and unwrapped versions of these images can be obtained by calling `phase->SetWrap(false)` (Figure 2, center left and far right, respectively). Though not demonstrated in this submission, the mean, standard deviation, and seed of the noise can be manually set using the `SetNoiseMean()`, `SetNoiseSD()`, and `SetNoiseSeed()` methods.

[3]For visualization purposes, all images have been rescaled prior to writing to png.

2 Phase Quality and Path-Dependent Unwrapping

2.1 The Itoh Phase Unwrapping Algorithm

`itk::ItohPhaseUnwrappingImageFilter` (defined in `itkItohPhaseUnwrappingImageFilter.h`) implements the simplest possible phase unwrapping algorithm, proposed by Itoh [9]. In this algorithm, the image is traversed linearly, once per image dimension, unwrapping each pixel relative to the previous pixel. In the case of the uncorrupted, wrapped phase ramp (Figure 2, far left), the correct unwrapped image (Figure 2, center left) is obtained with one traversal along the horizontal (x) direction (identical to input, not shown). Additionally traversing along the vertical (y) direction has no effect. This is true regardless of which dimension is traversed first.

However, in the case of the noise-corrupted phase (Figure 3), two notable observations should be made. First, regardless of which direction is traversed first, the original phase image is not recovered. Rather, the noisy region causes streaks to form 'downstream' of the corrupted data. Therefore, the Itoh algorithm is inadequate for unwrapping of this (and virtually all real world) data sets. Second, vertical followed by horizontal traversal does not produce the same image as the reverse. This suggests that, whereas the unwrapping of some images (e.g., the uncorrupted phase ramp) is path-independent, the unwrapping of other images (e.g., the noisy-corrupted phase) is path-dependent.

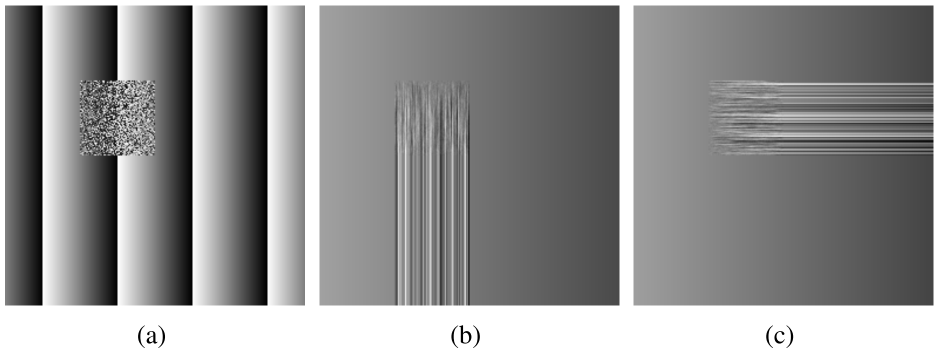


Figure 3: Noise-corrupted phase example ramps, unwrapped via the Itoh algorithm (`itk::ItohPhaseUnwrappingImageFilter`) in the horizontal then vertical directions (3b) and in the vertical then horizontal directions (3c). The wrapped image (3a) is reproduced for comparison.

2.2 Phase Residues

In order to predict when path-independent phase unwrapping is possible, it is necessary to introduce the concept of phase residues. A residue charge is calculated for a 2x2 neighborhood of pixels and a given pair of dimensions, by summing the wrapped phase differences across a clockwise trajectory through the neighborhood and dividing the result by 2π.[4] Effectively, this gives the number of positive discontinuities encountered less the number of negative discontinuities encountered (positive meaning that 2π would be added in order to remove it).

It has been proven that all phase residues have a value in the set $\{-1,0,1\}$ [2]. Take the following minimal

[4]We here use the clockwise convention, as presented by Goldstein, et al [7]. However, as it is only the relative sign that is important for phase unwrapping, the clockwise/counter-clockwise convention makes no practical difference, so long as consistency is maintained.

example (Figure 4):[5]

$$
\begin{array}{ccccccc}
0.0 & \rightarrow & 0.3 & \rightarrow & 0.3 & \rightarrow & 0.0 \\
\uparrow & \odot & \downarrow\uparrow & & \downarrow\uparrow & \otimes & \downarrow \\
0.0 & \leftarrow & -0.3 & \leftarrow & -0.3 & \leftarrow & 0.0
\end{array}
$$

Figure 4: Phase Residues. Phase residues are calculated by summing the wrapped phase differences in a clockwise trajectory through a 2x2 neighborhood. A positive residue ($\odot$) is noted in the leftmost neighborhood, a negative residue ($\otimes$) in the rightmost neighborhood, and no residue in the central neighborhood.

The presence of one or more phase residues in a phase image denote path-dependent unwrapping. In the above example, unwrapping along $(0,0) \rightarrow (1,0) \rightarrow (1,1)$ produces a value of 0.7 for pixel $(1,1)$ $(-0.3+1.0=0.7)$, whereas unwrapping from $(0,0) \rightarrow (0,1) \rightarrow (1,1)$ produces a value of -0.3 because no phase wraps are encountered.[6] Another useful result is that the wrapped line integral around a larger neighborhood yields the sum of the phase residues contained within that neighborhood. In this example, the line integral is 0, which rightly reflects the presence of a single positive and a single negative residue. Goldstein, et al [7], were the first to appreciate the significance of phase residues for path-dependency of phase unwrapping, by introducing the concept of residue 'balancing' for 2D phase unwrapping. They observed that, by defining branch cuts across which phase unwrapping was disallowed (either between residues of opposing sign or from a residue to the edge of the image), a subset of paths could be defined for which the wrapped line integral was zero (i.e., a subset of paths, none of which enclose an unbalanced residue). Within this subset, any path taken will arrive at an identical unwrapped solution (to an arbitrary additive multiple of 2π).

Appropriately balancing residues will ensure consistent, but not necessarily correct results [2]. All possible paths within a specific configuration yield the same result, to an additive multiple of 2π. However, results are not necessarily consistent between branch-cut configurations. Figure 5 illustrates this point. Two possible branch cut configurations are shown, each with one possible phase unwrapping path. It is noteworthy that the values of pixels $(1,1)$ and $(2,1)$ differ between the two configurations.

$$
\begin{array}{ccccccc}
0.0 & \rightarrow & 0.3 & \rightarrow & 0.3 & \rightarrow & 0.0 \\
 & \odot & — & & — & \otimes & \downarrow \\
0.0 & \leftarrow & -0.3 & \leftarrow & -0.3 & \leftarrow & 0.0
\end{array}
$$

(a)

$$
\begin{array}{ccccccc}
0.0 & \rightarrow & 0.3 & & 0.3 & \rightarrow & 0.0 \\
\uparrow & \odot & \downarrow & & \uparrow & \otimes & \downarrow \\
0.0 & | & 0.7 & \rightarrow & 0.7 & | & 0.0
\end{array}
$$

(b)

Figure 5: Branch Cut Configurations. The above example demonstrates two possible branch cut configurations. Configuration (a) connects residues of opposing sign to one another. Configuration (b) connects each residue to the image border. For each configuration, one possible unwrapping path is demonstrated. '—' and '|' represent vertical and horizontal branch cuts, respectively; arrows represent the path of phase unwrapping.

It should also be noted that the branch cutting theory so far discussed in this section is specific to two dimensional phase unwrapping. An extension of the branch cutting concept to three dimensions has been

[5]For simplicity of calculation, the phase has been divided by 2π such that the range is from $(-0.5, 0.5]$ and the phase residues do not need to be rescaled.

