

A Real-Time Navigator Approach to Compensating
for Motion Artifacts in Coronary Magnetic
Resonance Angiography

Amelia Buerkle

Keith Chung
Farhan Shamsi

Anthony Nuval

May 10, 2002

Abstract

Object motion results in a degradation of image quality in magnetic resonance imaging. This is especially true in coronary magnetic resonance angiography where the subject is in constant respiratory and cardiac motion. The simultaneous multiple volume acquisition algorithm, a technique to compensate for this motion, is under development at the Cornell Medical Center. Improvements to the underlying algorithm are made, increasing the efficiency of the overall algorithm.

Contents

1	Introduction	1
2	Principles of Magnetic Resonance Imaging	5
2.1	Nuclear Magnetic Resonance Theory	5
2.1.1	The Bloch equation	8
2.1.2	Excitation	10
2.2	Introduction to Signal Localization	11
2.2.1	Selective Excitation	12
2.2.2	Frequency Encoding	15
2.2.3	Phase Encoding	15
2.2.4	An Introduction to k -space	16
2.2.5	How it all comes together: Pulse Sequences and the Scanner . . .	19
2.3	Image Reconstruction	25
2.3.1	General Image Reconstruction	26
2.3.2	Fourier Transform Reconstruction	26
2.3.3	Radon Transform Sample Reconstruction	28
2.3.4	A Unified View of Image Reconstruction	30
3	Motion and Magnetic Resonance Imaging	33
3.1	Effects of Motion on Image Reconstruction	33
3.2	Motion Compensation Techniques	37
4	Our Work	45
4.1	Testing Hardware	45
4.2	Testing Software	46
4.3	The PAWS Algorithm	47
4.4	Results	49
5	Future Improvements	51
5.1	Edge Detection	51

6 Conclusion	55
A The Code	57
B Data	89

Chapter 1

Introduction

Reducing the effects due to unwanted motion is one of the main challenges in developing effective imaging techniques. Magnetic resonance imaging (MRI), a noninvasive technique used primarily in medical applications to allow visualization of the interior of the human body, suffers from motion artifacts. The primary focus of this investigation is on coronary magnetic resonance angiography (CMRA), in which motion artifacts arise from coronary motion due to respiration and normal cardiac rhythms. Phase ordering with automatic window selection (PAWS) [1] is an algorithm that is currently used to minimize the occurrence of motion artifacts in CMRA, but it is inefficient due to discarded data. Simultaneous multiple volume acquisition (SMVA), an algorithm under development at the Weill Medical College of Cornell University, resolves some of the inefficiencies of PAWS by using the discarded data to reconstruct another image. Work on improving the algorithm was carried out at the facilities at Cornell University, where a General Electric MRI scanner that performs the actual scanning along with a Sun Microsystems workstation used for the intermediate calculation and feedback to the scanner serves as the development base. The user interface of the current implementation is compliant with the Motif Window Manager.

Magnetic resonance imaging is based on the nuclear magnetic resonance (NMR) phenomenon that was first discovered by Felix Bloch and Edward Purcell in 1946. Both Purcell and Bloch were awarded the Nobel Prize in 1952 for their discovery. Further developments made by Paul Lauterbur in 1972 enabled image formation using NMR signals. The technique became known as magnetic resonance imaging rather than nuclear magnetic resonance imaging as a result of the negative connotations associated with the word "nuclear." In 1977, Raymond Damadian successfully demonstrated MRI on the human body. Today, MRI enables physicians to diagnose patients without exploratory surgery.

The quality of the image is one of the most critical properties of the MRI process. Images obtained from the scanner are used for diagnosis and must provide a clear picture

for the physician to examine the internal structure of the body. This is especially true in CMRA, where the volume being imaged include the small structures within the heart and lungs. However, respiration and cardiac motion hinder the imaging process. The main challenge is to minimize the occurrence of motion artifacts, such as blurring and ghosting.

The MRI technique relies on a strong uniform magnetic field, which establishes a reference axis. Three orthogonal coils provide local variation in the magnetic field in order to isolate a region of space. In order to determine the properties of the region, a probing signal is used to elicit a response, known as an echo. A variety of echoes can be used to derive image information from an object. Navigator echoes are a special class of echoes that are collected primarily for detection of object motion.

The development of navigator echo techniques has enabled the implementation of a variety of algorithms to reduce motion artifacts. Gating, one of the first approaches based on navigator echoes, requires that the data acquisition occur within a narrow acceptance window to reduce the effective variability of the patient's respiration. Early gating techniques were inefficient because the acquired data that fell outside the window were discarded. Another method, known as view ordering, has been shown to improve image quality by rearranging the data to minimize motion effects. Furthermore, the combination of view ordering and gating has allowed for larger acceptance windows and a more efficient scan. However, the effectiveness of any algorithm based on acceptance windows depends on the selection of the window. A change in a patient's respiration pattern during a scan may cause a window to be out of range and render the data collected useless.

To solve this problem, an algorithm known as phase ordering with automatic window selection (PAWS) [1] uses multiple windows; the window that completes its acquisition first is chosen. Although this technique is resistant to changes in respiration, it still discards the data acquired in the windows that are not chosen. Simultaneous multi-volume acquisition (SMVA) is an algorithm that takes advantage of the data that are not chosen to acquire another image. An implementation of the SMVA algorithm requires dynamic histogram generation to determine the most probable location of the diaphragm position. In addition, a scheduling algorithm is used to complete acquisition as fast as possible.

Aside from the algorithm, the actual implementation relies on a graphical user interface (GUI) based on the Motif Window Manager to complete the end-to-end functionality. The GUI provides user controls and display information about the scan in real-time. As the scan is being performed, the user can view the results of the navigator echoes as well as track the progress of the scan. User-determined parameters, such as the type of navigator echo to be used and the number of images to acquire, are specified through the GUI.

The overall implementation operates as follows. The patient is first scanned with the General Electric MRI scanner to obtain a set of preliminary images. In order to eliminate motion artifacts in the rest of the images, a position on the diaphragm is chosen as a reference for the SMVA algorithm. During the second scan, a Sun Microsystems workstation processes the data from the scanner as required by the algorithm. The final images are retrieved with an inverse Fourier transform on the resulting data set. The efficiency of the algorithm is calculated as the percentage of data that is used to reconstruct the images.

The purpose of this work is to optimize the PAWS algorithm and make its implementation stable. The following chapters explain the principles behind NMR, data acquisition and image reconstruction. The document continues into a description of motion effects in MRI and the various algorithms used to correct them. The findings of this study are then explained. The hardware and software components used in this study are also described. Finally, the improvements due to the changes are calculated.

Chapter 2

Principles of Magnetic Resonance Imaging

2.1 Nuclear Magnetic Resonance Theory

Magnetic Resonance Imaging (MRI) is a non-invasive imaging technique used primarily in medical settings to produce images of the inside of the human body. MRI is based on the nuclear magnetic resonance (NMR) phenomenon first observed by Bloch and Purcell in 1946. NMR principles have made high quality image recovery possible by allowing the encoding of spatial information into signals that can be detected outside the body.

The MRI hardware consists of a main magnet along with three gradient coils to provide smaller magnetic fields. The MRI machine is a large cube with a horizontal tube, known as the *bore*, running through the center. The patient lies on his back or stomach so that the part of the body to be scanned is at the *isocenter* (the exact center) of the main magnet. The main magnet provides a very stable and intense magnetic field while the gradient coils provide variable localized fields. Transmitter coils generate oscillating magnetic fields as probe signals while receiver coils convert magnetic fields into electrical signals.

NMR is based on the principle that any object can be subdivided into atoms. Each atom is composed of two types of charged particles: protons and electrons. Subatomic particles such as electrons and protons have a property known as *spin*. Although a proper explanation of subatomic particle behavior requires quantum mechanics, MRI principles can be accurately described using classical models because MRI involves collections of particles. Spin is often associated with the angular momentum \vec{J} of the atom and may be visualized as a top spinning about its vertical axis as shown in Figure 2.1. Spin values come in integer multiples of 0.5 and may be positive or negative. The spin for an unpaired subatomic particle is 0.5. The net spin of a collection of particles is the sum

of the individual spins. Techniques based on NMR, including MRI, are dependent on collections of particles with net spin.

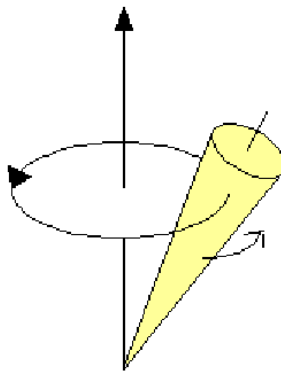


Figure 2.1: Precession

According to Ampere's law, a time-varying electric charge distribution induces a magnetic field. As a consequence, the nucleus of an atom with net spin will produce a small magnetic field. This magnetic field is represented by the vector quantity $\vec{\mu}$ and known as the *nuclear magnetic dipole moment* or the *magnetic moment*. The magnetic moment is related to the spin angular momentum by the following relationship:

$$\vec{\mu} = \gamma \cdot \vec{J} \quad (2.1)$$

The constant of proportionality, γ , known as the *gyromagnetic ratio*, is dependent on the nucleus type. The γ values for select nuclei are listed in Table (2.1).

Nucleus	Spin	Gyromagnetic Ratio
^1H	0.5	42.58
^{13}C	0.5	10.71
^{19}F	0.5	40.05
^{31}P	0.5	11.26

Table 2.1: NMR properties of select nuclei

Most clinical applications of MRI require an abundance of hydrogen inside the object to be imaged because its relatively high gyromagnetic ratio eases the imaging process. The human body is primarily fat and water, both of which have many hydrogen atoms.

The magnitude of the magnetic moment can be calculated by

$$\mu = \gamma \cdot \hbar \cdot \sqrt{I(I + 1)} \quad (2.2)$$

where \hbar is Planck's constant h (approximately $6.626 \cdot 10^{-34} \text{J}\cdot\text{s}$) divided by 2π , and I , the *nuclear spin quantum number*, is the magnitude of the spin value mentioned earlier and must take on a value that is a positive integer multiple of 0.5, such as

$$I = 0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0 \quad (2.3)$$

Although the magnitude of $\vec{\mu}$ is known, its direction is random unless an external magnetic field is applied. When placed in an external magnetic field, the magnetic moment vector $\vec{\mu}$ of a subatomic particle will align with the external field in two possible energy states. The low energy state requires that the magnetic moment align itself parallel to the direction of the magnetic field, as shown in Figure 2.2. The higher energy state, shown in Figure 2.3, occurs when the magnetic moment is aligned in the opposite direction as the main magnetic field.



Figure 2.2: Low energy state for a particle



Figure 2.3: High energy state for a particle

When placed in a magnetic field of strength B , a particle with a net spin can absorb a photon of frequency ω . The frequency of the absorbable photons is linearly dependent on the strength of the magnetic field:

$$\omega = \gamma B \quad (2.4)$$

A particle with net spin can undergo a transition between the two energy states with the absorption of a photon. Because particles have only two discrete energy states, the energy of the absorbed photon must equal the difference between the states. The energy of a photon is linearly dependent on its frequency of the photon:

$$E = h\omega \quad (2.5)$$

In this case, the constant of proportionality is Planck's constant h . The frequency ω is known as the *Larmor frequency*. Equations (2.4) and (2.5) can be combined for an expression of the energy between the states:

$$E = h\gamma B \quad (2.6)$$

In order to create a macroscopic magnetic field inside the body, an external magnetic field is applied. Convention dictates that the field have a strength of B_0 and lie along the z -axis such that

$$\vec{B}_0 = B_0 \vec{k} \quad (2.7)$$

where \vec{i} , \vec{j} , and \vec{k} represent unit vectors in the x , y , and z directions, respectively.

2.1.1 The Bloch equation

A rotating reference frame is a coordinate system whose transverse (XY) plane is rotating with respect to the laboratory's coordinate system. In this case, the rotation is assumed to be clockwise at a frequency ω . The new axes will be labeled x' , y' , and z' to distinguish them from the conventional frame. This frame is related to the conventional frame by the following transform:

$$\vec{i}' = \cos(\omega t)\vec{i} - \sin(\omega t)\vec{j} \quad (2.8)$$

$$\vec{j}' = \sin(\omega t)\vec{i} + \cos(\omega t)\vec{j} \quad (2.9)$$

$$\vec{k}' = \vec{k} \quad (2.10)$$

The individual spins of the atoms can add up to a bulk magnetization \vec{M} using the following relation:

$$\vec{M} = \sum_{n=1}^N \vec{\mu}_n \quad (2.11)$$

where N is the total number of spins in the object being imaged. At equilibrium, the bulk magnetization vector lies in the direction of the applied uniform magnetic field \vec{B}_0 and is represented by \vec{M}_0 . Conventionally, the uniform magnetic field is applied in the z -direction; therefore, the bulk magnetization vector also lies in the z -direction. There

are no transverse (M_x or M_y) components for the bulk magnetization, assuming that there are no other external fields other than the main magnetic field. The variations in the bulk magnetization vector can be described by

$$\frac{d\vec{M}}{dt} = \gamma \vec{M} \times \vec{B}_0 \quad (2.12)$$

Equation (2.12), known as the simple form of the *Bloch equation*, describes the *precession* that the bulk magnetization vector undergoes if the system is left undisturbed.

If the system is perturbed from its equilibrium state, the laws of thermodynamics dictate that it will return to its equilibrium state provided that the perturbing agent is removed and sufficient time is given. If the bulk magnetization has a component in the xy -plane, it will rotate about the z -axis at a frequency equal to the Larmor frequency. This revolution of the bulk magnetization \vec{M} in the xy -plane is known as *free precession*. The recovery in the axis of the external magnetic field is described by

$$\frac{dM_{z'}}{dt} = -\frac{(M_{z'} - M_{0,z'})}{T_1} \quad (2.13)$$

where T_1 is the longitudinal or spin-lattice time constant and $M_{z'}$ and $M_{0,z'}$ are the z -components of the instantaneous and equilibrium bulk magnetization vectors respectively. This phenomenon is known as *longitudinal relaxation* or T_1 *relaxation*. As the bulk magnetization returns to its equilibrium state, the transverse component is attenuated. This decay, known as *transverse relaxation* or T_2 *relaxation*, is defined by

$$\frac{dM_{x'}}{dt} = -\frac{M_{x'}}{T_2} \quad (2.14)$$

where T_2 is known as the transversal or spin-spin time constant and $M_{x'}$ denotes the component of the bulk magnetization vector in the x -direction; a similar relationship exists in the y -direction. The two relaxation processes occur simultaneously and are only separated by the axes along which they happen. Experience has shown that T_2 is always less than or equal to T_1 . Unfortunately, equation (2.14) describes an ideal case. There are two primary factors that contribute to transverse relaxation: molecular interactions and variations in the main magnetic field B_0 . Molecular interactions contribute to what is known as *pure T_2* while aberrations in B_0 contribute to an *inhomogeneous T_2 effect*. Variations in the main magnetic field change the T_2 constant to a $T_2^{(*)}$, related by:

$$\frac{1}{T_2^{(*)}} = \frac{1}{T_2} + \frac{1}{T_{2,\text{inhom}}} \quad (2.15)$$

A general equation describing a change in the magnetization can be obtained by combining the simple Bloch equation with equations (2.13) and (2.14):

$$\frac{d\vec{M}_{\text{rot}}}{dt} = \gamma \vec{M}_{\text{rot}} \times \vec{B}_0 - \frac{(M_{z'} - M_{0,z'})\vec{k}'}{T_1} - \frac{M_{x'}\vec{i}' + M_{y'}\vec{j}'}{T_2} \quad (2.16)$$

2.1.2 Excitation

A uniform external field aligns the individual magnetic moments in the body along the z -axis. The transverse component of the bulk magnetization vector is zero at equilibrium because the magnetic moments have random phases. The *phase* of a magnetization vector is the angle between the x -axis and the projection of the vector onto the transverse plane. The constituents of the bulk magnetization vector rotate at the same frequency about the z -axis but are randomly oriented in the transverse plane and cancel each other.