[6]In accordance with ITK convention, the origin is taken to be the upper left corner of the image, and zero-indexing is used.

proposed [8], which requires phase residues to be calculated with respect to each pair of dimensions (xy, xz, yz). To our knowledge, the branch cutting method has not been extended to higher dimensions. We have included a discussion of residues for completeness, and because we believe that the concepts are essential to the understanding of path dependence. However, as our interest in this submission is with phase unwrapping filters that may be implimented over images of arbitrary dimension, no branch cutting algorithms have been included.

Phase residues may be calculated using the `itk::PhaseResidueImageFilter` class (defined in `itkPhaseResidueImageFilter.h` (Figure 6). The filter inherits from `itk::PhaseImageToImageFilter`, and uses the `Wrap()` method to calculate phase residues as described above. Note that, due to the theoretical limitations described above, the filter only has a clear physical significance for 2D images.

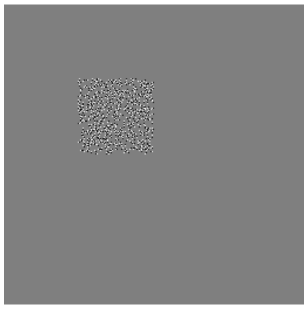


Figure 6: Phase residues are calculated for the noise-corrupted phase ramp example using the `itk::PhaseResidueImageFilter` class.

2.3 The Helmholtz Decomposition

Given a vector field v, if there exists some A such that $v = \nabla \times A$, then A is called the 'vector potential' of v. Given that the divergence of the curl of any vector field is the zero vector, v must therefore be entirely rotational (i.e., non-conservative):

$$\nabla \cdot v = \nabla \cdot (\nabla \times A) = 0 \tag{5}$$

Likewise, if there exists some ψ such that $v = \nabla\psi$, then ψ is called the 'scalar potential' of v. Given that the curl of the gradient of any scalar field is zero, v must therefore be entirely irrotational (i.e., conservative):

$$\nabla \times v = \nabla \times (\nabla\psi) = 0 \tag{6}$$

In the general case, A and ψ cannot be found (that is, the general case of a vector field need not be entirely rotational or irrotational). However, the Helmholtz Decomposition states that a vector field can be decomposed into rotational and irrotational components as the sum of scalar and vector potentials:

$$v = -\nabla\psi + \nabla \times A \tag{7}$$

In the context of phase unwrapping, the Helmholtz decomposition has specific implications for path dependence. In particular, the 'path-dependent' contribution is contained entirely within the rotational component, and the 'path-independent' contribution within the irrotational component. The irrotational and rotational components of the noise-corrupted phase ramp are shown in the left and right panels of Figure 7. Note in the

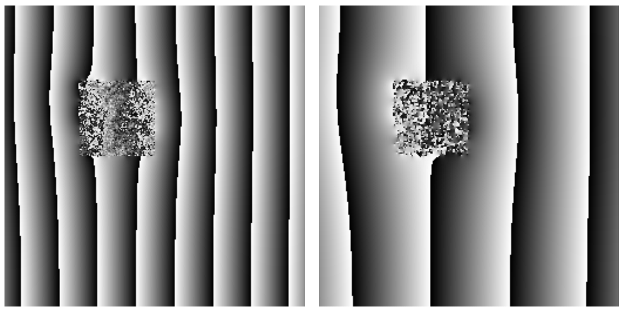


Figure 7: The `itk:HelmholtzDecompositionImageFilter` classe, applied to the noise-corrupted phase ramp example. Shown are the irrotational (curl-free) component (7a), and the right panel is the rotational (divergence-free) component (7b).

irrotational component that corruption of the original phase extends beyond the patch of noise itself, though there are no ambiguities.

The Helmholtz Decomposition is provided in the `itk::HelmholtzDecompositionImageFilter` class (defined in `itkHelmholtzDecompositionImageFilter.h`), which inherits from `itk::PhaseImageToImagefilter`. The irrotational and rotational components may be retrieved through the `GetRotational()` and `GetIrrotational()` methods, respectively.

The implementation of `itk::HelmholtzDecompositionImageFilter` takes advantage of the fact that the irrotational component is equivalent to the unweighted, least squares solution (`itk::DCTPhaseUnwrappingImageFilter`, discussed in a later section), re-wrapped into the range $(-\pi,\pi]$. The rotational component may then be obtained by subtracting the irrotational component from the original wrapped phase, and re-wrapping the result.

2.4 Phase Derivative Variance

It addition to phase residues, it is also useful to consider continuous measures of phase quality, of which the phase derivative variance is the most commonly used. Phase derivative variance σ in a single dimension d is the sum of squared differences of the local wrapped phase gradients ($\Delta\phi_i$) and the mean wrapped phase gradient ($\overline{\Delta\phi}$, Equation 8).

$$\sigma_d = \sum_i \left(\Delta\phi_i - \overline{\Delta\phi}\right)^2 \tag{8}$$

The total phase derivative variance can then be found by dividing the sum of the root variance in each dimension by the square of the one-sided length of the neighborhood k (Equation 9). Phase derivative variance is a convenient measure of phase quality because it (a) is easily generalized to arbitrary dimension and (b) can be computed entirely from the phase data itself.

$$\frac{\sum_d \sqrt{\sigma_d}}{k^2} \tag{9}$$

Phase derivative variance is an inverse measure of phase quality; i.e., a large phase derivative variance is indicative of low phase quality. In order to be used as a quality metric, phase derivative variance is typically rescaled into the range $[0,1]$ and multiplied by -1, such that brighter pixels represent higher phase quality. This is the default behavior of `itk::PhaseDerivativeVarianceImageFilter`. The unnormalized phase derivative variance can be obtained by setting `SetNormalize(false)`. The rescaled phase derivative variance image for the noise-corrupted phase ramp is shown in Figure 8.

Note that the patch of noise can be clearly identified in the original wrapped image, the phase residue image, the phase derivative variance image, and the rotational component of the Helmholtz decomposition.

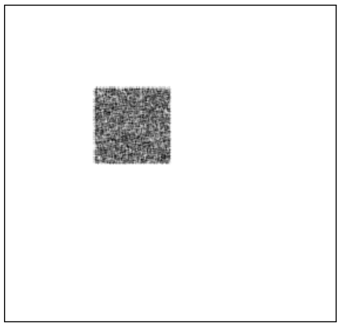


Figure 8: Phase Derivative Variance. Rescaled phase derivative variance for the noise-corrupted phase ramp, computed with `itk::PhaseDerivativeVarianceImageFilter`. The patch of noise can be clearly identified as a region of low image quality. Note that a thin black border has been added in order to allow visualization of the edge of the image.

3 Phase Unwrapping

3.1 Path-Following vs Minimum Norm Algorithms

Path-Following Algorithms

Phase unwrapping algorithms may be categorized as path-following or minimum norm algorithms. Path-following algorithms generally specify one or more starting points at which the phase is considered to be 'true.' Residues, masks, and/or a continuous measure of phase quality may then be used to restrict or guide the allowed path(s), along which the encountered pixels are unwrapped based on local pixel values. Because the unwrapping operation consists entirely of adding some multiple of 2π, the resulting phase is necessarily 'congruent' to the wrapped phase input.

The `itk::ItohPhaseUnwrapping` class presented above is the simplest possible example of a path-following algorithm. More sophisticated algorithms have been described which attempt to restrict the possible paths so as to avoid unwrapping through low-quality regions:

- Goldstein's Branch Cut Phase Unwrapping [7]
- Flynn's Quality-Guided Residue-Mask Phase Unwrapping [3]
- Flynn's Minimum Discontinuity Phase Unwrapping [4]
- Quality-Guided Phase Unwrapping

While Goldstein's and Flynn's algorithms rely on residues, the quality-guided algorithm relies on a user-defined quality map (such as phase derivative variance, discussed above). This distinction is important because of the observation that residues are only physically meaningful in two dimensions. Therefore, Goldstein's and Flynn's algorithms are only useful for two-dimensional phase unwrapping. Phase derivative variance and other quality measures, however, may easily be generalized to arbitrary dimension. Therefore, the quality-guided approach may be implemented for n-dimensional images. For this reason, `itk:QualityGuidedPhaseUnwrappingImageFilter` (defined in `itkQualityGuidedPhaseUnwrappingImageFilter.h`) was implimented for this submission, whereas Goldstein's and Flynn's algorithms were not considered.

Minimum Norm Algorithms

The framework for the minimum L^p-norm algorithms was first presented by Ghiglia, et al, in [6]. Minimum norm algorithms may be subclassified according to (a) whether they take into account phase quality and (b) the exponent of the term being minimized. Weighted algorithms penalize differences in low-quality pixels less than high-quality pixels, whereas unweighted algorithms weight differences in all pixels equally, regardless of quality. To give physical significance to the particular norm chosen, L^0-norm methods minimize the number of pixels that ate not congruent to the input, L^1-norm methods minimize the absolute differences, and L^2-norm methods minimize the squared differences. Unweighted, minimum L^2-norm phase unwrapping may be implimented efficiently by taking advantage of the discrete cosine transform (DCT). This algorithm is implemented in the `itk:DCTPhaseUnwrappingImageFilter` (defined in `itkDCTPhaseUnwrappingImageFilter.h`).

3.2 Quality-Guided Phase Unwrapping: Implementation Summary

The `itk::QualityGuidedPhaseUnwrapping` class iterates over three images: the wrapped image, the quality image, and a binary image. A pixel in the binary image is `true` if the pixel has been unwrapped, or `false` otherwise. Additionally, a list is maintained which contains the index and quality of all 'candidate pixels.' A pixel is considered a candidate if (a) it adjoins a pixel that has been unwrapped and (b) has not been unwrapped.

Efficiently maintaining the list of candidate pixels is of particular importance for this filter. To this end, this submission provides the `itk::IndexValuePair` class (defined in `itkIndexValuePair.h`), which has properties `Index` and `Value`. The class overloads the == operator such that `pair1 == pair2` is `true` if `pair1.Index == pair2.Index` is `true`, and overloads the > operator such that `pair1 > pair2` is `true` if `pair1.Value > pair2.Value` is `true`. As such, instances of this class can be stored in a `std::set`, ensuring that (a) the same pixel location cannot be stored twice and (b) the pixels are sorted according to their quality score. Therefore, an `itk::IndexValuePair` can be added to the list without first checking whether it is already a member, and the highest-quality pixel can always be retrieved by selecting the last element of the `std::set`.

The user sets the active index using the `SetTruePhase()` method. This is taken as the starting point for iteration, and is set to be `true` in the binary image. `itk::IndexValuePair`s for the pixels plus or minus one index in each direction are added to the list of candidate pixels. The pixel in the list with the highest quality is unwrapped and made the active index, and the corresponding index in the binary image is set to `true`. The process repeats while the list of candidate pixels is non-empty.

Once `Update()` has been called, the unwrapped and quality images may be retrieved via the `GetPhase()` and `GetQuality()` methods, respectively. Currently, the filter only supports phase derivative variance as a quality measure, though the filter could be easily extended to include others.

3.3 DCT Phase Unwrapping: Implementation Summary

The unweighted, minimum L^2 norm solution is calculated efficiently in `itk::DCTPhaseUnwrappingImageFilter` (defined in `itkDCTPhaseUnwrappingImageFilter.h`) by taking advantage of Fourier methods for solving Laplace's equation ($\triangle\phi = 0$), as described in [6]. `itk::DCTPhaseUnwrappingImageFilter` uses the two contributed classes `itk::DCTImageFilter` and `itk::DCTPoissonSolverImageFilter`. We refer the reader to the Appendix for a discussion of these classes, as they are of general interest to the image processing community, but may obscure the primary objectives of this submission if described in full here. Briefly:

- The wrapped phase Laplacian of the input is calculated using `itk::WrappedPhaseLaplacianImageFilter` (defined in `itkWrappedPhaseLaplacianImageFilter.h`).
- The forward DCT is taken using `itk::DCTImageFilter` (defined in `itkDCTImageFilter.h`).
- The transformed image undergoes a pixelwise modulation.
- The inverse DCT is taken.

`itk::DCTImageFilter` provides access to the n-dimensional real-to-real transform provided in the FFTW library [5]. For this reason, in order to use this filter (and its dependents), ITK must be built with the cmake variable `USE_FFTWD` set to `ON`, and any project using this filter is subject to the GPL license.

4 Clinical Examples

4.1 Susceptibility Weighted Imaging (SWI) of the Brain

SWI is an MR technique used to enhance microhemorrhage and calcification in the brain. This sequence exploits distortion of the MR field by paramagnetic (e.g., deoxyhemoglobin) and diamagnetic (e.g., dystrophic calcification) compounds, requiring the combination of magnitude and unwrapped phase images. An example of a wrapped phase image obtained from an SWI sequence is shown in Figure 9. The image is an axial slice taken through the brain.

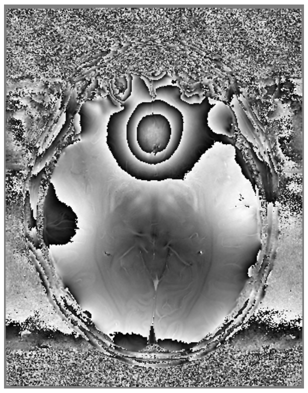
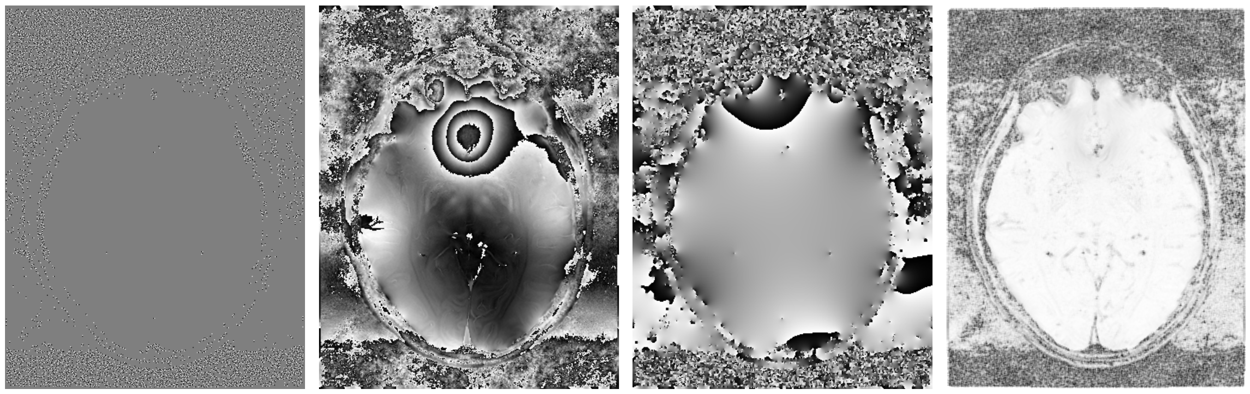


Figure 9: Wrapped phase data from SWI of the brain.