The magnetic moments must constructively interfere or have phase coherence to have a non-zero transverse magnetization component. In order to enforce phase coherence, a perturbing field is added in the transverse plane. Establishment of phase coherence among the precessing spins is referred to as *resonance*. To establish phase coherence, an oscillating magnetic field $\vec{B}_1(t)$ of the form

$$\vec{B}_1(t) = 2B_1^e(t) \cos(\omega_{\text{rf}}t + \varphi)\vec{i} \quad (2.17)$$

is applied in the transverse (xy) plane. The B_1 field is also known as an RF pulse because it has a short duration and oscillates in the radio-frequency range. According to Planck's law, the energy associated with an electromagnetic wave is proportional to its frequency:

$$E_{\text{rf}} = h\omega_{\text{rf}} \quad (2.18)$$

The energy provided by the RF pulse must equal the difference in the energy states for the magnetic moment. Equation (2.6) stated that the energy of an absorbed photon is directly proportional to the magnetic field experienced by the atom. Since radiation energy is interchangeable, one can say that the energy for an RF pulse must equal

$$E = h\gamma B \quad (2.19)$$

where B is the magnetic field experienced by an atom. If

$$\omega_0 = hB \quad (2.20)$$

then resonance requires

$$\omega_{\text{rf}} = \omega_0 \quad (2.21)$$

Equation (2.21) is known as the *resonance condition*. When a collection of particles begins to resonate, the bulk magnetization precesses. A group of particles that share the same resonant frequency is called an *isochromat*. Earlier, it was stated that a time varying electric charge produces a magnetic field; similarly, a time varying magnetic field produces an electric field. This means that any conducting loop resonating at ω_0

can be used as a receiver coil. Specifically, the flux through the resonating external coil by $M(\vec{r}, t)$ can be described by

$$\Phi = \int_{\text{object}} \vec{B}_r(\vec{r}) \cdot \vec{M}(\vec{r}, t) d\vec{r} \quad (2.22)$$

where $B_r(\vec{r})$ is the laboratory frame magnetic field at location \vec{r} . Then, according to Faraday's law of induction, the voltage induced in the coil is

$$V(t) = -\frac{\partial \Phi}{\partial t} = -\frac{\partial}{\partial t} \int_{\text{object}} \vec{B}_r(\vec{r}) \cdot \vec{M}(\vec{r}, t) d\vec{r} \quad (2.23)$$

This system is not practical because the output signal represents a sum of all resonant particles and gives no spatial information. Three gradient coils (one for each axis) provide a local change in the magnetic field, thereby changing the resonant frequency and the output signal. For example, a linear gradient field in the z -direction causes the resonant frequency to vary as a function of longitudinal position:

$$\omega = \gamma B = \gamma(B_0 + zG_z) \quad (2.24)$$

Therefore, a given RF pulse will only be able to resonate with particles in the transverse plane, orthogonal to the gradient field.

Gradient fields allow the data processing unit to differentiate among slices, but the exact location of a particle is still unknown. One possible solution is to use all three gradient coils simultaneously to isolate a point. However, this is not possible because multiple points can still share the same resonant frequency. Frequency and phase encoding, discussed later, are two mechanisms for encoding spatial information that allow isolation of a point in space in a reasonable amount of time.

2.2 Introduction to Signal Localization

A simple method has been suggested to generate a measureable signal from the subject. A large homogeneous field is applied to uniformly align most of the nuclear spins and an RF pulse is used to establish resonance with those molecules that share the same precession, or Larmor, frequency. This method does not allow the selection of one region of the subject for study; an RF pulse is only frequency selective and all regions sharing the same Larmor frequency are excited in the same manner. If, for example, the doctor would like to see what the problem in the liver is rather than in the heart, he needs to selectively excite the liver area to differentiate between both regions' local magnetic moments of the same material. The methods used to selectively excite regions of the subject and the techniques which allow distinction among received signal components' origins are discussed.

There are three methods used in signal localization. The first, *slice selection*, selectively excites a slice of the subject to image. The slice can be of arbitrary orientation as determined by the gradient coils' direction and relative strength. Once a two-dimensional slice is selected, the resultant signal must be processed to determine which part of the slice is sending each component of the signal. The remaining two methods, *frequency* and *phase-encoding*, are used for this purpose.

2.2.1 Selective Excitation

Slice selection is the first step in signal localization. In order to allow the RF pulse to be spatially selective rather than only frequency selective, the resonant frequency must be position-dependent. A gradient field produced by the three gradient coils and the shaped RF pulse are required. The gradient coils produce a linearly varying field which results in resonant frequency varying linearly along the direction of the gradient, or perpendicular to the selected slice, according to

$$\omega(\vec{r}) = \omega_o + \gamma \vec{G} \cdot \vec{r} \quad (2.25)$$

where \vec{G} denotes the gradient and \vec{r} , the position. The gradient field is applied during RF excitation in addition to the homogeneous B_o field. The direction of the field remains in the same direction as B_o but the amplitude of the field now varies along the direction of the gradient. Figure 2.4 shows how a slice along the μ_g direction is selected with the gradient \vec{G} . The RF pulse is now designed to make use of the slice-selection capability provided by the gradient field. In its general form, an RF pulse is given as $B_1(t) = B_1^e(t)e^{-i\omega_{rf}t}$, where $B_1^e(t)$ is the pulse envelope and ω_{rf} the excitation frequency. The simplest method used to design the RF pulse is the Fourier method. In order to simplify the following expressions, the gradient is assumed to lie in the z -direction and can be written as $\vec{G} = G\vec{k}$, where \vec{k} is the unit vector in the z -direction. The boxcar function in equation (2.26) can be used to describe the spatial selection of this slice in the $x - y$ plane.

$$p_s(z) = \Pi\left(\frac{z - z_o}{\Delta z}\right) \quad (2.26)$$

The thickness of the slice is given by Δz and it is centered at $z = z_o$. As a result of the gradient field, the Larmor frequency at position z is given in equation (2.25). Like field strength it varies linearly with z -position. The chosen slice can be represented as a boxcar function in frequency as well.

$$p(\omega) = \Pi\left(\frac{\omega - \omega_c}{\Delta\omega}\right) \quad (2.27)$$

Here $\Delta\omega = \gamma G_z \Delta z$ is the slice thickness and ω_c is the frequency at the center of the slice. The Fourier method assumes $B_1(t)$, the RF pulse, is related to $p(\omega)$, the frequency

slice, according to

$$B_1(t) \propto \int_{-\infty}^{\infty} p(\omega) e^{-i\omega t} d\omega \quad (2.28)$$

$$\propto \Delta f \text{sinc}(\pi \Delta f t) e^{-i\omega_c t} \quad (2.29)$$

Where the Fourier transform of the boxcar function is used to obtain the second proportion. Comparing this form with the general RF pulse given above, the excitation frequency, ω_{rf} , is given by ω_c , the frequency at the center of the slice, and the envelope function, $B_1^e(t)$, is given below.

$$B_1^e(t) = A \text{sinc}(\pi \Delta f t) \quad (2.30)$$

A is a constant determined by the flip angle, α , according to $\alpha = \int_0^{\tau_p} \gamma B_1^e(t) dt$, where τ_p is the duration of the RF pulse. The RF pulse will, in practice, differ from equation (2.28) because it will be finite in duration, lasting only τ_p seconds, and it will be shifted in time to start at (or after) $t = 0$. The envelope function of equation (2.30) can be rewritten to reflect this time shift.

$$B_1^e(t) = A \text{sinc}\left[\pi \Delta f \left(t - \frac{\tau_p}{2}\right)\right], \quad 0 \leq t \leq \tau_p \quad (2.31)$$

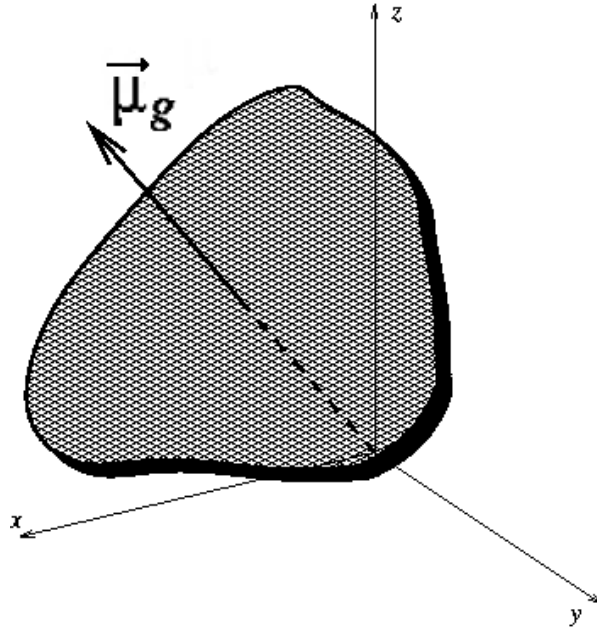


Figure 2.4: An arbitrary slice in the μ_g direction.

Ignoring pulse truncation effects and assuming the RF pulse is symmetric about $t = \tau_p/2$, the slice-selection profile of equation (2.26) can also be rewritten to reflect the effects of a shifted envelope function. The standard Fourier relationship between a shift in time and resulting phase shift is used.

$$p(z) = \Pi\left(\frac{z - z_o}{\Delta z}\right) e^{i\gamma G_z(z - z_o)\tau_p/2} \quad (2.32)$$

The Bloch equation approach is another, superior method of designing the RF pulse.

The linear phase shift $e^{i\gamma G_z(z - z_o)\tau_p/2}$ introduced across the slice by the slice-selective gradient results in signal-loss due to dephasing. The constituents of the bulk-magnetization vector acquire differing phase shifts according to their position. The phase shift can be removed by applying a refocusing gradient in a process known as *post-excitation rephasing*. Using the Fourier method, an acceptable form of the rephasing gradient can be found. Denoting the rephasing gradient as $G_{r,z}$, the phase at position z during the rephasing period is given by equation (2.33). The rephasing gradient simply adds the same type of phase shift as the slice-encoding gradient.

$$\phi(z, t) = \gamma G_z(z - z_o)\frac{\tau_p}{2} + \gamma G_{r,z}(z - z_o)(t - \tau_p) \quad t \geq \tau_p \quad (2.33)$$

The phase at the end of a rephasing period of duration τ_r is

$$\phi(z, t = \tau_p + \tau_r) = \gamma G_z(z - z_o)\frac{\tau_p}{2} + \gamma G_{r,z}(z - z_o)(\tau_r) \quad (2.34)$$

In order to cancel the phase shifts, $\phi(z, t = \tau_p + \tau_r)$ is set to 0, yielding the following relation between the rephasing gradient strength and duration.

$$G_{r,z}\tau_r = -\frac{1}{2}G_z\tau_p \quad (2.35)$$

Therefore, picking either the duration or the strength of the rephasing gradient will yield the other value. The effectiveness of this method in rephasing the signal constituents is limited because the dephasing amount is an approximate value. The actual amount of dephasing which occurs depends on the particular RF pulse and can be more precisely calculated using the Bloch equation. The necessary rephasing gradient can then be calculated using the method outlined above.

In addition to dephasing due to pulse time shift, pulse truncation should also be considered. Since the RF pulse is limited in time, the actual slice selection will deviate from the boxcar function. This will be further addressed when slice thickness is discussed. Assuming truncation to be a square filter applied to the infinite sinc function of $B_1^e(t)$, the selection profile in *frequency* is given by a convolution formula:

$$\hat{p}(\omega) = \Pi\left(\frac{\omega - \omega_c}{\Delta\omega}\right) * \text{sinc}\left[\frac{1}{2}(\omega - \omega_c)\tau_p\right] \quad (2.36)$$

This formula also neglects the phase shift previously considered.

2.2.2 Frequency Encoding

Frequency encoding is another method used to excite spins in specific regions. Like the process of slice selection, frequency encoding makes use of the gradient coils and forces the precessional frequency to be a function of spatial location. In contrast, however, this method applies a gradient during signal detection rather than during the RF excitation periods.

An example of frequency encoding in an arbitrary direction, denoted by $\vec{\mu}_{fe}$, is considered. In addition to the large, homogeneous magnetic field, \vec{B}_o , assume there is a linear gradient in the $\vec{\mu}_{fe}$ -direction during signal detection. Then the Larmor frequency is given as a function of this direction as follows:

$$\omega(\vec{\mu}_{fe}) = \omega_o + \gamma \vec{G}_{fe} \cdot \vec{r} \quad (2.37)$$

where $\vec{G}_{fe} = (G_x, G_y, G_z)$ is the frequency-encoding gradient in the $\vec{\mu}_{fe}$ direction. This is the same as equation (2.25) given for slice selection. For an object with spin distribution $\rho(\vec{r})$, the local signal generated by the spins in a volume $d\vec{r}$ about point r is shown in equation (2.38).

$$dS(\vec{r}, t) = \rho(\vec{r}) d\vec{r} e^{-i\gamma(B_o + \vec{G}_{fe} \cdot \vec{r})t} \quad (2.38)$$

The received signal is the integral of this local signal over the entire object 2.39

$$\begin{aligned} S(t) &= \int_{object} dS(\vec{r}, t) = \int_{-\infty}^{\infty} \rho(\vec{r}) e^{-i\gamma(B_o + \vec{G}_{fe} \cdot \vec{r})t} d\vec{r} \\ &= \left(\int_{-\infty}^{\infty} \rho(\vec{r}) e^{-i\gamma(\vec{G}_{fe} \cdot \vec{r})t} d\vec{r} \right) e^{-i\omega_o t} \end{aligned} \quad (2.39)$$

The signal $e^{-i\omega_o t}$ can be removed after demodulation. It is important to note that *all* points along the line $\vec{G}_{fe} \cdot \vec{r} = \text{constant}$ share the same Larmor frequency because they experience the same magnetic field. As shown in Figure 2.5 these define isofrequency lines perpendicular to the frequency-encoding gradient. The third signal localization scheme, phase encoding, is necessary to distinguish the points along the isofrequency lines.

2.2.3 Phase Encoding

Phase encoding is the same as frequency encoding except in when the gradient is applied. In *phase*-encoding the auxiliary gradient field is turned on after the slice-selective RF pulse and gradient but prior to signal detection and the frequency-encoding gradient. During the phase-encoding gradient, the object is, in reality, frequency-encoded; however, upon termination of the phase-encoding gradient, the signals are left with different phases according to their location in the subject. This is illustrated in Figure 2.6.

Usually the object is linearly phase-encoded in the y -direction, however, an example of the general case of an arbitrarily-directed phase-encoding gradient, \vec{G}_{pe} , is considered. A gradient, $\vec{G}_{pe} = (G_x, G_y, G_z)$, is applied for a short time, T_{pe} , known as the phase-encoding interval, after the RF pulse has been applied. The local signal under the influence of this gradient, in addition to the large, homogeneous field \vec{B}_o , is given in equation (2.40).

$$dS(x, t) = \begin{cases} \rho(x) e^{-i\gamma(B_o \vec{k} + \vec{G}_{pe} \cdot \vec{r})t} & 0 \leq t \leq T_{pe} \\ \rho(x) e^{-i\gamma \vec{G}_{pe} \cdot \vec{r} T_{pe}} e^{-i\gamma B_o \vec{k} t} & T_{pe} \leq t \end{cases} \quad (2.40)$$

While the gradient is on, $0 \leq t < T_{pe}$, the signal is the same as a frequency-encoded signal. For $t \geq T_{pe}$, however, only the initial phase angle imparted to the signals is present. It is given by $\phi(\vec{r}) = -\gamma \vec{G}_{pe} \cdot \vec{r} T_{pe}$. The total signal is the integral of the local signal in 2.40 over the entire object:

$$S(t) = \int_{object} dS(\vec{r}, t) \quad (2.41)$$

$$= \left[\int_{object} \rho(\vec{r}) e^{-i\gamma \vec{G}_{pe} \cdot \vec{r} T_{pe}} d\vec{r} \right] e^{-i\omega_o t} \quad (2.42)$$

As in frequency encoding, the carrier signal, $e^{-i\omega_o t}$, is removed after demodulation.

2.2.4 An Introduction to k -space

By considering the relationship between spatial encoding and the Fourier transform an alternate representation for encoding, known as k -space, can be arrived at. By noting

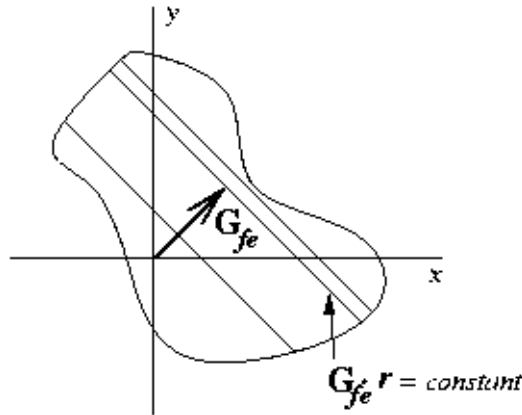


Figure 2.5: Isofrequency lines perpendicular to the frequency-encoding gradient \vec{G}_{fe} .