The `itk::PhaseResidueImageFilter`, `itk::PhaseDerivativeVarianceImageFilter`, and `itk::HelmholtzDecompositionImageFilter` classes are demonstrated for the SWI data in Figure 10. The residues are almost entirely restricted to the region outside the skull. The Helmholtz decomposition shows that, as expected, most of the information in the low-quality, residue-rich region outside the skull is due to the rotational contribution, whereas most of the upward spike in phase in the anterior brain tissue is due to the irrotational contribution. The phase derivative variance image shows high quality phase data within the brain tissue and low quality data otherwise.



(a) (b) (c) (d)

Figure 10: Phase data from susceptibility weighted imaging of the brain. Shown are the residues (10a); irrotational component (10b); rotational component (10c); and normalized phase derivative variance (10d).

We now present the results of the Quality-Guided and DCT phase unwrapping algorithms (Figure 11). The

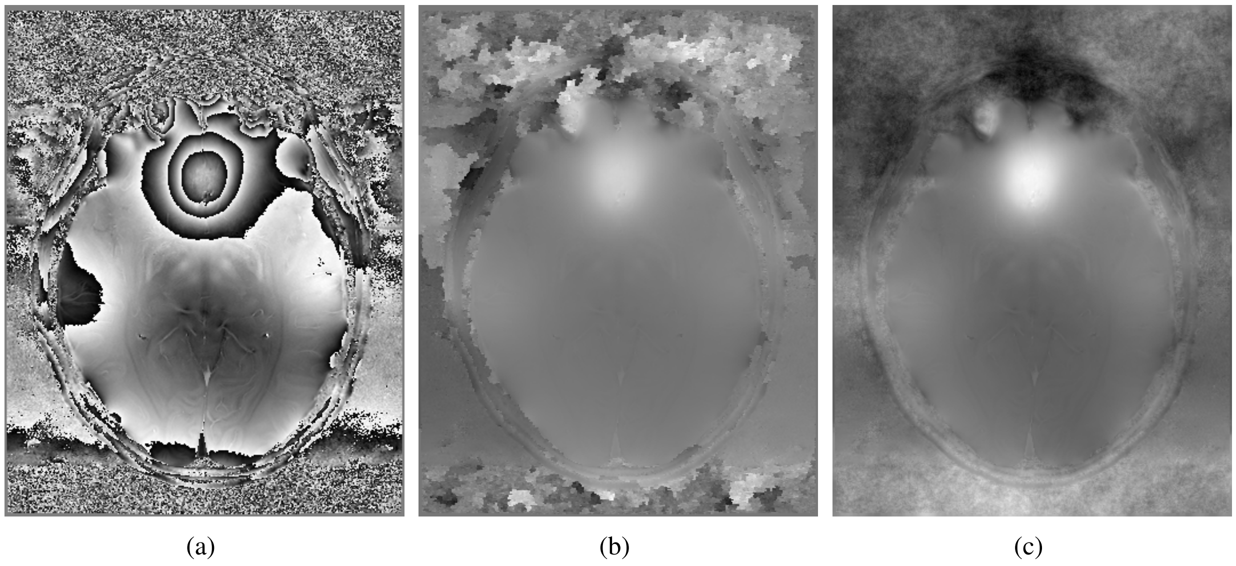


Figure 11: Phase data from susceptibility weighted imaging of the brain. Presented are the original image (11a), the result of the quality-guided phase unwrapping algorithm (`itk:QualityGuidedPhaseUnwrappingImageFilter`, 11b), and of the DCT phase unwrapping algorithm (`itk:DCTPhaseUnwrappingImageFilter`, 11c).

quality-guided approach (center) appears to correctly unwrap through the vast majority of the brain matter. Upon close inspection, however, small areas of 'roughness' are noted in regions of the brain matter closely correlating with residue pairs and clusters in the original image, presumably due to local noise (Figure 12). The minimum L^2-norm approach gives a smooth result throughout.

A cross-section through the vertical centerline of the image is plotted in Figure 13. The quality-guided approach gives a congruent result throughout. The DCT algorithm follows the gradient closely, but underestimates the phase.

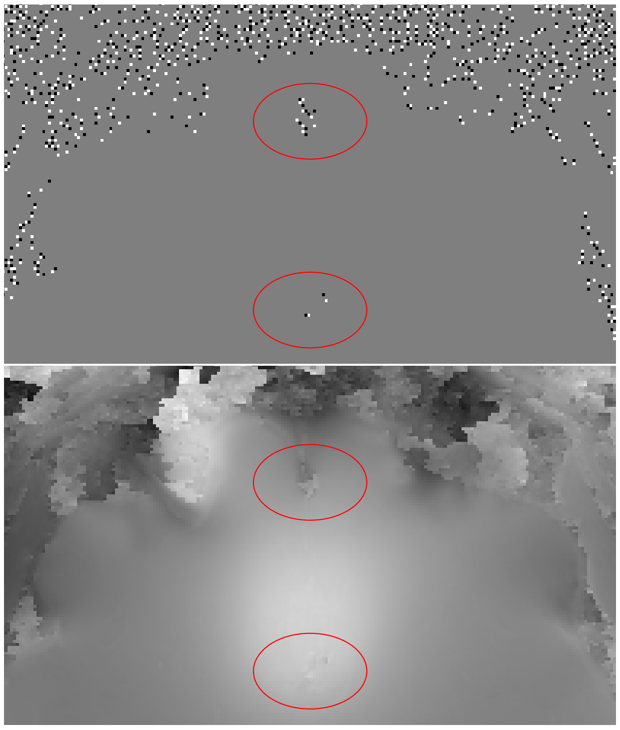


Figure 12: Inset of the SWI residue (upper) and quality-guided phase unwrapped (lower) images. Note the regions of minor discontinuity in the unwrapped image in regions corresponding to pairs/clusters of residues.

4.2 Harmonic Phase (HARP) Imaging of the Heart

Spatial modulation of magnetization (SPAMM) is an MRI pulse sequence which modulates the signal magnitude in a periodic fashion across the plane of the image, resulting in lines of decreased signal [1]. The phase of this magnetude modulation is a material property of the tissue, and as such remains constant over time and with movement. Therefore, tag lines laid down in the heart at end diastole will deform along with the heart during contraction, and so can be used to visualize cardiac deformation (Figure 14).

Harmonic phase analysis is an automated technique for calculating strain from tagged MR images [12]. Cardiac strain is calculated from “HARP” images, which are the result of filtering in the Fourier domain. Many versions of this algorithm require that these images be unwrapped prior to strain calculation [13, 14]. SSFP, SPAMM, and HARP images at end systole and end diastole are shown in Figure 14.

The `itk::PhaseResidueImageFilter`, `itk::PhaseDerivativeVarianceImageFilter`, and `itk::HelmholtzDecompositionImageFilter` classes are demonstrated for the HARP data in Figure 15.

We now present the results of the Quality-Guided and DCT phase unwrapping algorithms (Figure 16). From the unwrapped image it is clear that the quality-guided approach gives an unsatisfactory result, with phase wraps passing through the myocardial wall. The result of the DCT approach is smooth, but otherwise difficult to evaluate by looking at the image directly.

A plot through the centerline of the image shows that the DCT approach is also unsatisfactory, as the gradient differs considerably from that of the wrapped phase input.

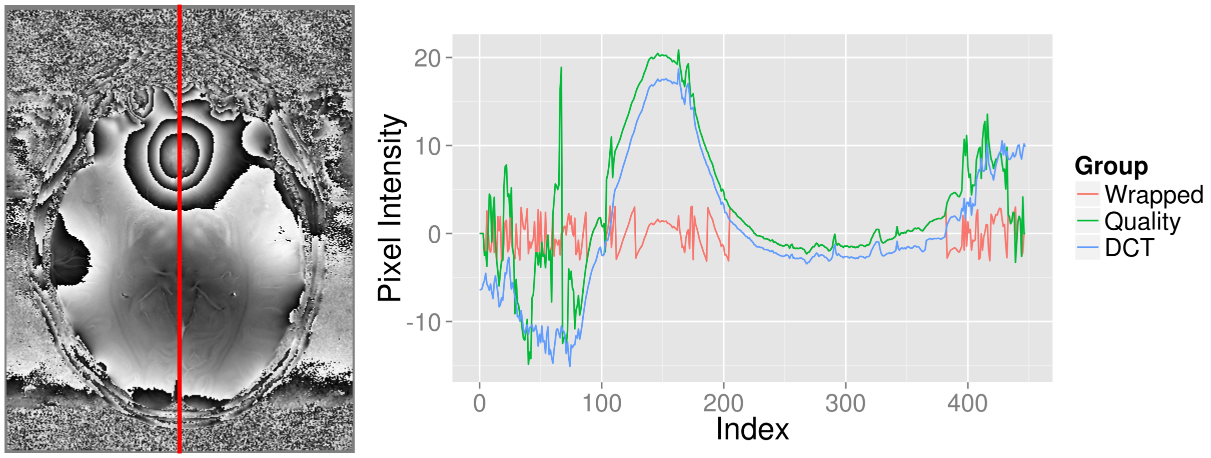


Figure 13: Congruence in SWI Data. The pixel intensities from the vertical centerline of the SWI image are plotted for the unwrapped and original SWI data. Note that the quality-guided approach is congruent throughout, but contains a discontinuity. The DCT approach is smooth throughout, but is not congruent.

4.3 A Note on Computational Speed

The `itk::TimeProbe` class was used to measure the excecution time for the quality-guided and L^2-norm implementations on a single-processor Ubuntu 12.0.2 virtual machine with 4GB of memory. For the SWI data, the quality-guided and L^2-norm approaches took 1.14 and $0.75s$, respectively. For the HARP data, the quality-guided and L^2-norm approaches took 0.40 and $0.27s$, respectively. Successive runs differed by less than 0.01s. Therefore, the quality-guided approach took slighly less than twice as long to execute, though this difference may be insignificant for many applications. Importantly, execution times are expected to grow linearly with input size for the quality-guided approach and by $O(n \log n)$ for the L^2-norm approach. The quality-guided approach could be sped up by allowing for a mask that would prevent the unwrapping of pixels outside the area of interest and the DCT approach could be sped up by taking advantage of multi-threading. However, as has been shown, for some applications it may be necessary to incorporate the DCT algorithm into an iterative algorithm such as preconditioned conjugate gradient weighted L^2-norm in order to achieve satisfactory results, which could negate advantages gained through multithreading.

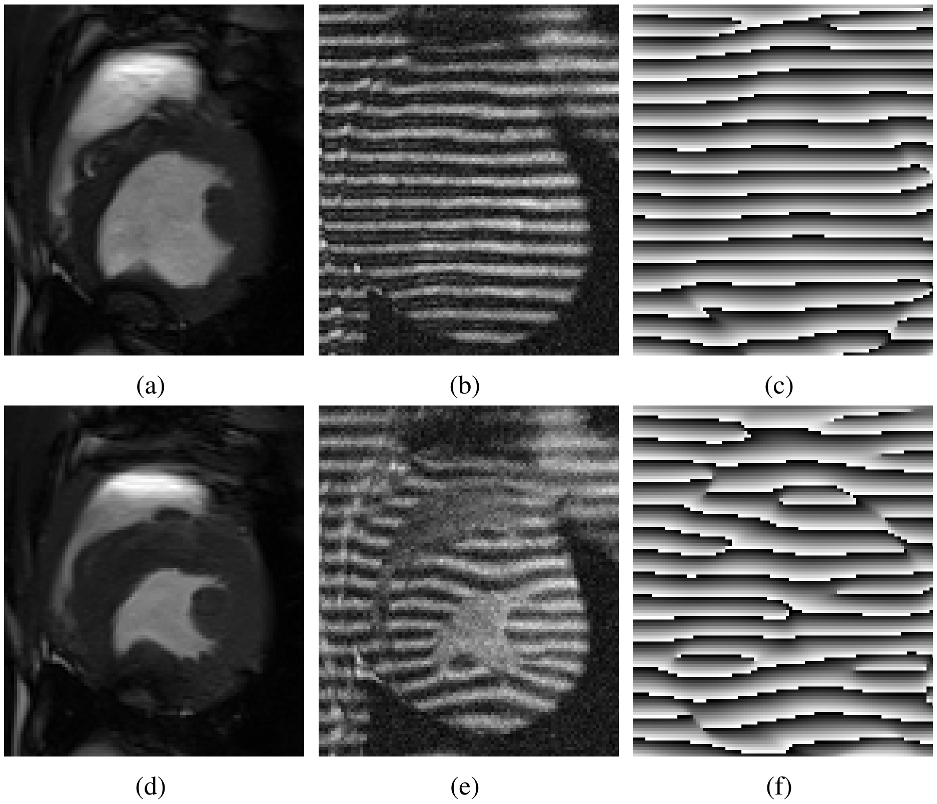


Figure 14: Balanced steady state free precession (SSFP) images (14a, 14d) are shown alongside corresponding tagged SPAMM (14b, 14e) and HARP (14c, 14f) images in a canine subject. Shown are short axis, mid-ventricular images. The upper panels were acquired at end diastole (no contraction); the lower panels were acquired at end systole (maximal contraction). Note that circumferential shortening and radial thickening of the myocardium is apparent by observing the deformation of the tag lines.

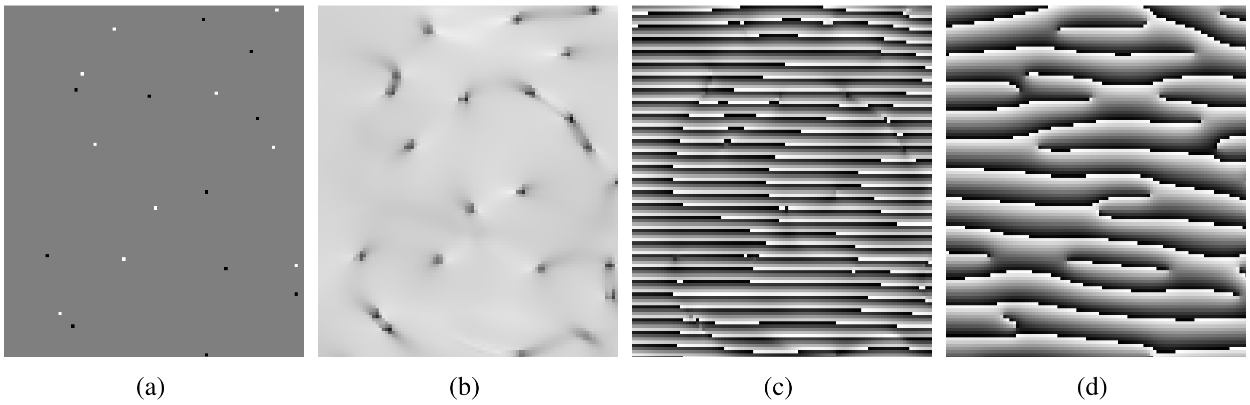


Figure 15: Phase data from HARP imaging of the heart at end systole. Shown are the residues (15a), phase derivative variance (15b), irrotational component (15c), and rotational component (15d).