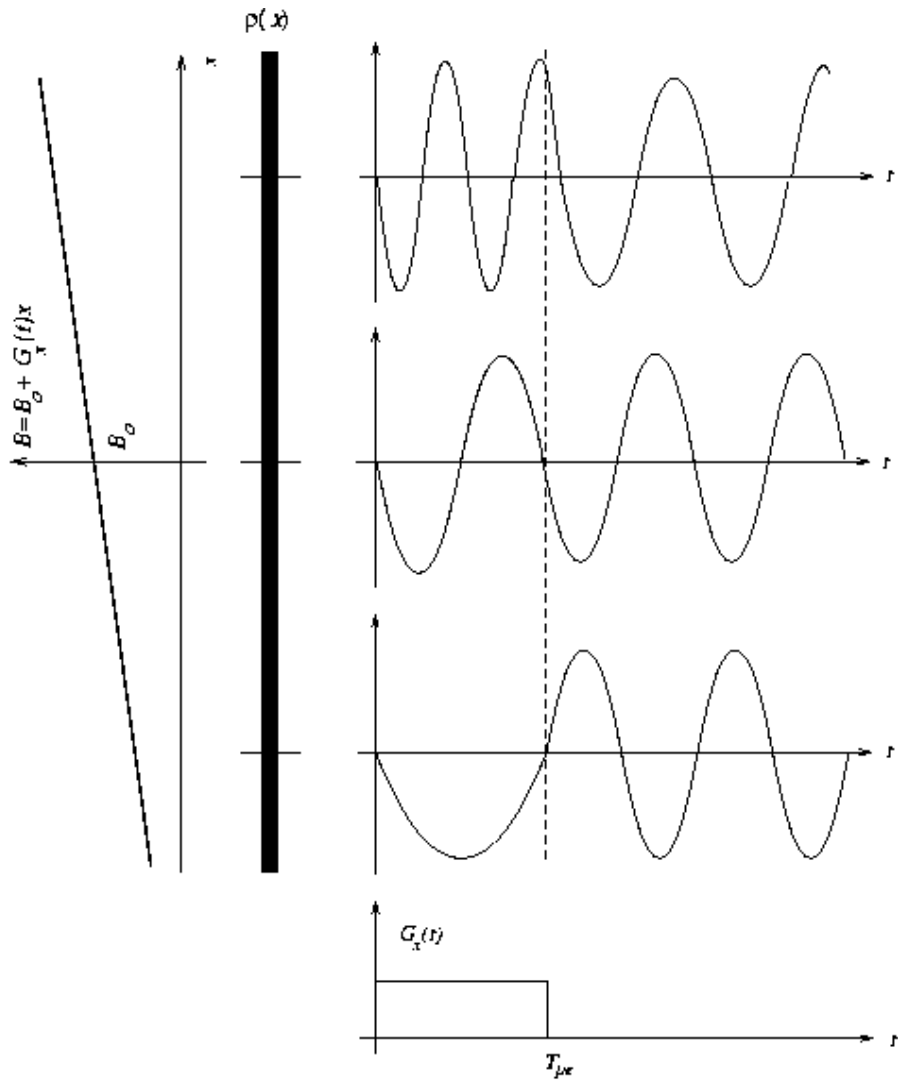


Figure 2.6: Phase-encoded signals from a one-dimensional object. The phase is dependent on the pre-frequency encode time T_{pe} .

that the signal in the last equality of equation (2.39), after removal of the carrier signal, is a Fourier transform this simpler notation can be used to describe complex sampling and encoding schemes. The Fourier transform of $\rho(x)$, a general function but in this case the spin density, is given in equation (2.43).

$$S(\vec{k}) = \int_{\text{object}} \rho(\vec{r}) e^{-i2\pi\vec{k}\cdot\vec{r}} d\vec{r} \quad (2.43)$$

The use of k -space notation in frequency encoding is first considered. Observe that the exponential term in this equation yields the following relation when equated with the exponential term, $-i\gamma(\vec{G}_{fe} \cdot \vec{r})t$, of equation (2.39).

$$\vec{k} = \begin{cases} \bar{\gamma}\vec{G}_{fe}t & \text{FID signals} \\ \bar{\gamma}\vec{G}_{fe}(t - T_E) & \text{echo signals} \end{cases} \quad (2.44)$$

The gradient provides a mapping from the time domain to k -space. The signal $S(\vec{k})$ exists only for those points in k -space where sampling takes place as defined by the gradient fields. The plot of these points in k -space is called the *sampling-trajectory*.

As a simple example, consider a constant gradient in the x -direction. In this case, $k_x = \bar{\gamma}G_x t$, which is seen as a straight line along the k_x -axis in part *a* of Figure 2.7. If the gradient is in both the x and y directions, the line plotted in the $k_x - k_y$ plane would be at an angle $\phi = \arctan \frac{k_y}{k_x} = \arctan \frac{G_y}{G_x}$. This is shown in part *b* of Figure 2.7. Straight lines result only in the case of constant gradients. In the more general case of a time-varying gradient, \vec{k} , now a function of time, is given by equation (2.45).

$$\vec{k}(t) = \bar{\gamma} \int_0^t \vec{G}_{fe}(\tau) d\tau \quad (2.45)$$

Phase encoding yields a similar result in k -space. Once again, by comparing the exponent in the Fourier transform of equation (2.43), with the exponent in the phase-encoded signal of equation (2.41), $-i\gamma(\vec{G}_{pe} \cdot \vec{r})T_{pe}$, a relationship between the gradients and \vec{k} is established. In this case, $\vec{k} = \bar{\gamma}\vec{G}_{pe}T_{pe}$. Since the phase-encoding gradient is applied prior to signal detection, it leaves a constant phase value on each component of the signal according to its location in the object. Therefore, in phase encoding, \vec{k} has a constant value for a given phase encode interval, T_{pe} , and gradient, G_{pe} , whereas in frequency encoding, \vec{k} is given as a function of time. This implies phase encoding affects only the starting position of the sampling trajectory in k -space while frequency encoding determines the shape and form of the trajectory. In the more general case of a time varying phase encode gradient \vec{k} is given as

$$\vec{k} = \bar{\gamma} \int_0^{T_{pe}} \vec{G}_{pe}(\tau) d\tau \quad (2.46)$$

This is similar to the expression given in the case of a time varying frequency encoding gradient except it is a constant value. Another important point that can be extracted from equation (2.46) is that as long as the area under the phase encode gradient remains the same, the starting point of the trajectory will remain the same. The shape of the gradient is inconsequential in this regard. During the phase encode periods, however, \vec{k} varies according to $\frac{d\vec{k}(t)}{dt} = \gamma\vec{G}(t)$ and k moves from the origin to its next trajectory location according to this expression. Therefore, a constant gradient yields a constant speed and a variable gradient, a variable speed.

2.2.5 How it all comes together: Pulse Sequences and the Scanner

A complete example of an MRI scan involving all three signal localization methods will be considered to clarify the mathematical explanation already given. Figure 2.8 depicts the setup of the scanner and the orientation of the z -coils. The large, homogeneous field, B_0 , lies in the z -direction. An image of a slice in the $x-y$ plane is desired. First, using a slice selective gradient in the z -direction, the slice is selectively excited upon application of the tailored RF pulse. The location along the z -axis which is chosen varies according to Figure 2.9; in this example, it is wherever the local field has the same strength as the homogeneous field. It is easily seen by changing the relative strengths of the auxiliary fields created by the gradient coils in the z -direction the slice moves along the z -axis. As briefly discussed, the RF has finite duration in time which implies it possesses more

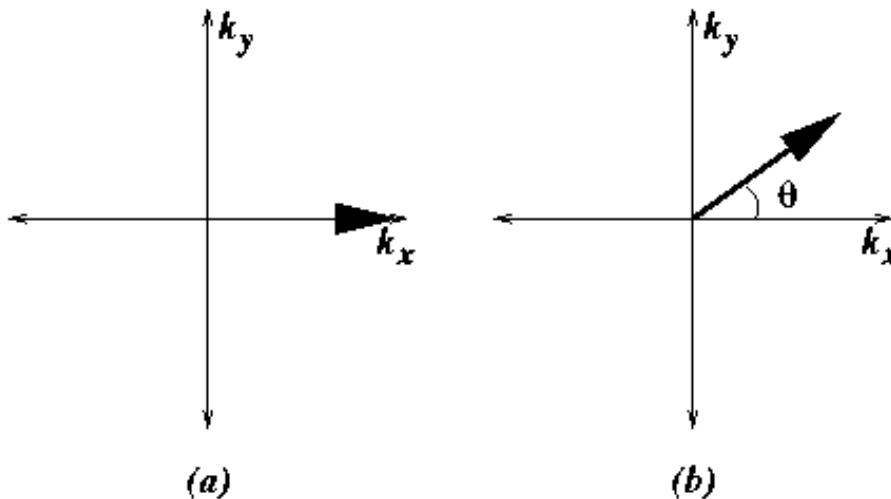


Figure 2.7: The sampling trajectories of two simple gradients.

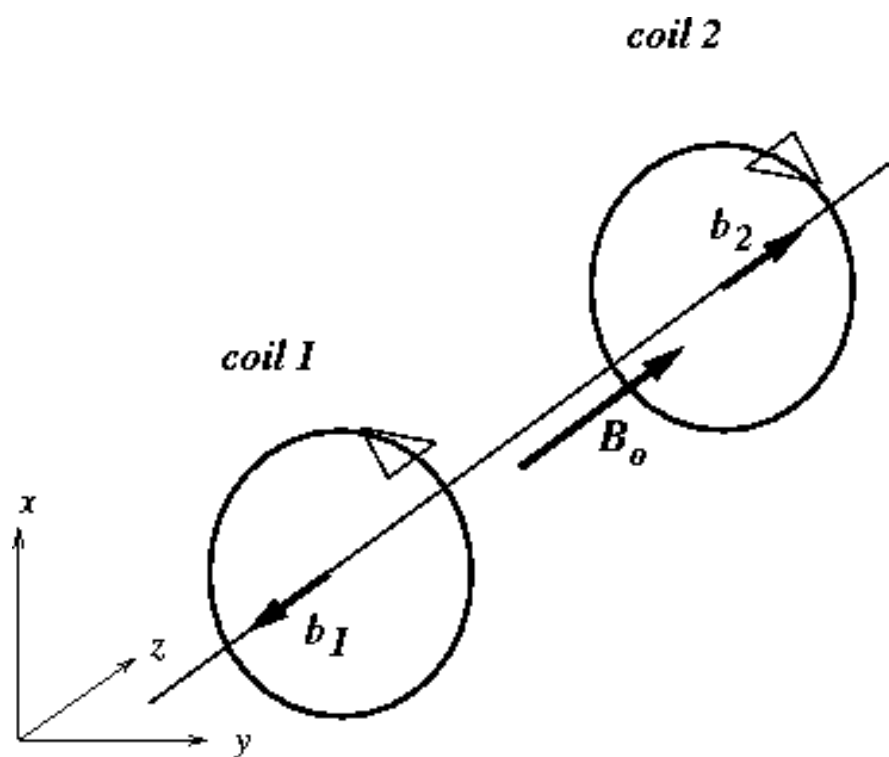


Figure 2.8: Setup of scanner and orientation of the z -gradient coils.

than one frequency. Since it excites protons precessing at the same frequency, all of those protons sharing the frequencies it possesses will be excited. The bandwidth of the pulse can be changed through varying its duration; the longer it lasts in time the fewer frequencies it will possess. The extreme case being an infinite RF pulse in time possessing a single frequency. As a result of this property, a slice with finite thickness will be selected as protons on either side of z_0 , the center of the slice, will share similar frequency and be excited by the RF pulse. In practice it is more common to vary the strength of the slice-selective gradient, the magnitude of which determines how rapidly the precessional frequency changes along the z -axis, rather than vary the duration of the RF pulse. Both techniques have the same effect; a large gradient ensures neighboring protons precess at largely different frequencies. Figure 2.10 illustrates the relationship between slice thickness, RF pulse bandwidth, and gradient strength. Now that the slice has been selected with the simultaneously applied RF pulse and slice-selection gradient, the echo signal emanating from the slice must be further broken down to convey information about regions within the slice. In order to facilitate doing so, the slice is divided into a two-dimensional grid of *pixels*. In image reconstruction, each pixel is assigned a grayscale value corresponding to the signal strength emanating from its location. Since the slice has a finite width, rather than being two-dimensional areas, the basic units become three-dimensional volumes called *voxels*. In the example considered

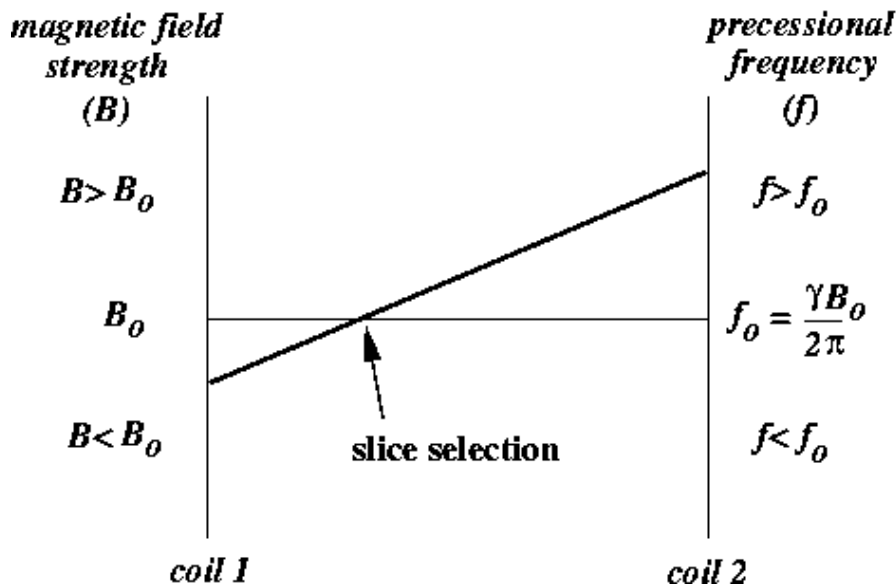


Figure 2.9: The slice along the z -axis that is selected depends on the gradient produced by coil 1 and coil 2.

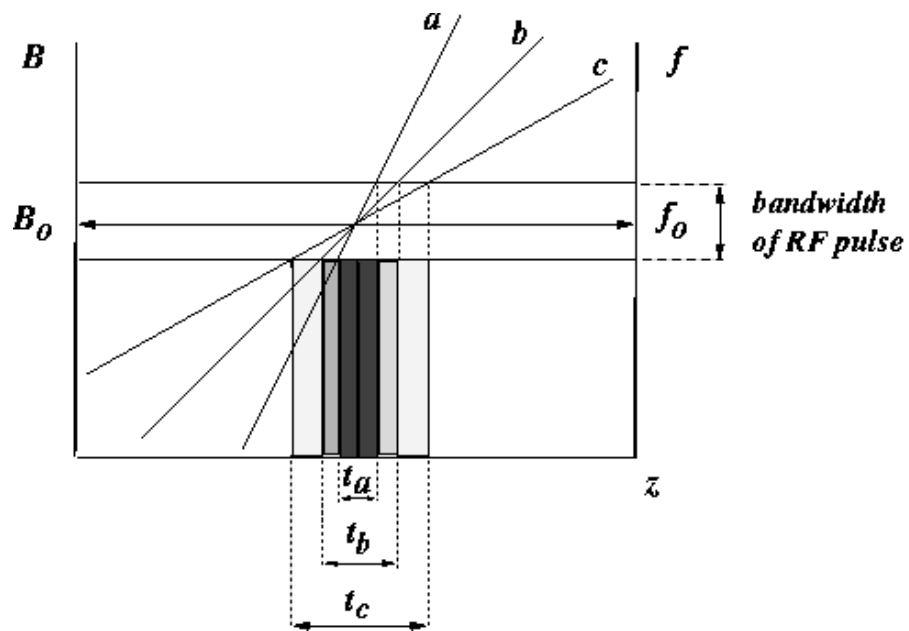


Figure 2.10: The relationship between slice thickness, RF pulse bandwidth, and gradient strength. The strength of the gradient determines the thickness of the slice selected by the RF pulse.

here, there will be 256 pixels in the x -direction and 128 pixels in the y -direction, see Figure 2.11. This means there are 32,768 total pixels or corresponding voxels.

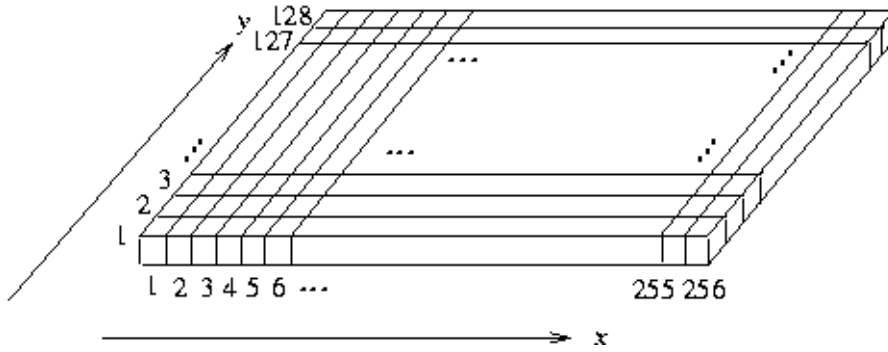


Figure 2.11: The selected slice is broken up into a 256 by 128 grid of voxels.

Frequency and phase encoding will be employed along the x and y -directions, respectively. The x and y gradient coils produce auxiliary fields that *vary* along their respective directions but whose *direction* is aligned with the homogeneous field in the z -direction. The coils shown in Figure 2.12.a and 2.12.b produce the gradients in the x and y directions, or a combination of both. As in slice selection, the relative strengths of the gradient coil fields change the local magnetic field experienced by different locations within the object.

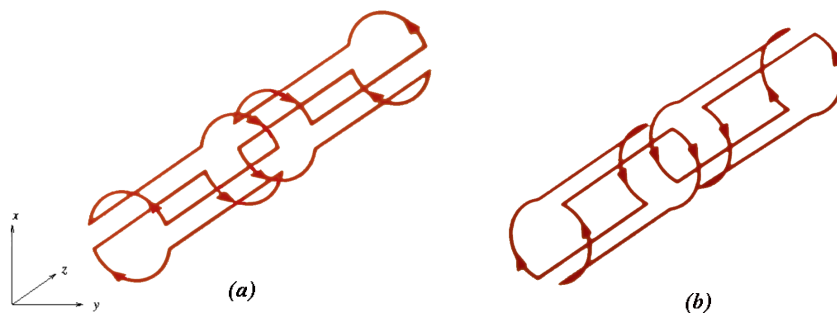


Figure 2.12: a)The x gradient coil. b)The y gradient coil.