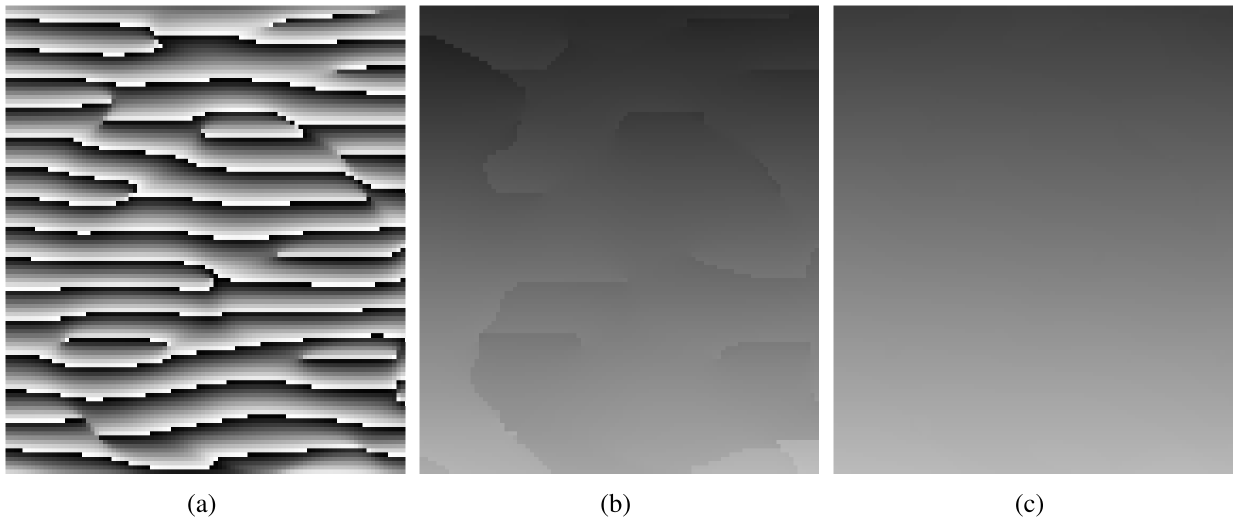


Figure 16: Results of the phase unwrapping algorithms. Quality guided (16b) and DCT (16c) results are presented. The original wrapped image is also presented (16a) for comparison.

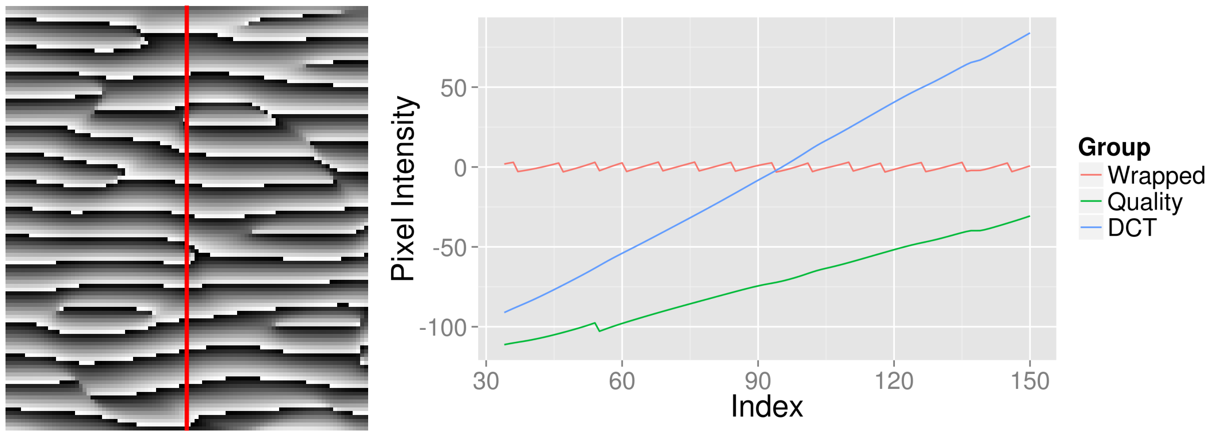


Figure 17: Congruence in HARP Data. The pixel intensities from the vertical centerline of the HARP image are plotted for the unwrapped and original HARP data. Note that the quality-guided approach is congruent throughout, but contains a discontinuity. The DCT approach is smooth throughout, but is not congruent.

5 Conclusions and Future Directions

ITK is a large and powerful framework for medical (and general) image processing. However, the lack of filters for understanding, manipulating, and unwrapping phase data is a current limitation of the library. We have here presented an ITK module which we hope will begin to bridge that gap. The two most significant contributions are the `itk::QualityGuidedPhaseUnwrappingImageFilter` and `itk::DCTPhaseUnwrappingImageFilter` classes, which implement efficient n-dimensional unwrapping algorithms. The quality-guided approach has the advantage of giving a result that is congruent to the input. Moreover, this approach avoids low-quality phase data, given an adequate quality map. The unweighted L^2-norm approach has the advantage of giving a smooth result throughout, but is not congruent with the input and weights all pixels equally regardless of quality.

These algorithms both gave servicable results when presented with the SWI data, which had few residues within the region of interest, and in which the low quality data was largely relegated to the periphery. However, both algorithms failed to produce an adequate result when presented with the more difficult HARP image. In the case of the quality-guided approach, this is likely due to the inadequacy of phase derivative variance as a quality map, because in HARP images phase varies quite smoothly even in regions where there is little to no signal. In the case of the DCT algorithm, this is likely because the region of interest is relatively small compared to the image as a whole.

In the future, it would be of great benefit to allow for other quality maps (such as maximum phase gradient, pseudocorrelation, and user-defined masks) in addition to phase derivative variance. This would allow for finer control over the path the algorithm takes in the case of difficult cases such as HARP images. Additionally, it would be of benefit to implement a weighted L^2-norm method, so that the DCT approach could also exclude low-quality or uninteresting regions. Weighted L^2-norm phase unwrapping algorithms have been described which iteratively apply unweighted algorithms to weighted wrapped phase Laplacians. The preconditioned conjugate gradient (PCG) approach in particular makes use of this method [6], and would be an important next step in the development of this module.

This submission has also described `itk::DCTImageFilter` and `itk::DCTPoissonSolverImageFilter`, which are efficient implementations of general-purpose utilities important in image compression, gradient image editing, and phase unwrapping. The former is a simple wrapper to the FFTW library, allowing for the discrete cosine transform to be integrated into an ITK pipeline. The latter makes use of the DCT class to recover an image from its Laplacian. We refer the interested reader to the appendices for a proper discussion.

A Laplace's Equation

The Laplacian operator, Δ, is defined for an n-dimensional function as the sum of its unmixed partial second derivatives (Equation 10):

$$\Delta f = \nabla^2 f = \sum_{i=1}^{n} \frac{\delta^2 f}{\delta x_i^2} \tag{10}$$

In the two-dimensional case, this takes the following familiar form (Equation 11):

$$\Delta f = \frac{\delta^2 f}{\delta x^2} + \frac{\delta^2 f}{\delta y^2} \tag{11}$$

In the special case where $\Delta f = 0$, this is known as Laplace's Equation. The discrete Laplacian may be approximated via simple second differences. The one-dimensional second-difference operator $[1 \quad -2 \quad 1]$ may be taken successively in each image dimension to produce a Laplacian image. In the two-dimensional case, the discrete Laplacian operator is computed as follows:

$$\begin{array}{c} 0\ 1\ 0 \\ 1\ \text{-4}\ 1 \\ 0\ 1\ 0 \end{array} \tag{12}$$

In ITK, the discrete Laplacian operator is provided in the `itk::LaplacianOperator` class, and is applied to an image using the `itk::LaplacianImageFilter` class. Laplace's equation can be solved efficiently by means of the discrete cosine transform (DCT) [6]. In particular, the original image f can be recovered by taking the forward DCT of the Laplacian image Δf, applying a pixelwise modulation, and taking the reverse DCT on the result (Equation 13). In this equation, n is the dimension of the image, S_n is the number of pixels across the image in direction d, and i_n is the pixel index in direction n.

$$f = DCT^{-1}\left\{ \frac{DCT\{\Delta f\}}{\sum_{n=0}^{N-1} 2\cos\left(\frac{\pi i_n}{S_n}\right) - 2N} \right\} \tag{13}$$

It is possible to formulate the problem in such a way that it can be solved with either the fast Fourier transform (FFT), discrete sine transform (DST), or discrete cosine transform (DCT). The specifics of each implementation differ primarily in the steps that must be undertaken to satisfy Neumann boundary conditions. In order to be solved with the FFT, for instance, the Laplacian must be reflected to the left and upwards (Figure 18). The DCT (also known as the 'even' or 'real-to-real' transform), however, is performed on one half of an implicitly reflected matrix, obviating the reflection step. Though this difference is likely negligible for an isolated 2D calculation, use of the DCT could significantly reduce memory and processing requirements when applied iteratively or to higher dimensional data. For this reason, the DCT is prefered if available.

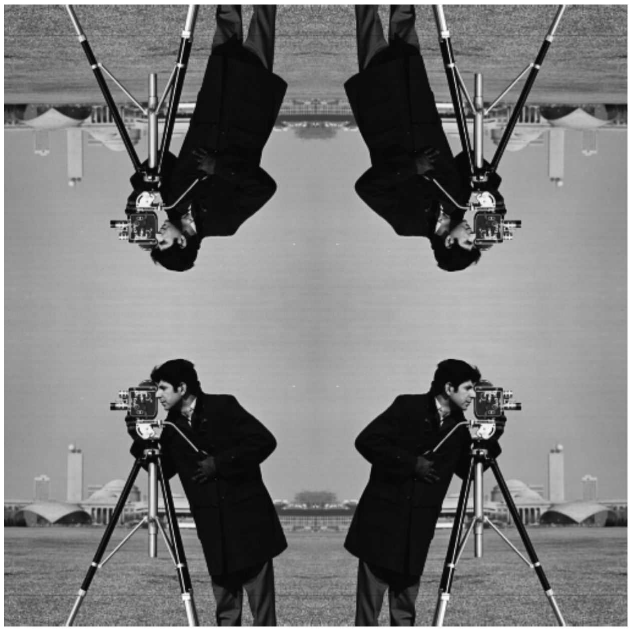


Figure 18: The “cameraman” test image, reflected to the left and upwards to satisfy Neumann boundary conditions.

B Implementation and Usage

B.1 Discrete Cosine Transform

FFTW (the ‘Fastest Fourier Transform in the West’ [5]) is a library in the C programming language, which can be made available by building ITK with the CMake variable `ITK_USE_FFTWF` or `ITK_USE_FFTWD` (single and double precision floating point, respectively) to `ON`. FFTW provides a wide selection of efficient discrete transforms, including DCT types I–IV (corresponding to ‘`kinds`’ `REDFT00`, `REDFT10`, `REDFT01`, and `REDFT11`, respectively). These plans differ practically from one another in terms of inverse transforms and normalization procedures (`REDFT00` and `REDFT11` are inverses of themselves, whereas `REDFT10` and `REDFT01` are inverses of each other). In particular, `REDFT10` and `REDFT01` are known as ‘the’ forward and reverse DCT, respectively. A real-to-real plan is constructed according to the following function definition:

```
fftw_plan fftw_plan_r2r(int rank, const int *n, double *in, double *out,
                                const fftw_r2r_kind *kind, unsigned flags);
```

`int rank` refers to the image dimension; `const int *n` to an array indicating the size in each dimension; `double *in` and `double *out` to input and output arrays allocated using `fftw_malloc`, `const fftw_r2r_kind *kind` to the type of transform that should be performed in each dimension, and `unsigned flags` to instructions concerning numerical precision. The `itk::Image` buffers are assigned to the input and output arrays using the `GetBufferPointer()` method. Each element of the `const fftw_r2r_kind` array is set to `FFTW_REDFT01` in the case of the forward transform and `FFTW_REDFT10` in the case of the inverse transform. The transform direction can be specified at the level of the itk class using the `TransformDirection()` method. Because FFTW returns an unnormalized transform, the image is divided by `numpix*pow(2, TInputImage::ImageDimension)` in the case of the inverse transform.

The provided class `itk::DCTImageFilter`, which inherits from `itk::ImageToImageFilter`, provides a simple, templeted interface to the `REDFT01` and `REDFT10` plans. The following minimal example code

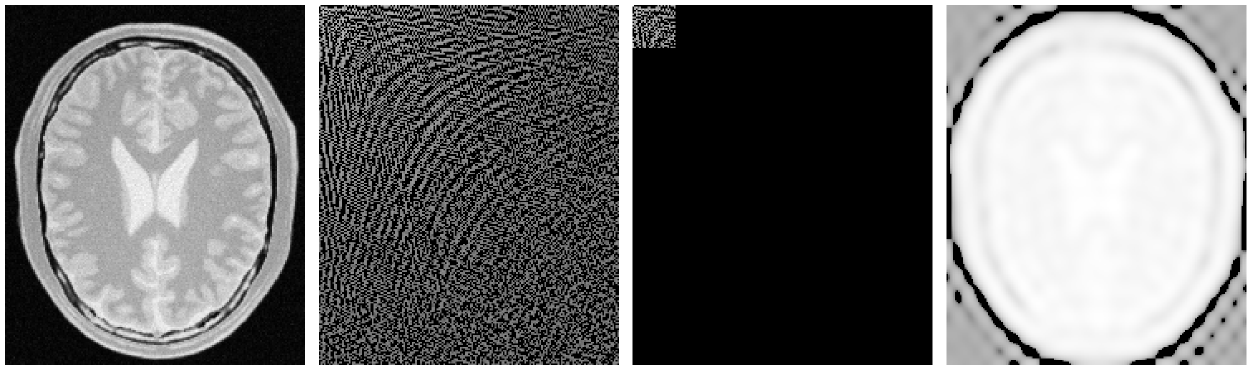


Figure 19: Simple demonstration of low-pass DCT filtering. Shown are the input image (19a), the DCT transform (19b), the masked transform(19c), and the output image (19d).

demonstrates definition, instantiation, and usage of the class, and Figure 19 illustrates a simple low-pass DCT filtering example (source code provided).[7]

```
typedef itk::DCTImageFilter< WorkImageType, WorkImageType > DCTType;
DCTType::Pointer dct = DCTType::New();
dct->SetInput( reader->GetOutput() );
dct->SetTransformDirection( DCTType::Forward );
dct->Update();
```

B.2 Poisson Solver

The `itk::DCTPoissonSolver` class transforms the input, applies a modulation, and reverse transforms back into the image domain, as described in Equation 13. The following minimal example demonstrates definition, instantiation, and usage of the `itk::DCTPoissonSolverImageFilter` class.