While the echo signal is detected, the frequency encoding gradient, G_x , is applied. Each pixel location along the x -axis will experience a slightly different field and will

precess at a corresponding frequency. Those in the same x location, however, will precess at the same frequency yielding the previously discussed isofrequency lines shown in Figure 2.5. A one-dimensional Fourier transform applied to the signal received from the excited slice distinguishes each component according to its frequency or origin along the x -axis. The signal obtained from one x location will be the sum of all 128 pixels along the y -axis that share the same local magnetic field. Using a one-dimensional Fourier transform and a single MR signal it is impossible to extract position information along the y -direction.

In order to obtain the remaining y -direction information, phase encoding is used. After the slice has been selected but before frequency encoding and echo detection, a phase-encode gradient is briefly applied in the y -direction. During gradient application each position along the y -direction will experience a different local field strength and will precess at the corresponding frequency. After the phase-encode gradient is removed, all positions will resume the same, or nearly the same, precessional frequency, but they will have acquired a different phase. Those having precessed at higher frequency acquiring larger phases and lower frequencies, smaller. Post phase-encode, the phase of the signal will vary linearly according to y -location.

Fourier analysis cannot distinguish the phases of added signals of the same frequency. Therefore, the signals from each x location which share the same frequency but differ in phase cannot be further decomposed with this technique. However, using phase information and many echos, signal origin in the y -direction can be determined. Each pulse sequence uses a phase-encoding gradient of different strength or duration. Since, in this example, there are 128 locations in the y direction to distinguish, 128 different pulse sequences must be used. The echo signal obtained with each pulse sequence is called a *view*.

Summarizing, in the x -direction frequency distinguishes the signal origins and in the y -direction the phase given to each of the 128 echos distinguishes them. Each of the 128 resulting echo signals are the samples 256 corresponding to the number of voxels in the x -direction. As will be discussed in the next section, Image Reconstruction, a two-dimensional Fourier Transform of these samples yields the 256 by 128 pixels image.

The pulse sequence used in this process is shown in Figure 2.13. As mentioned in section 2.1 the echo signal is obtained through the application of the 90° and 180° RF pulses. During these RF pulses, the slice selective gradient in the z -direction is applied. After the slice is selected, the phase-encode gradient in the y -direction is applied between RF pulses. Finally, the frequency-encoding gradient is applied in the x -direction during echo detection. The process is repeated for each desired y pixel location, or in this example, 128 times. In the pulse sequence diagram of figure 2.13 the timing is not to scale; the time between the echo and the 180° pulse, or echo time, should be a larger percentage than the repetition time, or time between subsequent 90° pulses.

The previous example employs the most basic imaging techniques. Its principles of signal localization, however, can be applied with more complex, superior imaging schemes. These include imaging more than one slice simultaneously, employing three-dimensional Fourier analysis on the detected signals, and using flip angles less than 90° . Methods such as these improve scanning efficiency and decrease the amount of time the patient is required to be in the scanner.

2.3 Image Reconstruction

The discrete measured data obtained from various sampling methods must be converted into a continuous image function. It is then necessary to digitize this continuous function for computational and display purposes. Figure 2.3 illustrates this process. Often, discretization is accomplished by performing a FFT on the sufficiently sampled image function.

In this section, basic image reconstruction procedures are discussed. The discretization of the image function, $I(\vec{r})$, is only briefly described. After an overview of the problem, two fundamental reconstruction procedures are described. Finally, a generalized view of all image reconstruction procedures is posed.

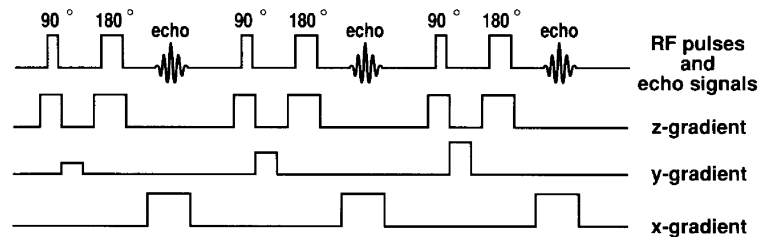


Figure 2.13: Basic pulse sequence to image slice in xy -plane

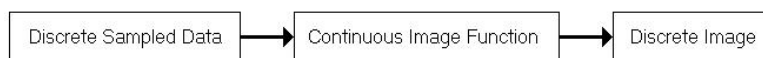


Figure 2.14: Steps of the Image Reconstruction Procedure

2.3.1 General Image Reconstruction

Image reconstruction consists of recovering an image function, $I(\cdot)$, which is consistent with the measured k -space data, S , according to:

$$S = T\{I\} \quad (2.47)$$

where T in this application is any spatial information encoding scheme, such as the Fourier transform. Any function satisfying equation (2.47) is called a feasible reconstruction. Alternatively, I is a feasible reconstruction if any object based on I , subject to the same experimental methods, T , yields the same measured data, S .

Theoretically, as long as T is invertible an image function can be recovered using the inverse transform:

$$I = T^{-1}\{S\} \quad (2.48)$$

In practice, however, since data space is finite, straightforward computation of $T^{-1}\{S\}$ is not possible. Alternate means, such as an approximation to the inverse, are necessary to compute I .

Three issues involved with the image reconstruction procedure are existence, uniqueness, and stability. An image function always exists for a given set of measured data since the data are from a physical object. The uniqueness and accuracy of such an image function, however, depends on how data space is sampled and measurement noise. If the sampling is finite, which is always the case in practice, the system is underdetermined and there are many feasible image functions that are consistent with the measured data S . Of these, one image function is chosen based on some optimal criteria. Finally, stability relates perturbations in the data, ΔS , to possible error in the reconstructed image, ΔI , where:

$$S + \Delta S = T\{I + \Delta I\} \quad (2.49)$$

Stability analysis of the encoding scheme, T , seeks to evaluate what error in the image, ΔI , is caused by a perturbation in the data, ΔS . The scheme may tend to exaggerate small perturbation in the data resulting in unacceptable error in the reconstructed image. In fact, it can be shown that, in some cases, a negligible error in the data can produce an arbitrarily large error in the image [4]. If the operator, or spatial encoding scheme, has this property, as it usually does, and due to finite data sampling, true image reconstruction is actually impossible [4]. Sufficient image quality can, however, be obtained by proper selection of the image function and subsequent compensation for known deviations from the true image function.

2.3.2 Fourier Transform Reconstruction

The most straightforward method of reconstruction uses the Fourier transform in order to recover the image function, $I(\vec{r})$. It is generally used in the case of Cartesian sampling

of the data space but, as will be shown later, can be applied to any sampling pattern. The measured data S are given by

$$S(\vec{k}_n) = \int I(\vec{r}) e^{-i2\pi\vec{k}_n \cdot \vec{r}} d\vec{r}, k_n \in D \quad (2.50)$$

where D is the set of k -space samples. In theory, to find $I(\vec{r})$, only the inverse DFT on a set of uniformly spaced samples, S , needs to be computed. As stated before, however, in the case of finite sampling, this is not possible. An alternate computation follows.

The Fourier transform's property of separability is invoked and the evaluation of the inverse considered in one dimension for simplicity. Equation (2.50) then becomes

$$S(k_n) = \int_{-\infty}^{\infty} I(x) e^{-i2\pi k_n x} dx \quad (2.51)$$

with uniform sampling of k -space, $D = \{k_n = n\Delta k, n = \dots, -2, -1, 0, 1, 2, \dots\}$. Using $k_n = n\Delta k$, equation (2.51) can be rewritten as

$$S[n] = S(n\Delta k) = \int_{-\infty}^{\infty} I(x) e^{-i2\pi n\Delta k x} dx \quad (2.52)$$

Reconstruction of $I(x)$ can be achieved through use of the Fourier series. Each sample, $S[n]$, may be thought of as a Fourier series coefficient in the following equation derived from the Poisson formula:

$$\sum_{n=-\infty}^{\infty} S[n] e^{i2\pi\Delta k x} = \frac{1}{\Delta k} \sum_{n=-\infty}^{\infty} I(x - n) \quad (2.53)$$

On the left side, the fundamental frequency of the series is Δk . The right side is a periodic replication of $I(x)$ with frequency Δk . There are two cases to consider in evaluating equation (2.53), one of infinite sampling and the other more practical case of finite sampling. The simpler case of infinite sampling is considered first.

Since D , the set of all points in k -space where measurements are taken, ranges from $-\infty$ to ∞ , the Fourier series in equation (2.53) is defined. The image function $I(x)$ can be extracted from its periodically extended form if the frequency of its replication satisfies the Nyquist sampling theorem. The image can be reconstructed by picking out any one of its replications as long as $\Delta k < 1/W_x$, where W_x is the support of $I(x)$. The mathematical form of such a selection is:

$$I(x) = \Delta k \prod\left(\frac{x}{\Delta k}\right) \sum_{n=-\infty}^{\infty} S[n] e^{i2\pi\Delta k x} \quad (2.54)$$

The windowing function, \prod , is often omitted for convenience since $I(x)$ is, in practice, only evaluated in this interval. The case of finite sampling is more complex. In this case,

$$D = \{n\Delta k, -N/2 \leq n \leq N/2\} \quad (2.55)$$

A unique image function does not exist because D yields a set of data which is insufficient to define the series in equation (2.53). Therefore, a feasible image function is not unique. If $I(x)$ is a feasible reconstruction, then $\hat{I}(x) = I(x) + e^{i2\pi m\Delta kx}$ is also a feasible reconstruction for any $|m| > N/2$. In equation (2.54), $I(x)$ is a feasible reconstruction if the unmeasured coefficients take finite arbitrary values. That is,

$$I(x) = \Delta k \sum_{n=-N/2}^{N/2-1} S[n]e^{i2\pi n\Delta kx} + \sum_{n < N/2; n \geq N/2} c_n e^{i2\pi n\Delta kx} \quad (2.56)$$

is a feasible reconstruction for arbitrary c_n satisfying $\sum |c_n|^2 < \infty$. The choice of c_n is application-dependent; often, a minimum-norm constraint is used. In this case, the c_n 's are chosen to minimize $\int |I|^2 dx$. Application of Parseval's theorem makes all of the unmeasured c_n 's then go to zero yielding the following truncated Fourier series known as the *Fourier reconstruction formula*:

$$I(x) = \Delta k \sum_{n=-N/2}^{N/2-1} S[n]e^{i2\pi n\Delta kx}, |x| < \frac{1}{\Delta k} \quad (2.57)$$

Truncation causes ringing artifacts in the reconstructed image. In order to minimize ringing, a windowing function, w_n , is usually applied to the data samples to smooth the decay at $n = \pm N/2$. However, the image function will no longer produce the same measured data, $S[n]$, when the spatial encoding operator, T , is applied to it. Rather, it produces $w_n S[n]$. The known deviation caused by the windowing function is then compensated for in the image.

At this point, assuming the continuous reconstructed image function, $I(x)$, is available, it needs to be digitized for computation and display. As stated earlier, this will not be covered in depth. In short, a DFT is applied to $I(x)$; usually this is accomplished via the FFT in order to improve computational efficiency.

2.3.3 Radon Transform Sample Reconstruction

Radon transform reconstruction is related to Fourier reconstruction through the projection-slice theorem. Although MRI does not directly sample Radon space by taking projections of the object to be imaged, data to which this technique is applied are assumed to be from this space. It is generally used to reconstruct radially sampled k -space data. In general, the Radon transform sample reconstruction problem can be stated as follows. Given the projection data

$$P(p, \vec{\mu}) = R\{I\} = \int_{R_n} I(\vec{r}) \delta(p - \vec{\mu} \cdot \vec{r}) d\vec{r}, (p, \vec{\mu}) \in D \quad (2.58)$$

where D is the set of points where measurements are taken, the objective is to find $I(\vec{r})$. Recovering $I(\vec{r})$ involves taking the inverse Radon transform. As in the case of Fourier reconstruction, the uniqueness of the reconstruction depends on whether or not Radon space is sampled in its entirety. Finite sampling once again mandates a non-unique image function. The projection-slice theorem is first presented to develop understanding of the problem and then some practical considerations involved in this method are addressed.

The projection of a spatial function, $u(x, y)$, along a line oriented at θ degrees from the x -axis is denoted by g_θ . This is depicted in Figure 2.15. The projection-slice theorem states that the Fourier transform of g_θ is a one-dimensional slice of $U(\omega_x, \omega_y)$ along the line oriented at θ degrees or $G_\theta(\omega) = U(\omega \cos \theta, \omega \sin \theta)$. Since one slice of U is recovered

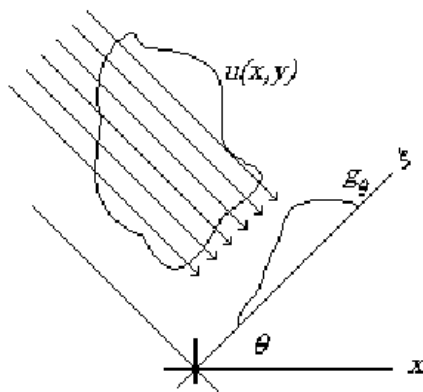


Figure 2.15: Projection of $u(x, y)$ along line ζ

at each angle, the original image function, u , can be obtained by measuring g_θ for all θ . Reconstruction is broken down into two steps: convolution and back-projection [2]. The inverse Fourier transform of $u(x, y)$, written as a function of (ω, θ) is

$$u(x, y) = \frac{1}{2\pi} \int_{\theta=0}^{\pi} \left[\frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} U(\omega \cos \theta, \omega \sin \theta) e^{i\omega(x \cos \theta + y \sin \theta)} |\omega| d\omega \right] d\theta \quad (2.59)$$

where $|\omega|$ is the absolute value of the Jacobian resulting from the coordinate transformation. Using the projection-slice theorem $U(\omega \cos \theta, \omega \sin \theta)$ can be replaced by $G_\theta(\omega)$. Rewriting the term in brackets with this substitution and ζ replacing $(x \cos \theta + y \sin \theta)$:

$$p_\theta(\zeta) = \frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} |\omega| G_\theta(\omega) e^{i\omega\zeta} d\omega \quad (2.60)$$

p_θ is the inverse transform of the multiplication in frequency of $G_\theta(\omega)$ with a filter of frequency response $|\omega|$. Therefore, $p_\theta(\zeta)$ is the convolution of g_θ with the inverse

transform of the filter of frequency response $|\omega|$, or $p = g_\theta * h$, where $F\{h\} = |\omega|$. The first step of the Radon sample reconstruction method is convolution. Equation (2.59) is rewritten as a function of $p_\theta(\zeta)$:

$$u(x, y) = \frac{1}{2\pi} \int_{\theta=0}^{\pi} p_\theta(\zeta(x, y)) d\theta \quad (2.61)$$

It should be emphasized that p_θ are filtered projection data, not the directly measured projection data. The filtered data at a point q along the line ζ oriented at θ degrees from the x -axis are then uniformly projected back along the line perpendicular to ζ . Hence, the second step is back-projection. Each point q along ζ for every θ in $[0, \pi]$ is also back-projected in the same manner. The integral in equation (2.61) is computed over all of the back-projected data values corresponding to a particular q value, yielding one point in the original function, $u(x, y)$.

In practice, a finite number of samples are collected. This means the sampling angles are discrete and are separated by some $\Delta\theta$. Equation (2.61) then becomes

$$u(x, y) = \frac{1}{2} \sum_{i=0}^N p_{\theta_i}(x \cos \theta_i, y \sin \theta_i) \Delta\theta \quad (2.62)$$

where N is the number of sampling angles in $[0, \pi]$. It is evident each point $u(x_0, y_0)$ in the reconstructed image, u , is the average value of the back-projected, filtered data.

The filter in equation (2.60) must also be considered in practical applications. Often, approximate band-limited filters are used in place of the exact filter to reduce high-frequency noise. Commonly used approximations include the Ram-Lak or Shepp-Logan filters [4]. In higher dimensions the filter is not of the form $|\omega|$. However, it is also approximated with various filter designs to reduce high-frequency noise.

2.3.4 A Unified View of Image Reconstruction

King [3] poses an interesting view of image reconstruction, stating "...various imaging 'methods' are...minor algorithmic variations of a single, general, transform inversion of reciprocal-space data to reconstruct a spatial image." Such a conclusion is demonstrated by means of a derivation using physical parameters involved in the acquisition and measurement stages such as field strength and the magnetic moment vector. The detected signal can be represented as

$$S(t) = C \int m_\perp(\vec{r}, t) e^{-i2\pi\vec{q}\cdot\vec{r}} d\vec{r} \quad (2.63)$$

where C is a constant related to the receiver coil's Q and field-to-current ratio and the mean Larmor frequency and $m_\perp(\vec{r}, t)$ is the transverse component of the magnetic

moment vector. \vec{q} is defined as the product of the gradient field, G , with the Larmor frequency and the time variable, t . According to equation (2.63) the observed signal is proportional to the spatial Fourier transform, $\tilde{m}(\vec{q})$, of the magnetization vector, where $\tilde{\cdot}$ denotes the transform. Defining $\tilde{m}(\vec{q}) = \int m_{\perp}(\vec{r}, t) e^{-i2\pi\vec{q}\cdot\vec{r}} d\vec{r}$ and using the substitution $t = (\vec{q}/\gamma G)$, equation (2.63) becomes

$$\tilde{S}(\vec{q}) = S(t)|_{t=\vec{q}/\gamma G} = C\tilde{m}_{\perp}(\vec{q}) \quad (2.64)$$

In practice, the substitution of \vec{q} for t is accomplished by an “integrate-sample-and-digitize” sequence performed on $S(t)$. With sufficiently dense sampling, $\tilde{S}(\vec{q})$ can be inverted to produce an image of the magnetization vector.

Finite sampling, which truncates \vec{q} , effectively applies a band-limiting windowing function to $\tilde{S}(\vec{q})$. The step generalizing all image reconstruction procedures into one inverse Fourier transform rests on this windowing function. The data set obtained following mandatory windowing is

$$I(\vec{q}) = \tilde{H}(q)\tilde{S}(\vec{q}) \quad (2.65)$$

where $\tilde{H}(q)$ represents the windowing function. Final reconstruction of the image follows an inverse Fourier transform:

$$I(\vec{r}) = \int \tilde{H}(q)\tilde{S}(\vec{q})e^{i2\pi\vec{q}\cdot\vec{r}} d^3\vec{q} \quad (2.66)$$

The multiplication in reciprocal data space becomes a spatial convolution.

$$I(\vec{r}) = \int H(r - \vec{r}')m_{\perp}(\vec{r}')d^3\vec{r}' \quad (2.67)$$

The difference between various image reconstruction techniques lies only in how the data are truncated via the windowing function prior to inversion and what inversion technique is necessary for the given sampled data geometry. All images result from an inverse Fourier transform [3].

Relating this conclusion to the equations for Fourier transform and Radon transform reconstructions above, the windowing functions in either case are indeed a result of finite sampling. The difference between the two fundamental methods lies in the employed sampling geometry, which determines which coordinate system the inverse Fourier transform is applied in. The Fourier reconstruction technique uses Cartesian inversion via the FFT on Cartesian coordinate sampled data. The back-projection technique, via the projection slice theorem, uses a numerical Fourier inversion in polar coordinates on radially sampled data.

In this chapter, only the simple cases of 2-D Cartesian and polar sampling trajectories are considered. The general reconstruction formula found in equation (2.66), however, can be applied to more complex sampling patterns, such as spiral or 3-D Cartesian and polar trajectories, as well as various pulse sequence techniques, such as the spin echo technique.

Chapter 3

Motion and Magnetic Resonance Imaging

Practical applications of MRI typically do not involve stationary objects. Motion, as in still photography and motion picture photography as well, make it difficult to reconstruct a clear, sharp image of the desired object. In still photography, too slow a shutter speed can result in a blurred image; a camera resting on an unsteady support results in jittery video. In clinical MRI, the sources of motion include blood flow, respiratory motion, cardiac motion, and gross movements of the body. Artifacts due to these types of motion include blurring and ghosting, examples of which are shown in Figures 3.1 and 3.2. Since it is virtually impossible to eliminate these without putting the patient at great risk, the motion must be compensated for during the imaging process. In order to do so, the effects of motion on the imaging equations must be analyzed.

3.1 Effects of Motion on Image Reconstruction

Motion of the object during the readout causes additional phase shifts, in addition to the phase shift due to excitation by the MR signal. The original phase shift is given by

$$\phi = \gamma \int_0^t \vec{G}(\hat{t}) \cdot \vec{r}(\hat{t}) d\hat{t} \quad (3.1)$$

where $r = (x, y)$. Assume that the object is undergoing motion with constant velocity, and that the gradient signal is present only in the x -direction. Then, the total phase shift is given by

$$\phi = \gamma x \int_0^t G_x(\hat{t}) d\hat{t} + \gamma v_x \int_0^t \hat{t} G_x(\hat{t}) d\hat{t} \quad (3.2)$$

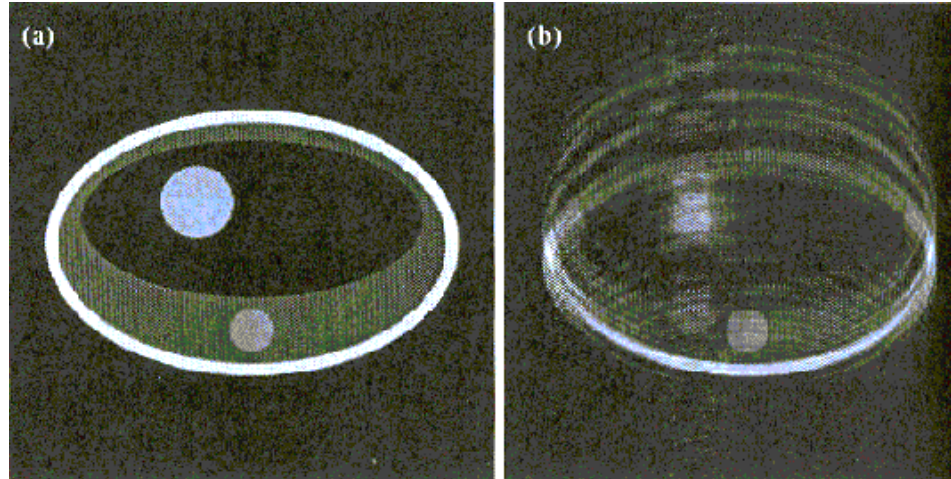


Figure 3.1: (a) Ideal reconstructed image (b) Image corrupted by blurring and ghosting

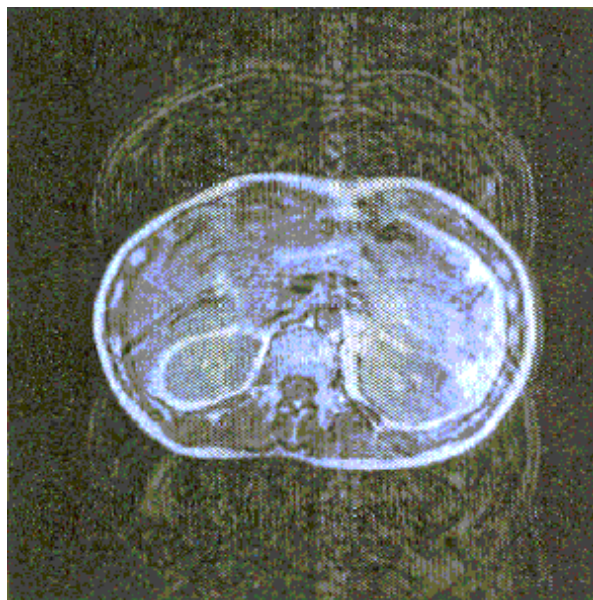


Figure 3.2: Image of abdominal cross-section displaying image artifacts

where v_x is the velocity of the object in the x -direction. The motion-introduced phase shift is therefore

$$\Delta\phi = \gamma v_x \int_0^t \hat{t} G_x(\hat{t}) d\hat{t} \tag{3.3}$$

which is dependent on the gradient structure. If the gradient is as shown in Figure 3.3, the motion-introduced phase shift is equal to

$$\Delta\phi = -\gamma \int_0^t G_x v_x \hat{t} d\hat{t} = -\frac{1}{2} \gamma G_x v_x t^2 \tag{3.4}$$

during the dephasing period, and during the data acquisition period, the phase shift is equal to

$$\Delta\phi = \frac{1}{2} \gamma G_x v_x (t^2 - 2\tau^2) \tag{3.5}$$

At the peak of the echo, moving spins are not completely rephased for this particular gradient waveform.

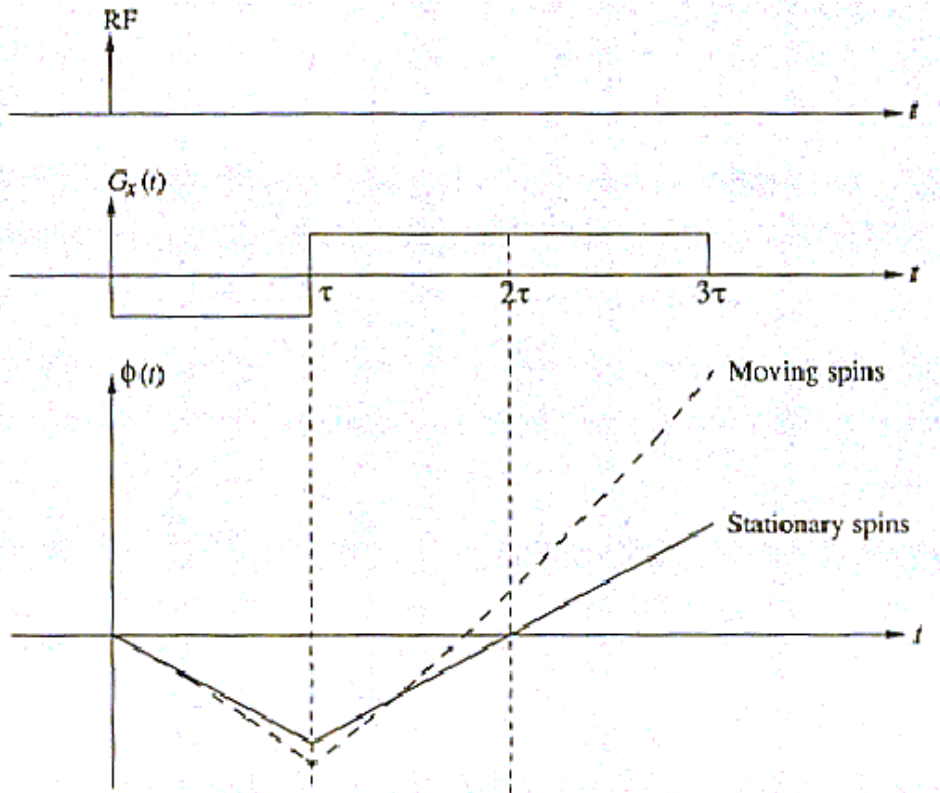


Figure 3.3: Gradient waveform and phase of spins as a function of time

The imaging effect along the frequency-encoding direction can then be described by the point spread function:

$$h(x) = \int_{\tau}^{3\tau} e^{-i\frac{1}{2}\gamma G_x v_x (t^2 - T_E)} e^{i\gamma G_x x (t - 2\tau)} dt \quad (3.6)$$

which can be approximated as

$$h(x) \approx \frac{(1-i)\sqrt{\pi}}{\sqrt{\gamma G_x v_x}} e^{-i\gamma G_x v_x \tau^2} e^{i\gamma G_x (x - x_s)^2 / v_x} \quad (3.7)$$

This equation indicates the following three imaging effects: phase shift, spatial shift, and blurring. The impact of the phase shift is negligible if the object is moving with a constant velocity; however, if its velocity varies within a voxel, the signal intensity will be reduced; if it varies from one acquisition to the next, image artifacts along the phase-encoding direction will be created. Spatial shifting is caused by points moving from the time the excitation pulse is applied to the time the echo peaks, meaning that the image reflects the points' positions at the echo time. Finally, blurring is caused by convolution with the point spread function. The effects of object rotation are calculated similarly. The total phase shift is given by:

$$\phi = \int_0^t [x(0) \cos \omega_0 \hat{t} + y(0) \sin \omega_0 \hat{t}] G_x(\hat{t}) d\hat{t} \quad (3.8)$$

The effects of motion along the phase-encoding direction are more of a problem than effects along the frequency-encoding direction since the sampling rate in the phase-encoding direction is at least two orders of magnitude slower than that in the frequency-encoding direction. However, because there is no phase accumulation from one phase-encoding step to the next, motion effects in this direction are easier to analyze.

The motion-introduced phase shift can be derived in one of two ways, either by using equation (3.1), or by beginning with the Fourier shift theorem, ignoring the frequency-encoding direction and assuming phase encoding is instantaneous. The imaging equation is then

$$S(k_n) = \int_{-\infty}^{\infty} I(x, t_n) e^{-i2\pi k_n x} dx \quad (3.9)$$

Remembering that the object function now contains a velocity component, the equation becomes

$$S(k_n) = e^{-i2\pi k_n v_x t_n} \int_{-\infty}^{\infty} I_0(x) e^{-i2\pi k_n x} dx \quad (3.10)$$

where the motion-introduced phase shift term is

$$\Delta\phi = 2\pi k_n v_x t_n \quad (3.11)$$

3.2 Motion Compensation Techniques

Some general methods for motion compensation include gradient moment nulling, navigator echoes, ghost phase cancellation (GPC), dynamic imaging by model estimation (DIME), and the use of navigator echoes to determine in real-time motion parameters prior to acquiring data. However, clinical applications have needed more specialized methods, such as gating techniques and the diminishing variance algorithm, to suppress motion effects. In practice, combinations of the above techniques have proven to be very effective.

The navigator echo method uses companion echo signals collected from the same k -space location in order to determine object motion parameters. With this information, motion artifacts can be eliminated during image reconstruction. Navigators are classified according to how they are mapped into k -space: linear and circular. Linear navigators are collected along a straight line, usually along the k_x or k_y axis. Two sets of navigators are therefore needed in order to determine object translations. Circular navigators, on the other hand, trace circular paths in k -space and can encode both translation and rotation in only one set of navigators.

Motion parameters from linear navigators can be determined in the following manner. First, let $S_n(k_x, 0)$ represent a set of navigators, $n = 0, 1, 2, \dots$, collected from the object $I(x, y, t)$. at times $t = t_n$. Translation of the object is represented by

$$I(x, y, t_n) = I(x + \Delta x_n, y + \Delta y_n, t_0) \quad (3.12)$$

The projection-slice theorem yields

$$\mathcal{F}^{-1}\{S_n(k_x, 0)\}(x) = \int_{-\infty}^{\infty} I(x + \Delta x_n, y, t_0) dy \quad (3.13)$$

Then,

$$\mathcal{F}^{-1}\{S_n(k_x, 0)\}(x) = \mathcal{F}^{-1}\{S_0(k_x, 0)\}(x + \Delta x_n) \quad (3.14)$$

from which the x-direction motion parameter can be calculated. For the y-direction parameter, navigator echoes are needed in the k_y axis. Similarly, it can be shown that

$$\mathcal{F}^{-1}\{S_n(0, k_y)\}(y) = \mathcal{F}^{-1}\{S_0(0, k_y)\}(y + \Delta y_n) \quad (3.15)$$

Motion parameters can be extracted from circular navigators in a similar approach. Consider two circular navigators $S_1(k_0, 0)$ and $S_2(k_0, 0)$ collected from $I_1(x, y)$ and

$$I_2(x, y) = I_1(x \cos \theta_0 + y \sin \theta_0 - x_0, -x \sin \theta_0 + y \cos \theta_0 - y_0) \quad (3.16)$$

It can be shown that

$$S_2(k_0, 0) = S_1(k_0, \theta - \theta_0) e^{-i2\pi k_0 [x_0 \cos(\theta - \theta_0) + y_0 \sin(\theta - \theta_0)]} \quad (3.17)$$

This demonstrates that magnitude shifts encode rotation, while phase differences encode translation. The rotation parameter θ_0 can be determined from the correlation function of S_1 and S_2 . Then, the translation parameters can be found from the following equations:

$$x_0 = -\frac{1}{2\pi^2 k_0} \int_0^{2\pi} \frac{d\Phi(\theta)}{d\theta} \sin(\theta - \theta_0) d\theta \quad (3.18)$$

$$y_0 = -\frac{1}{2\pi^2 k_0} \int_0^{2\pi} \frac{d\Phi(\theta)}{d\theta} \cos(\theta - \theta_0) d\theta \quad (3.19)$$

where

$$\frac{d\Phi(\theta)}{d\theta} = -ip^*(\theta) \frac{dp(\theta)}{d\theta} \quad (3.20)$$

and

$$p(\theta) = \frac{S_1(k, \theta) S_2^*(k, \theta - \theta_0)}{|S_1(k, \theta) S_2(k, \theta - \theta_0)|} \quad (3.21)$$

Gating is a technique in which the object is only scanned if it is within a certain window, as shown in Figure 3.4. Most of the early MRI image recovery methods involved the use of a gating window, which proved effective in reducing motion artifacts. However, since the object was only scanned when it was in a certain window, a lot of scan time was wasted. Furthermore, not all of the scanned data was actually useable in the image recovery process. The earliest algorithm relied on a motion test that determined if the data quality was sufficient. This was an acceptance/rejection algorithm that used data only if it passed the motion data. Much of the data ended up discarded, which required very long scan times in order to retrieve decent images. Many improvements were made, such as using weighing methods and motion histograms to set the gating window to where the object was most likely to be. Furthermore, the use of a navigator improved scan efficiency by increasing the gating window size. The variability of the subject's respiration was also a major drawback to all of these improvements since they depended upon predetermining where the most likely position would be.