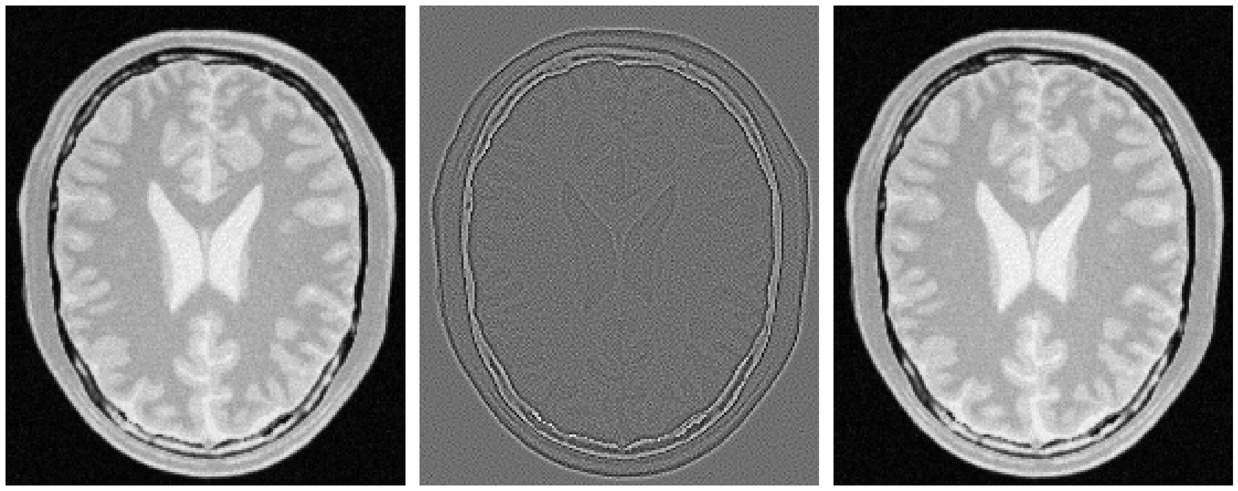
```
typedef itk::DCTPoissonSolverImageFilter< WorkImageType, WorkImageType >   PoissonType;
PoissonType::Pointer solver = PoissonType::New();
solver->SetInput( laplacian->GetOutput() );
solver->Update();
```

Figure 20 shows an example of an image of the brain (20a), its Laplacian (20b), and the solution of the Laplacian (20c). In the rescaled image, the solution appears to match exactly the input image. However, by plotting the vertical centerline of each image (Figure 21), we see that the input and output differ by an additive constant of ~ 123.7, as is expected from any integration procedure.

References

[1] L Axel and L Dougherty. MR Imaging of Motion with Spatial Modulation of Magnetization. *Radiology*, 171:841–845, 1989. 4.2

[7]Note that all images have been linearly rescaled (except for transforms, which have been logarithmically rescaled) for visualization purposes.



(a) (b) (c)

Figure 20: The original (20a), Laplacian (20b), and solution of the Laplacian (20c) images for an axial CT slice through the brain.

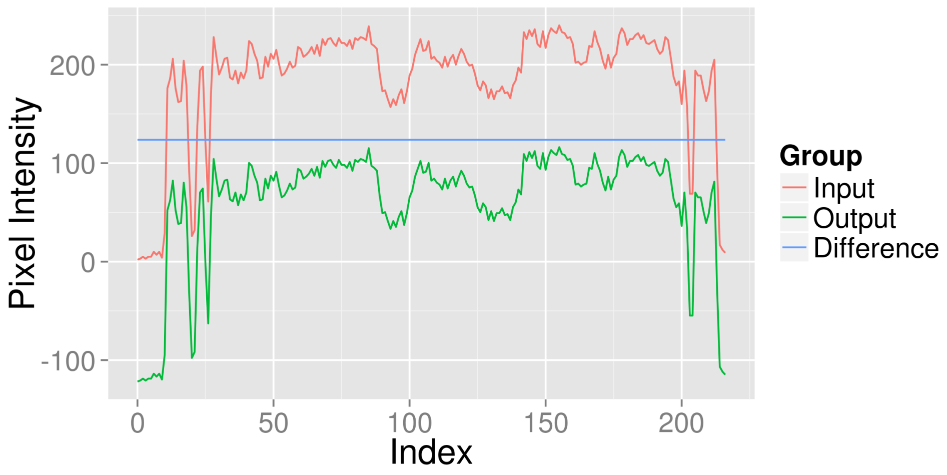


Figure 21: Plot of grayscale intensities along the vertical centerline of the input and output images. Note that the two curves differ by an additive constant.

[2] D J Bone. Fourier fringe analysis: the two-dimensional phase unwrapping problem. *Applied optics*, 30(25):3627–3632, 1991. 2.2, 2.2

[3] Thomas J Flynn. Consistent 2-D phase unwrapping guided by a quality map. *IGARSS '96. 1996 International Geoscience and Remote Sensing Symposium*, 4:2057–2059, 1996. 3.1

[4] Thomas J. Flynn. Two-dimensional phase unwrapping with minimum weighted discontinuity, 1997. 3.1

[5] Matteo Frigo and Steven Johnson. The Design and Implementation of FFTW3. *Proc. IEEE*, 93(2):216–231, 2005. 3.3, B.1

[6] DC Ghiglia and MD Pritt. *Two-dimensional phase unwrapping: theory, algorithms, and software*. 1998. 3.1, 3.3, 5, A

[7] RM Goldstein, HA Zebker, and CL Werner. Satellite radar interferometry: Two-dimensional phase unwrapping. *Radio Science*, 23(4):713–720, 1988. 4, 2.2, 3.1

[8] Jonathan M Huntley. Three-dimensional noise-immune phase unwrapping algorithm. *Applied Optics*, 28(23):3268–3270, 2001. 2.2

[9] Kazuyoshi Itoh. Analysis of the phase unwrapping algorithm. *Applied optics*, 21(14):2470, 1982. 1.1, 2.1

[10] Wei Li, Bing Wu, and Chunlei Liu. Quantitative susceptibility mapping of human brain reflects spatial variation in tissue composition. *NeuroImage*, 55(4):1645–56, April 2011. 1.1

[11] N F Osman, W S Kerwin, E R McVeigh, and J L Prince. Cardiac motion tracking using CINE harmonic phase (HARP) magnetic resonance imaging. *Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine*, 42(6):1048–60, December 1999. 1.1

[12] N F Osman, W S Kerwin, E R McVeigh, and J L Prince. Cardiac motion tracking using CINE harmonic phase (HARP) magnetic resonance imaging. *Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine*, 42(6):1048–60, December 1999. 4.2

[13] Bharath Ambale Venkatesh, Himanshu Gupta, Steven G Lloyd, Louis Dell 'Italia, and Thomas S Denney. 3D left ventricular strain from unwrapped harmonic phase measurements. *Journal of magnetic resonance imaging : JMRI*, 31(4):854–62, April 2010. 4.2

[14] Bharath Ambale Venkatesh, Chun G Schiros, Himanshu Gupta, Steven G Lloyd, Louis Dell'Italia, and Thomas S Denney. Three-dimensional plus time biventricular strain from tagged MR images by phase-unwrapped harmonic phase. *Journal of magnetic resonance imaging : JMRI*, 34(4):799–810, October 2011. 4.2