The diminishing variance algorithm (DVA) is an improvement over earlier methods that involved gating. This algorithm adaptively selects the most appropriate acceptance window based on a preliminary set of data frames that it acquires from a pre-scan. As with some of the previous algorithms, DVA can be combined with phase ordering and weighting approaches so as to increase its efficiency. This algorithm reacquires data to improve the image quality.

The flow chart of the DVA is shown in Figure 3.5. A frame of data is first acquired using a navigator to track the motion and computes its motion histogram based on the frame of data. This process is repeated until an entire set of data has been completed. Each iteration through the algorithm is tracking the most likely position of the

diaphragm and acquires the next frame of data based on the histogram. The gating window is thus being adaptively selected.

When the data set has been completed, the most flawed data frame is then corrected. The statistical mode is computed based on the data frames. Next, the frame furthest from the mode is determined; this frame has the worst quality. In the algorithm, the user presets a threshold variance, which is the tolerance level of quality of the retrieved data frames. If the frame that is furthest from the mode is not within the threshold limit, that frame of data is reacquired. This step is repeated until either the time limit has been reached or all the frames are within the threshold.

The diminishing variance algorithm is a reacquisition strategy. Data frames are reacquired based on the variance among the positions on the histogram. Each reacquisition diminishes the variance among the positions on the histogram-hence its name. While the use of DVA has increased image quality in comparison to fixed gating techniques, it still suffers from several inefficiencies. Each time a data frame is reacquired, some previously acquired data is discarded. Another disadvantage to this algorithm is that a tradeoff between scan time and image quality is necessary. A high-resolution image is obtained at the cost of a longer scan time: a scan time that is 1.5 to approximately 4 times that of a normal scan produces an image that is higher in quality. While this algorithm is clearly superior to all the previous techniques, it is not resistant to significant changes in the breathing pattern that can occur during a scan due to its adaptive selection of the gating window.

The development of navigator echo techniques has enabled the implementation of several algorithms to reduce motion effects from an image during acquisition. Early gating techniques based on the acceptance/rejection algorithm were hindered by the loss in scan efficiency which resulted from the use of narrow gating windows required to effectively reduce artifacts and account for the variability of the subject's respiration. The use of phase-encode ordering and weighting can improve image quality. The combi-

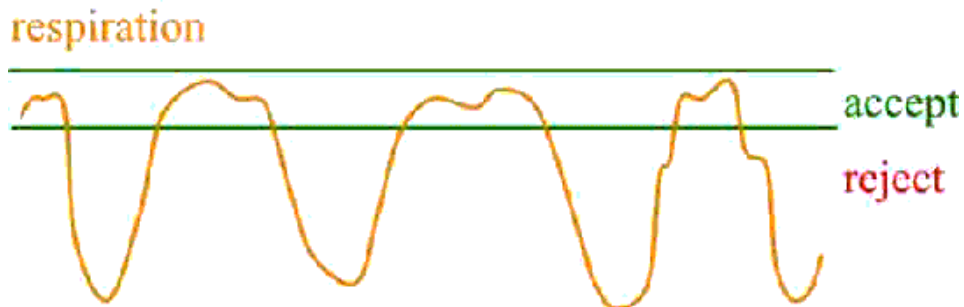


Figure 3.4: Illustration of generalized gating

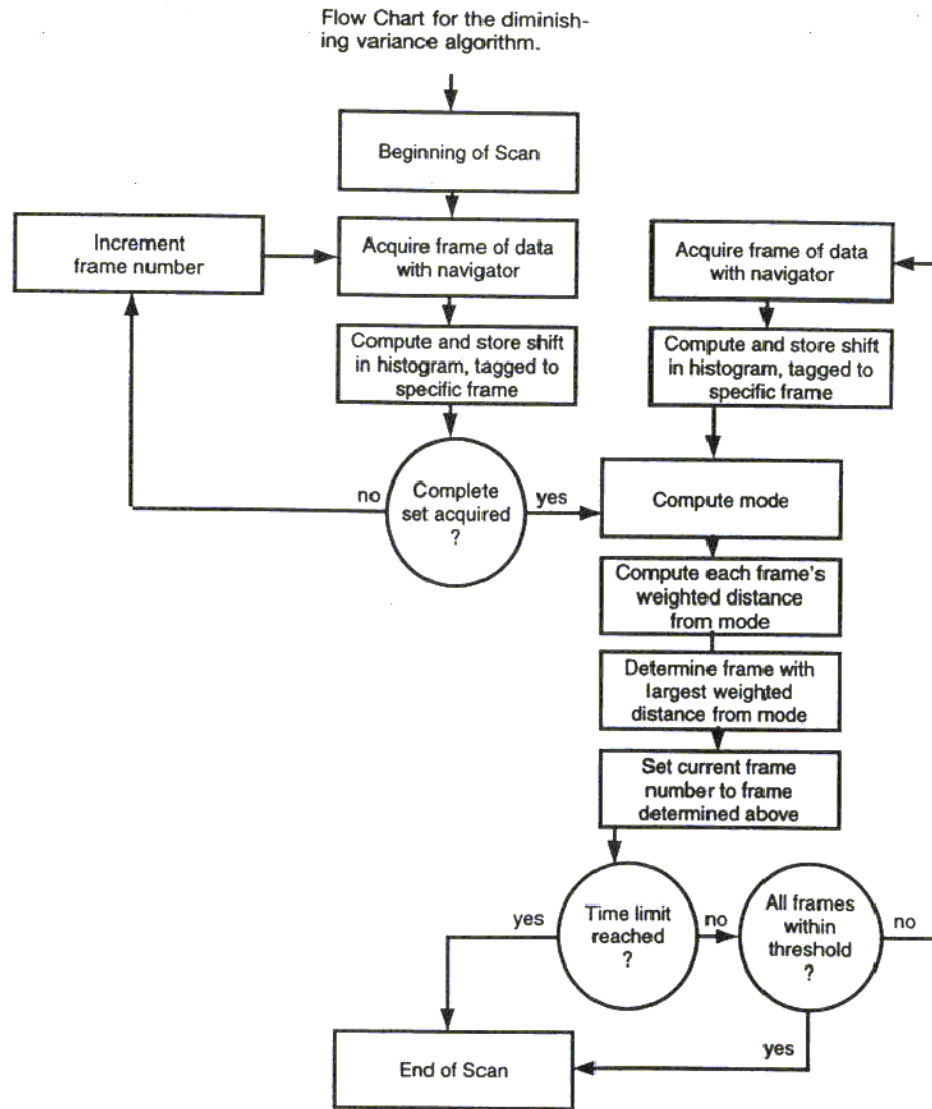


Figure 3.5: The diminishing variance algorithm

nation of these methods with a navigator acceptance window allows for a larger gating window to be used without reduction in image quality while improving scan efficiency.

The use of phase ordering and weighting techniques has been shown to significantly improve image quality over non-ordered window methods. However, the use of an acceptance window is inherent in all these techniques as a decision to accept or reject data. This limits the effectiveness of such techniques to the choice of the acceptance window. Furthermore, a change in the respiratory pattern during a scan may cause a window previously accepting data around the end-expiration position to be out of the new respiratory range, thereby causing inefficiencies.

A new technique known as phase ordering with automatic window selection (PAWS) uses a multilevel approach where no acceptance window is specified. This method uses the initial position of the diaphragm as the reference and all further diaphragm positions are given an index position, which is the displacement from this reference. Each index position, called a bin, is then allotted a starting position in a region of k-space. The bins are then split in to groups of three so that any three can be used to complete k-space. Image acquisition is complete once the entire k-space has been acquired by three consecutive bins.

Scan time using PAWS can be reduced further by two techniques. The bin size can be increased, allowing more than one index position to map each individual bin. Alternately, the navigator resolution can be decreased so that a wider range of positions map to the same index position. However, image quality may be compromised, as the phase ordering within each individual bin is less well defined.

There are a few possible ways to run the PAWS algorithm, as shown in Figure 3.6. Specifically, the bin allocation may be performed in different ways. For most cases, the bins are split into groups of three in order to enable optimum scanning time. If only two bins are used to complete k-space, the multilevel approach of PAWS can be used to provide the shortest possible scan time only when partial Fourier methods are employed. In general, two bins require an increase in scan time, as data are not acquired fast enough. Three bins provides a suitable compromise between scan time and image quality, while the use of a greater number of bins introduces problems of duplication of phase encoding lines in the same acceptance window and reduced scan efficiency. More than three bins also makes it impossible to predict the most suitable starting points for each diaphragm position, in particular the two center bins. This causes an inevitable overlap of phase encode-lines since the most efficient placement of starting position cannot be predetermined, resulting in an increase in scan inefficiency. Furthermore, three bins provide the ability to use a starting position in the center of k-space, which allows special conditions to be placed on acquisitions in this region while allowing an efficient method of acquiring and ordering data.

For many cases, the use of a three-point resolution acceptance window is too critical

and imposes severe time penalties on data acquisition. Therefore, increasing the size of the outer regions allows a reduction of scan time while limiting motion artifacts through the limited motion in the central region of the k-space. Time penalties of using this method have been shown to be minimal, and the use of three bins has proven to be effective in both reducing scan time and improving image quality.

One of the changes proposed by Wang was to modify the PAWS algorithm. In the published version, the bins are assigned to complete k-space in one of three ways: from the left edge towards the right, from the center towards the edges, and from the right edge towards the left. Assignment is done in that order as shown in Figure 3.7. In this method, a possible set of completed data can be for example right-left-center. For this example, the distance between the bins is larger. Displacement in k-space corresponds to a phase shift, so this case and the other case of center-right-left can suffer from phase distortion.

Dr. Wang believes that it would be both faster and have higher image quality if the allocation of the bins was modified. In the modified version, the center bin is allocated for every other bin. Left and right bins are then allocated every other set of four. This method has the disadvantage of having fewer possible combinations to complete the k-space scan, but it does have several advantages owing to how often the center of k-space is sampled. Center-assigned bins often determine how fast the k-space is completed. The center of the k-space also has the highest signal to noise ratio, which means that if the center of the k-space is not scanned correctly the image can become blurred.

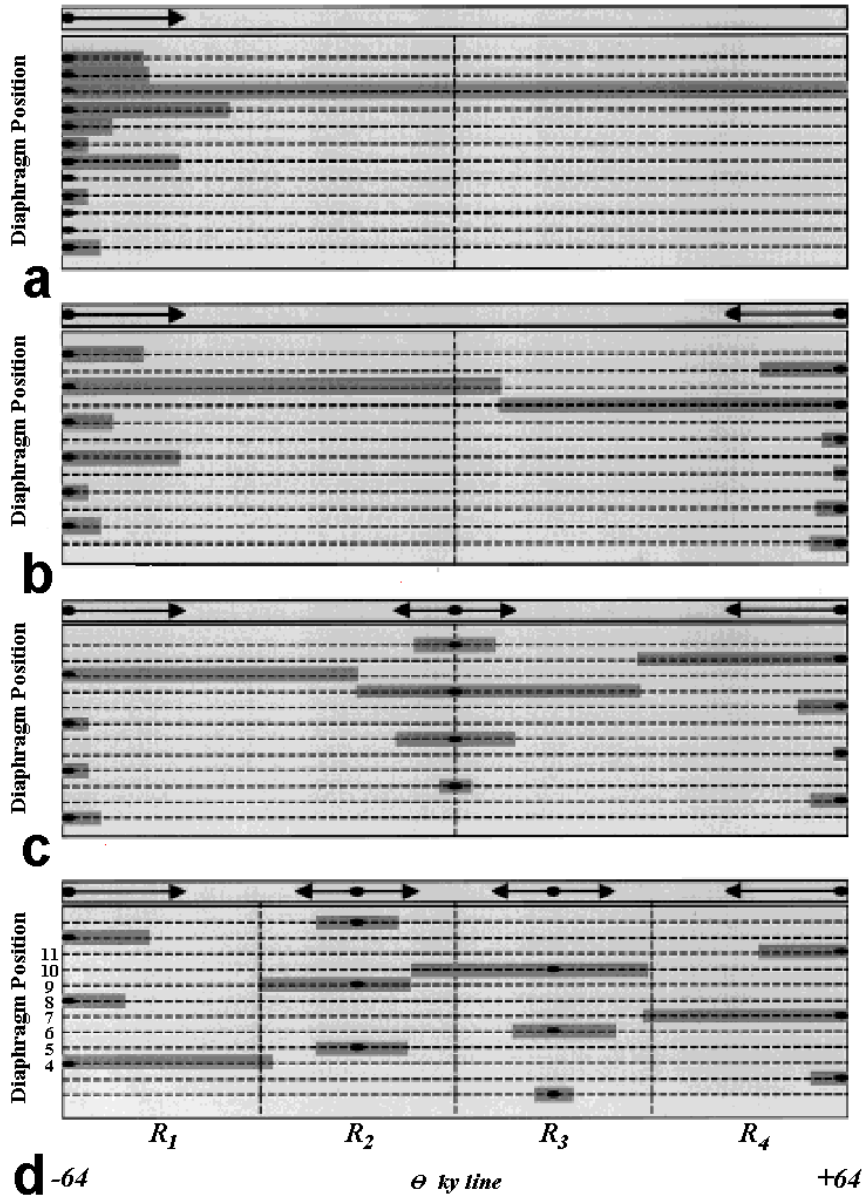


Figure 3.6: Possible permutations of the PAWS algorithm

Chapter 4

Our Work

4.1 Testing Hardware

The research facilities at the Weill Medical College of Cornell University have two General Electric Signa 1.5T magnetic resonance imaging scanners available for use. These scanners operate in a GE-proprietary environment, known as EPIC, which consists of a host computer (an SGI Indigo2) connected to the Transceiver Processing and Storage (TPS) chassis. The TPS unit contains the bulk of the hardware, including memory for storage of acquired data and the Integrated Pulse Generator (IPG), which is the heart of RF pulse generation.

The instructions for pulse generation are contained in programs known as PSDs (power sequence development), which reside on the host computer and, when executed, program the waveform memory with the pulse shapes desired. Logic Cell Array (LCA)-based sequencers read the waveform memory and produce 16-bit digital waveforms for the gradient and RF amplifiers located in the scan room. When a scan is performed, the appropriate PSD is specified at the host terminal for the type of scan to be carried out and the IPG is instructed accordingly. For specialized operations, custom PSDs can be written and compiled for use.

Received RF signals are picked up by receiver hardware in the scan room and are passed on to the Combined Exciter Receiver DAB (CERD) in the TPS chassis. The CERD board contains four Analog Receive Modules (ARMs) that demodulate and sample the signals. These modules operate at a sampling rate of 500 kHz, in accordance with the Nyquist criterion for a signal occupying 250 kHz of bandwidth and also allows for twice-oversampled ± 62.5 kHz digital data to be processed. The CERD board is controlled in real time by the PSD. The resulting digital output are then stored in TPS Bulk Access Memory (BAM).

Once data acquisition is complete, the raw MR data stored in the BAM are processed

by the ReflexAP section of the TPS and then placed in an image work area within the BAM. As these images are created, they are then transferred back to the host computer into the proper directory.

A real-time control interface, written in the C programming language by Vladimir Kolmogorov, resides on a Sun machine attached to the local area network. This program interfaces to the IPG and BAM and controls data acquisition according to the PAWS algorithms being tested.

4.2 Testing Software

The two General Electric 1.5T magnetic resonance scanners require the use of the EPIC program. As aforementioned in the previous section, the EPIC software is proprietary and is used to control the scanners from a remote workstation. The functions transfer what is known as a *common block*. The common block is a discrete unit of data that is sent between the MRI controller and the Sun Microsystems workstation. Vladimir Kolmogorov has developed the majority of the code necessary to manage the scan parameters from the PAWS/SMVA code. The only required changes involved conforming the source code to newer versions of the common block.

The primary concern of this thesis was to test the difference in image quality and efficiency between different versions of PAWS. The main test is between the Jhooti and Cornell versions of PAWS. The version developed by Jhooti requires that any three consecutive bins be allowed to complete k-space sampling. The Cornell version of PAWS relaxes that requirement in order to reduce the phase difference between consecutive bins. More details are in Chapter 4.3. Other tests include using two bins or four bins instead of the conventional three in order to test the theoretical evidence. In order to implement the changes, the code was changed to reflect the number of bins.

A motion artifact simulator was developed in order to assess the benefits of the different versions of PAWS. The code is contained in `simulator.c` (Appendix A). The motion artifact simulator works in three modes.

The first mode imposes a pure translation upon an image. The simulator accepts an image along with a phase parameter. The simulator program takes advantage of the fact that a translation in image domain corresponds to a phase shift in the spatial frequency domain. The simulator reads in the data and uses the FFTW (Fastest Fourier Transform in the West) library to transform the data into k-space (spatial frequency) if needed. At that point, it does a discrete phase shift and an inverse Fourier transform on the k-space data. The image is saved into a file and can be viewed via MATLAB or any program that allows the display of binary data.

The second mode simulates motion-artifacts on an image without a corrective algorithm. The program accepts an image along with a *navigator record*. The navigator

record contains the motion data from the diaphragm from the sequence of pencil-beam navigator echoes. The motion data are sampled sequentially. The ghosting effect is simulated by imposing a non-uniform phase shift over the image. As the image moves, the translation, and therefore the phase shift, changes. A different phase shift is applied to different regions of the image.

The final mode uses PAWS to correct motion artifacts. As mentioned before, PAWS specifies a new way to sample the motion data in order to reduce the artifacts. The PAWS code was modified to produce a *bin record*. The bin record specifies which window completed k-space and how it did so (which portions of k-space did each bin sample). The program uses this information to provide an accurate simulated image of an image that has undergone the PAWS algorithm.

The simulator accepts a variety of image file types; however, the files must be of similar format. The image file types that are accepted have headers followed by information about the image. The data after the header are the *pixels* that make up the image. Each pixel represents a discrete unit of area and its value corresponds to its color and intensity

The researchers at the Weill Medical College at Cornell University use the *.MR* file type to store many of their images. An MR file contains a header followed by pixel data. Each pixel is 2 bytes and represents by a real value only. In addition, this data are in image space. This format is similar to the common bitmap format provided in Microsoft Windows.

The other file type that is used in MR scanners is commonly known as a *P-File*. A P-file contains a header followed by pixel data. Each pixel is represented by a real and imaginary component, each of which uses 2 bytes. This complex data are in k-space, which means that no transform is required to use the phase shift on either

4.3 The PAWS Algorithm

Part of this project is comparing the published version of the PAWS algorithm [1] against the modified version of the PAWS algorithm written by Vladimir Kolmogorov [2]. In the published version, the bins are assigned to complete k-space in one of three ways: from the left edge towards the right, from the center towards the edges, or from the right edge towards the left. These bin assignments are repeated over all of the displacement bins as shown in Figure 4.1. In this method, a possible set of completed data can have three different configurations: right-left-center, bins 69-70-71 in Figure 4.1a, left-center-right, bins 70-71-72, and center-right-left, bins 71-72-73. For example, if k-space completed with bins 69-70-71, there would be a difference of 2 bin sizes between the center bin and the right bin. Displacement in real space corresponds to a phase shift in k-space. Therefore, cases where k-space is completed by non-adjacent bins, as in the cases using

center-right-left and right-left-center bin configurations, can suffer from phase distortion.

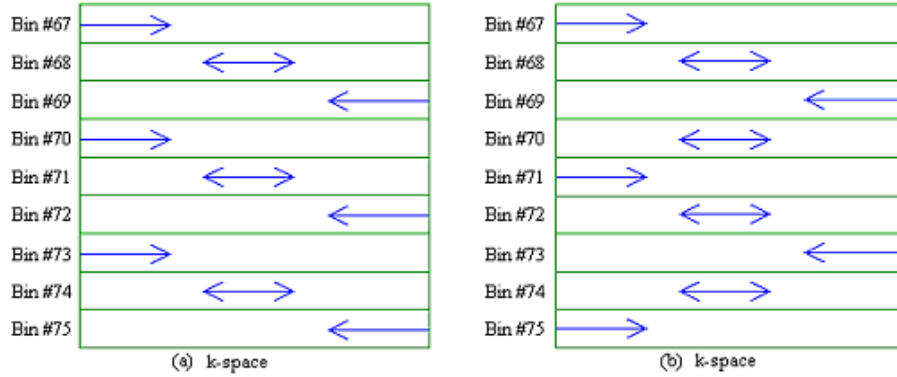


Figure 4.1: PAWS algorithm bin assignments: a) Jhooti's version. b) Cornell version

Theoretically, this published method finishes faster than the alternate version written by Kolmogorov since there are more possible ways to complete k-space. Each center bin has three possible configurations of completing k-space associated with it. As stated above, two of these configurations involve a large separation between adjacent bins; the left bin is adjacent to the right bin and vice versa. Translation in position over time corresponds directly with a phase shift in k-space. Therefore, a larger displacement causes more blurring.

Furthermore, there is no pre-sampling in the PAWS algorithm. Bins are pre-allocated and the bin number is set before the scan begins. For example, bins 1,4,7 all scan k-space in the same direction regardless of the appropriateness of this assignment to the current navigator record. There is also a high probability that the bin with the largest histogram value, or the most frequently sampled bin, does not correspond with a bin sampling from the center of k-space outward. In this case, k-space may be completed in such a way that the left or right bins intersect with the center bin near the center of k-space. For example, a highly-sampled left or right bin may meet the center bin near the center of k-space. Since the center of k-space has the highest signal to noise ratio, if sampling is completed with bins meeting in this region, the image will have more ghosting. For these reasons, the motion corrected image quality obtained using this method is likely to be inferior to the modified PAWS algorithm.

In Kolmogorov's version, every other bin is allocated as a center bin sampling outward, as shown in Figure 4.1b. The remaining two bins in each set of four are allocated as left and right sampling bins. Compared with Jhooti's version of PAWS, there are 1.5 times more center bins but there is only one configuration to complete k-space instead

of three. This means, on average, that this method requires more navigator samples to complete k-space. Kolmogorov’s version of PAWS does, however, possess several advantages over Jhooti’s version.

The modified version of PAWS is more resistant to frequent breathing changes than Jhooti’s version. As mentioned earlier, the frequency histogram is subject to constant change, leading to changes in the bins that would complete k-space. Since every other bin is a center bin, the likelihood of the peak in the histogram corresponding to a center bin is increased. As a result, the center is sampled more frequently than the right and the left bins and k-space is often completed with bins meeting away from the center of k-space, preserving the high SNR ratio of this region.

4.4 Results

The process of gathering results was difficult, as many problems were encountered when getting the PAWS code to work with the MRI scanner. In the meantime, the simulator code was used to generate artifacted images of an unmoving object imaged using the MRI scanner. Six unique navigator records of breathing patterns were passed to the simulator and the resulting images were qualitatively examined. Artifacting was quantitatively determined by using the original image data and the artifacted image data to obtain point spread functions. The full-width half-max value of the function was used to describe the degree of artifacting—larger numbers indicate more blurring of the image. In addition, the number of strong peaks in the function suggests the number of ghosted images seen in the artifacted images. The efficiency of the different PAWS algorithms given the specific navigator records were also calculated by using the ratio of data used to construct the final image to the total data collected.

A summary of the results is given in Table 4.1. The original image, artifacted images, and point spread function graphs may be found in Appendix B.

Navigator ID	PAWS (Jhooti)			PAWS (Cornell)		
	Efficiency	Center Bin	Q	Efficiency	Center Bin	Q
ithaca703	54%	82	5	51%	83	5
jql1288	28%	112	5	27%	111	7
mrp1424	19%	100	5	19%	99	5
vnk1060	29%	82	5	22%	83	3
yifeng1128	30%	121	7	29%	121	10
yw1322	19%	100	5	19%	99	5

Table 4.1: Results summary

Chapter 5

Future Improvements

5.1 Edge Detection

Key to the performance of the PAWS algorithm is the ability to track the position of the diaphragm. To this end, a type of navigator echo, known as a *pencil beam* navigator echo is used. Pencil beam navigator echoes excite a very narrow cylindrical region of space, effectively producing a one-dimensional profile.

In the experimental setup, a normal scan of the subject in the sagittal plane consisting of twenty slices is first conducted. This is used to determine the optimal location of the pencil beam—the region excited by the echo should ideally contain the possible range of motion of the diaphragm and include as few extraneous internal structures as possible.

Once the coordinates of the pencil beam have been programmed into the PSD’s user control variables, real-time control of the MRI scanner is handed off to the PAWS algorithm. During the scan, the program attempts to locate the position of the diaphragm by running an edge detection algorithm on the navigator echo data. Since the diaphragm effectively marks a boundary between the lungs, a region in the human body relatively free of hydrogen atoms, and the abdomen, a region rich in hydrogen, the navigator echo contains a high level of contrast, ideal for edge detection. The PAWS implementation written by Kolmogorov uses a windowing method that reduces the chance of smaller internal structures causing false edge readings; however, in practice this method does not adequately track diaphragm motion.

Better methods for extracting motion data from navigator echoes have been proposed by Ahn and Cho [7] and Nguyen and Wang [6]. Ahn and Cho proposed an unconventional average by taking the nearest neighbor correlation, which is computationally efficient but susceptible to noise. Nguyen and Wang suggest using a k -space weighted least-squares algorithm in which a straight line is fit to the motion-induced phase shift. This

straight line is weighted strongly to points in the center of k -space, corresponding to a high SNR, and only weakly weighted to the poor SNR points at the edges of k -space.

Let $S_{ref}[k]$ be the reference navigator echo and $S_{dis}[k]$ be a displaced echo. Then, the displacement d between the reference and displaced echoes corresponds to a linear phase shift in k -space:

$$s_{dis}[x] = s_{ref}[x + d] \stackrel{DFT}{\leftrightarrow} S_{dis}[k] = S_{ref}[k]e^{i2\pi/Nkd} \quad (5.1)$$

It follows that the displacement d can be determined thusly:

$$d = \frac{N}{2\pi} (\arg\{S_{dis}^*[k]S_{dis}[k+1]\} - \arg\{S_{ref}^*[k]S_{ref}[k+1]\}) \quad (5.2)$$

This indicates that only a pair of adjacent points is needed to derive the displacement information. Since all data points, in practice, are contaminated with noise, using only two data points would result in substantial error. To reduce this error, all other points in the echo can be averaged.

Ahn and Cho proposed taking the following average, taking the nearest neighbor correlation prior to the phase operator:

$$\frac{N}{2\pi} \arg E\{S^*[k]S[k+1]\} = \frac{N}{2\pi} \arg \sum_{k=-N/2}^{N/2-2} S^*[k]S[k+1] \quad (5.3)$$

The divider is dropped, since it has no effect on the phase. The displacement is then

$$\hat{d}_{AHN} = \frac{N}{2\pi} \left(\arg \sum_{k=-N/2}^{N/2-2} S_{dis}^*[k]S_{dis}[k+1] - \arg \sum_{k=-N/2}^{N/2-2} S_{ref}^*[k]S_{ref}[k+1] \right) \quad (5.4)$$

In this algorithm, k -space points contribute equally to the final estimate, making it susceptible to noise-dominated terms from the edges of k -space. However, the algorithm only requires $4N$ multiplications and $4N$ additions per estimate.

To reduce the contributions by noise-heavy terms, Nguyen and Wang use linear regression on k -space navigator phase data by fitting a straight line (in the weighted least-squares sense) to the phase change to obtain an optimized average:

$$\hat{d} : \min \left\{ \sum_{k=-N/2}^{N/2-1} \frac{|S[k]|^2}{2\sigma^2} \left(\Delta\varphi[k] - \frac{2\pi}{N}kd \right)^2 \right\} \quad (5.5)$$

where $\Delta\varphi[k] = \arg S_{dis}[k] - \arg S_{ref}[k]$. The weighting factor, $\frac{|S[k]|^2}{2\sigma^2}$, takes into account the inverse relationship between the noise in k -space phase data and SNR, which arises

from the Gaussian nature of the noise []. The problem posed in equation (5.5) has the following solution:

$$\hat{d}_{kLS} = \frac{N \sum_{k=-N/2}^{N/2-1} k |S[k]| \Delta\varphi[k]}{2\pi \sum_{k=-N/2}^{N/2-1} k^2 |S[k]|^2} \quad (5.6)$$

This approach requires only N multiplications and N additions per estimate if $k|S[k]|$ and $\sum_{k=-N/2}^{N/2-1} k^2 |S[k]|^2$ are precomputed.

Chapter 6

Conclusion

Based on our analysis, the Jhooti version of the PAWS algorithm is comparable to the Cornell version. The Cornell version can result in slightly better image quality for a similar scan time. One of the main reasons for this is that phase distortion occurs when the bins that are sampling k-space intersect. In addition, since signal-to-noise ratio is highest in the center of k-space, a phase-distortion can have a greater effect. The Cornell version allocates more bins to sample the center of k-space to prevent the phase-distortion from occurring around that region. The reason that the image quality is only slightly better is because the bins sampling the center of k-space must sample in both directions, which slows the center sampling significantly. These conclusions may be slightly premature (additional tests still need to be conducted), but they agree with theoretical evidence.

Appendix A

The Code

```
/**
 * This function introduces a phase shift onto a
 * given image file.
 *
 * Compile by:
 *
 * $ gcc simulator.c -lm -lfftw
 *
 * run by:
 * ./a.out file1 file2
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <math.h>
#include <fftw-2.1.3/fftw/fftw.h>
```

```
#define SIZE 256
```

```
// NEw additions (FS)
// Use PAWS or not
#define PAWS 1
```

```
// For the kspace data
#define KSPACEDATA_FILE "kspace.dat"
#define BINS 3
#define START 0
#define END 1
#define BINSIZE 3
#define LEFT 10
#define RIGHT 20
#define CENTER 30

#define BUFSIZE 1000
// Done

//optional scaling of navigator displacements
#define SCALE 10

// Pixel size in # of bytes
#define PIXSIZE 2

#define SIZEOFINT 4
#define SIZEOFCHAR 1
#define SIZEOFSHORT 2
#define SIZEOFLONG 4
#define SIZEOFLONGLONG 8

#define IMAGESIZE (SIZE*SIZE)
#define PI 3.1415926535897932384626
#define SAT256 1
#define SAT128 0

// Sets whether you want 2d or 1d
#define ONEDIM 0

#if ONEDIM
int row;
#endif
```

```

#define MAXNAV 10000
#define MAG2(x) (((x).im)*((x).im) + ((x).re)*((x).re))
#define MAG(x) (sqrt(MAG2(x)))
#define MAXIMUM_ACQUISITIONS 4096
#define FLAG_VALUE 12345
#define LOG_FILE "log.txt"
#define WINDOW_SIZE 30
#define SEARCH_SIZE 20
#define SIM_FILE "nav_raw"
#define SKIP 1
#define SIM_FILE_OFFSET 0
#define EFGRE3D_RT 1
#define EFGRE3D_CARD 2
#define EFGRE3D_CRT 3
#define EFGRE3D_CMS 4

typedef struct
{
int size; /* size of the navigator */
int sizeIm; /* size of the navigator (image space) */

short **nav; /* navigator history - nav[time][kx] */
fftw_complex **navIm; /* navigator history (image space) - navIm[time][x] */
float **navImMag; /* navigator history (image space, mag) - navImMag[time][x]*/
int *position; /* navigator position (in pixels) - position[time] */
int nav_num; /* number of navigators */

int ref_num; /* navigator reference */
int ref_disp; /* displacement of navigator window */

int nBins; /* number of bins */
int nav_accepted_num; /* number of accepted navigators */
int *iBin; /* history of iBin - only for accepted navigators */
float *hist; /* hist[iBin]/hist_norm is the frequency for bin iBin */
float hist_norm; /* normalizing coefficient for hist */

int navCoil; /* coil to be used for navigator */

```

```

fftw_plan plan; /* plan for computing fftw */
} NAV;

typedef struct
{
    void *address; /* Used as a checksum */
    int sequence; /* Pulse sequence type */
    int rhdxres; /* x-resolution, normally 160 (partial echo) or 256 */
    int rhnframes; /* y-resolution, normally <=256 */
    int opslquant; /* z-resolution, number of slices */
    int opfov; /* Field of view (mm) */
    int numrecv; /* Number of receive coils */
    int opfphases; /* Number of phases */
    int opentry; /* Patient Entry, 1=Head First, 2=Feet First */
    int view; /* View counter */
    int slice; /* Slice counter for multislice 2D acquisition */
    int pass; /* For multi-phase acquisition */
    int available; /* Navigator data ready to be uploaded */
    int vo_nhbs; /* Number of heartbeats for scan */
    int i_hb; /* Heartbeat to be acquired */
    float slaboffset; /* Displacement of acquisition slab in slice dir (mm) */
    float disp; /* Calculated displacement (mm) */
    int dabon; /* Acquire data this heartbeat */
                /* -1 = Nav Only, 0 = DABOFF, 1 = DABON */
    int status; /* 0 = PreDISDACQ, 1 = DISDACQ, 2 = ACQ */
    int finish; /* Exit scan */
    int old_dabon; /* Verify what was actually acquired */
    int old_i_hb; /* Verify what was actually acquired */
    float old_disp; /* Verify displacement used previously */
} COMMON_BLOCK;

int bam_address, ipg_address;
COMMON_BLOCK cBlock;

int pixelsPerBin = 3;
float *slaboffset;
int nVols;
int nBins;
int navCoil = 0;

```

```
int recon_size = 512;

int run_flag = 0;
int stop_flag = 0;
int exit_flag = 0;

int pixel_displacement[MAXNAV];
int pixel_displacement2[MAXNAV];

FILE *logfile;
FILE *SimFP;

int PIXEL_SHIFT;
int PFILE;
double PHASE_ANGLE;

double diff_mag(fftw_complex* in1, fftw_complex* in2);
void complex_mult(fftw_complex* in1, fftw_complex *in2, fftw_complex *out);
void xfm2d_shift(fftw_complex *fftw_image2d);
void xfm1d_shift(fftw_complex *fftw_image1d);
double magnitude(fftw_complex *c);
int ExtractDisplacement(void);
int round(double f);
int * argparse(int argc, char ** argv);

// Extracts info from PAWS
void PAWS_extractor();

int main(int argc, char** argv)
{
    FILE *fp_read, *fp_write, *fp_out, *fp_art;
    long last_part, totalsize, header, l;
    int i, c;
    fpos_t pos;
    fftw_complex fftw_image[IMAGESIZE];

    int * parsed;
```

```
// This is to determine pixel size
#if PIXSIZE == SIZEOFCHAR
    char image;
#elif PIXSIZE == SIZEOFSHORT
    short image;
#elif PIXSIZE == SIZEOFINT
    int image;
#elif PIXSIZE == SIZEOFLONG
    long image;
#else PIXSIZE == SIZEOFLONGLONG
    long long image;
#endif

// parse and check arguments
parsed = argparse(argc, argv);
PIXEL_SHIFT = parsed[0];
PFILE = parsed[1];
    printf("pixel shift = %d\n", PIXEL_SHIFT);
printf("pfile = %d\n", PFILE);

PHASE_ANGLE = -1*PIXEL_SHIFT*PI*2/SIZE;

    // Extract Navigator Data First
    ExtractDisplacement();

    // Open the file for reading
    fp_read=fopen(argv[1],"rb");
last_part = ftell(fp_read);

    // If there was an error opening the read file
    if (fp_read == NULL)
    {
        printf("ERROR: Cannot open %s\n", argv[1]);
        exit(1);
    }

    // Read to the end of the file
    totalsize = 0;
    while (getc(fp_read) != EOF)
```

```

    {
last_part=ftell(fp_read);
    totalsize++;
    }

    // At this point, fp_read is at the end of
    // the file and "totalsize" has the totalsize
    // of the file

    // Note the position of the file
last_part = ftell(fp_read);

    // Subtract out the IMAGESIZE part
last_part = (last_part >= IMAGESIZE) ? last_part - PIXSIZE*IMAGESIZE : 0;

// If the image file has imaginary data, I am going to
// have to move twice as much back
if (PFILE)
    last_part = last_part - PIXSIZE*IMAGESIZE;

if (last_part == 0)
{
    fprintf(stderr, "Warning: Original file < IMAGESIZE\n");
    fflush(stderr);
}

header = last_part;

/* puts the file pointer at the point pointed by "last_part"
   SEEK_SET means that its an offset from the beginning of the
   file */
fseek(fp_read, last_part, SEEK_SET);
fp_out = fopen("original.txt","wb");

    // Now read the part of the file that contains an image
if (PFILE)
{
    i = 0;

```

```

    while (!feof(fp_read))
    {
printf("i = %i\n",i);
fread(image, PIXSIZE*SIZEOFCHAR, 1, fp_read);
    fftw_image[i].re = image[0];
    fread(image, PIXSIZE*SIZEOFCHAR, 1, fp_read);
    fftw_image[i].im = image[0];

fprintf(fp_out, "%lf\n", fftw_image[i].re);
fprintf(fp_out, "%lf\n", fftw_image[i].im);

        i++;
    }
}
else // Image has only real components
{
    while (!feof(fp_read))
    {
fread(image, PIXSIZE*SIZEOFCHAR, 1, fp_read);
    fftw_image[i].re = image[0];
    fftw_image[i].im = 0;
    i++;
    }
}

#if PAWS
        PAWS_extractor();
    #endif

// Do fft->shift->ifft
#if ONEDIM
    for (i=0; i<SIZE; i++)
    {
        xfm1d_shift(fftw_image+i*SIZE);
    }
#else

```

```

    xfm2d_shift(fftw_image);
#endif

// Now write the data back
fp_write = fopen(argv[2],"wb");
fp_art = fopen("artifacted.txt","wb");

rewind(fp_read);

// put the rest of the data into the output file
for (i=0; i<IMAGESIZE; i++)
    {
double mag_im;
    if (PFILE)
    {
fwrite(&(fftw_image[i].re), sizeof(double),1,fp_write);
fwrite(&(fftw_image[i].im), sizeof(double),1,fp_write);
fprintf(fp_art, "%lf\n", fftw_image[i].re);
fprintf(fp_art, "%lf\n", fftw_image[i].im);
}
    else
    {
        mag_im = magnitude(fftw_image+i);
        fwrite(&mag_im, sizeof(double), 1, fp_write);
        mag_im = 0;
fwrite(&mag_im, sizeof(double), 1, fp_write);
}
    }

printf("Image done\n");
fflush(stdout);

// Close the files
fclose(fp_read); fclose(fp_write); fclose(fp_out); fclose(fp_art);

// Say Done!
printf("Done!!\n");
printf("totalsize = %d\n",totalsize);
printf("header = %d\n",header);

```

```
// Flush out all the buffers
    fflush(stdout);
fflush(stderr);

    return 0;
}

/**
 * Difference in magnitudes between two fftw_complex* variables
 */
double diff_mag(fftw_complex* in1, fftw_complex* in2)
{
    double mag1, mag2;

    mag1 = magnitude(in1); //(in1->re * in1->re) + (in1->im * in1->im);
    mag2 = magnitude(in2); //(in2->re * in2->re) + (in2->im * in2->im);

    return (mag1-mag2);
}

/**
 * Complex multiplication
 */
void complex_mult(fftw_complex* in1, fftw_complex *in2, fftw_complex *out)
{
    fftw_complex out_temp;

    out_temp.re = (in1->re * in2->re) - (in1->im * in2->im);
    out_temp.im = (in1->re * in2->im) + (in1->im * in2->re);

    out->re = out_temp.re;
    out->im = out_temp.im;

    return;
}
```

```

void xfm2d_shift(fftw_complex *fftw_image2d)
{
    // Two "plans" (configuration for xfms)
    // one for dft, the other for idft
    fftwnd_plan fft, ifft;

    int xfm_image2d[IMAGESIZE];

        fftw_complex  fftw_xfm_image2d[IMAGESIZE],
                    ex[IMAGESIZE];
    int i;

    fprintf(stderr, "Entering xfm2d_shift\n");
    fflush(stderr);

    // Initialize fftw_image2d
    for (i=0; i<IMAGESIZE; i++)
    {
        // Clear out the rest of the image-related data
        fftw_xfm_image2d[i].re = fftw_xfm_image2d[i].im
            = xfm_image2d[i]
            = 0;
    }

    // Create the fft/iff plans
    // fftwnd_plan fftw2d_create_plan(int nx, int ny,
    //  fftw_direction dir, int flags);
    fft = fftw2d_create_plan(SIZE, SIZE, FFTW_FORWARD , FFTW_ESTIMATE);
    ifft = fftw2d_create_plan(SIZE, SIZE, FFTW_BACKWARD, FFTW_ESTIMATE);

    if (!PFILE)
        fftwnd_one(fft, fftw_image2d, fftw_xfm_image2d);

    // Normalize
    for (i=0; i<IMAGESIZE; i++)
    {
        int displacement;
        if (PIXEL_SHIFT==1)
            {

```

```

displacement = SCALE*pixel_displacement2[i%SIZE];
//printf("disp = %d\n", displacement);
}
else
{
displacement = 1;
// printf("disp = %d\n", displacement);
}

#if PAWS
displacement = SCALE*pixel_displacement2[i%SIZE];
#endif

//displacement=0;
// int j=(int)((i/SIZE)-SIZE/2);
//printf("phase angle = %lf\n", PHASE_ANGLE);
    if (!PFILE)
    {
fftw_xfm_image2d[i].re = fftw_xfm_image2d[i].re / IMAGESIZE;
    fftw_xfm_image2d[i].im = fftw_xfm_image2d[i].im / IMAGESIZE;
    }

// Implement a phase shift
    ex[i].re = cos(displacement*BINSIZE*(PHASE_ANGLE)*(i%SIZE - 1*SIZE/2));
    ex[i].im = sin(displacement*BINSIZE*(PHASE_ANGLE)*(i%SIZE - 1*SIZE/2));

if (PFILE)
    complex_mult(ex+i, fftw_image2d+i, fftw_image2d+i);
else
    complex_mult(ex+i, fftw_xfm_image2d+i, fftw_xfm_image2d+i);
if(((i % SIZE) % 2) == 1) {
fftw_xfm_image2d[i].re = 0;
fftw_xfm_image2d[i].im = 0;
}
}

if (!PFILE)
fftwnd_one(iff, fftw_xfm_image2d, fftw_image2d);

```

```

// Reclaim resources claimed by plans
fftwnd_destroy_plan(fft);
fftwnd_destroy_plan(ifft);

fprintf(stderr, "Exiting xfm2d_shift\n");
fflush(stderr);

return;
}

/* 1D shift */
void xfm1d_shift(fftw_complex *fftw_image1d)
{
    int i;

    // Two "plans" (configuration for xfms)
    // one for dft, the other for idft
    fftw_plan fft, ifft;

    int xfm_image1d[SIZE];

        fftw_complex  fftw_xfm_image1d[SIZE],
                    ex[SIZE];

    fprintf(stderr, "Entering xfm1d_shift\n");
    fflush(stderr);

    // Initialize fftw_image1d
    for (i=0; i<SIZE; i++)
    {

        // Clear out the rest of the image-related data
        fftw_xfm_image1d[i].re  = fftw_xfm_image1d[i].im
                               = xfm_image1d[i]
                               = 0;
    }
}

```

```

// Create the fft/iff plans
// fftwnd_plan fftw1d_create_plan(int nx, int ny,
//                               fftw_direction dir, int flags);
fft = fftw_create_plan(SIZE, FFTW_FORWARD , FFTW_ESTIMATE);
ifft = fftw_create_plan(SIZE, FFTW_BACKWARD, FFTW_ESTIMATE);

if (!PFILE)
    fftw_one(fft, fftw_image1d, fftw_xfm_image1d);

// Normalize
for (i=0; i<SIZE; i++)
{
int displacement;
    if (PIXEL_SHIFT==1)
displacement = pixel_displacement2[i%SIZE];
else
displacement = 1;

#if PAWS
displacement = pixel_displacement2[i%SIZE];
#endif

if (!PFILE)
{
    fftw_xfm_image1d[i].re = fftw_xfm_image1d[i].re / IMAGESIZE;
    fftw_xfm_image1d[i].im = fftw_xfm_image1d[i].im / IMAGESIZE;
}

// Implement a phase shift
ex[i].re = cos(displacement*PHASE_ANGLE*i);
ex[i].im = sin(displacement*PHASE_ANGLE*i);
if (PFILE)
    complex_mult(ex+i, fftw_image1d+i, fftw_image1d+i);
else
    complex_mult(ex+i, fftw_xfm_image1d+i, fftw_xfm_image1d+i);
}

if (!PFILE)

```

```
    fftw_one(iff, fftw_xfm_image1d, fftw_image1d);

// Reclaim resources claimed by plans
    fftw_destroy_plan(fft);
    fftw_destroy_plan(iff);

    fprintf(stderr, "Exiting xfm1d_shift\n");
    fflush(stderr);

    return;
}

double magnitude(fftw_complex *c)
{
    double magt;

    magt  = (c->re)*(c->re);
    magt += (c->im)*(c->im);

    magt = sqrt(magt);

    //return round(magt);
return magt;
}

int round(double f)
{
    int x = 0;

    #if SAT256
        if (f < -256)
            f = -256;
        if (f >=255)
            f = 255;
    #endif
}
```

```

#if SAT128
    if (f < -128)
        f = -128;
    if (f >= 127)
        f = 127;
#endif

// return ceil(f);

// Round
if ( (f-floor(f)) < 0.5)
    return (floor(f));
else
    return (floor(f) + 1);
}

/*****DISPLACEMENT DATA*****/

int GetBAM(short *data, int bam_address, int length_in_bytes)
{
    int tn_readID = 0;
    int i;

    for (i=0; i<SKIP; i++)
    {
        if (fread((void *) data, 1, length_in_bytes, SimFP)<length_in_bytes)
        {
            fprintf(stderr, "%s: Failed to read %d bytes at position %d\n",
                    SIM_FILE, length_in_bytes, ftell(SimFP));
            return 0;
        }
    }

    return 1;
}

```

```
void SetBAM(short *data, int bam_address, int length_in_bytes)
{
    int tn_writeID = 0;
}

int InitializeCommunication(int *ipg_address, int *bam_address)
{
    int ipg_length;
    FILE *fp;
    char server_name[]="MRS2-tps";

    if( (SimFP=fopen(SIM_FILE, "rb")) == NULL)
    {
        printf("Error in open %s\n", SIM_FILE);
        exit(1);
    }
    fseek(SimFP, SIM_FILE_OFFSET, SEEK_SET);

    return 1;
}

void GetIPG(COMMON_BLOCK *cblock, int ipg_address)
{
    cblock->address = 0;
    cblock->sequence = EFGRE3D_CMS;
    cblock->rhdxres = 160;
    cblock->rhframes = 130;
    cblock->opslquant = 32;
    cblock->numrecv = 4;
    cblock->opfphases = 1;
    cblock->view = 0;
    cblock->slice = 0;
    cblock->pass = 0;
    cblock->available = 0;
    cblock->vo_nhbs = 130;
```

```

        cblock->i_hb = 0;
        cblock->status = 2;
        cblock->finish = 0;

    }
void SetIPG(COMMON_BLOCK *cblock, int ipg_address)
{
    int tn_writeID=0;

        cblock->old_i_hb = cblock->i_hb;
        cblock->old_dabon = cblock->dabon;

    }

NAV* InitNav(int size, int sizeIm, int nBins, int navCoil)
{
    int i;
    NAV *nav = malloc(sizeof(NAV));

    nav->size = size;
    nav->sizeIm = sizeIm;
    nav->nBins = nBins;
    nav->navCoil = navCoil;
    nav->ref_num = -1;

    nav->nav      = (short **) malloc(MAXNAV*sizeof(short*));
    nav->navIm    = (fftw_complex **) malloc(MAXNAV*sizeof(fftw_complex*));
    nav->navImMag = (float **) malloc(MAXNAV*sizeof(float*));
    nav->position = (int *) malloc(MAXNAV*sizeof(int));
    nav->iBin     = (int *) malloc(MAXNAV*sizeof(int));
    for (i=0; i<MAXNAV; i++)
    {
        nav->nav[i] = NULL;
        nav->navIm[i] = NULL;
        nav->navImMag[i] = NULL;
        nav->position[i] = -1;
        nav->iBin[i] = -1;
    }
}

```

```

nav->nav_num = 0;
nav->nav_accepted_num = 0;

nav->hist = (float *) malloc(nav->nBins*sizeof(float));
for (i=0; i<nav->nBins; i++) nav->hist[i] = (float)0;
nav->hist_norm = (float)0;

nav->plan = fftw_create_plan(nav->sizeIm, FFTW_BACKWARD, FFTW_ESTIMATE);

return nav;
}

```

```

int GetNavigator(NAV *nav, COMMON_BLOCK *cBlock)
{
int i;
int offset;
short *navdata;

offset = nav->navCoil*cBlock->opslquant*(cBlock->rhframes+1)*cBlock->rhaxres*2*sizeof(short);

navdata = (short *) malloc(cBlock->rhaxres*2*sizeof(short));
for (i=0; i<cBlock->rhaxres*2; i++) navdata[i] = FLAG_VALUE;
SetBAM(navdata, bam_address, cBlock->rhaxres*2*sizeof(short));

if (!GetBAM(navdata, bam_address+offset, cBlock->rhaxres*2*sizeof(short)))
{
free(navdata); return 0;
}

nav->nav[nav->nav_num++] = navdata;
return 1;
}

```

```

void ComputeNavIm(NAV *nav, int setref)
{
int i, j, k, imax = -1;
float v, vmax;

```

```

short *navData;
fftw_complex *src, *navIm;
float *tmp, *navImMag;
int size;

size = nav->sizeIm;
navData = nav->nav[nav->nav_num-1];
if (setref) nav->ref_num = nav->nav_num-1;

src = (fftw_complex *) malloc(size*sizeof(fftw_complex));
nav->navIm[nav->nav_num-1] = navIm = (fftw_complex *) malloc(size*sizeof(fftw_complex));
nav->navImMag[nav->nav_num-1] = navImMag = (float *) malloc(size*sizeof(float));

for (i=0; i<nav->size && i<size; i++)
{
if (i&1)
{
src[i].re = navData[2*i];
src[i].im = navData[2*i+1];
}
else
{
src[i].re = -navData[2*i];
src[i].im = -navData[2*i+1];
}
}
for (; i<size; i++) src[i].re = src[i].im = 0;
fftw_one(nav->plan, src, navIm);

for (i=0; i<size; i++) navImMag[i] = MAG(navIm[i]);

tmp = (float *) malloc(size*sizeof(float));
tmp[0] = 0;
for (i=0; i<WINDOW_SIZE; i++) tmp[0] += navImMag[i];
for (i=0; i+WINDOW_SIZE<size; i++)
{
tmp[i+1] = tmp[i] + navImMag[i+WINDOW_SIZE] - navImMag[i];
}
for (i++; i<size; i++) tmp[i] = 0;

```

```

if (setref)
{
vmax = 0;
imax = size/2;
for (i=100; i<size-100; i++)
{
v = tmp[i] - tmp[i-WINDOW_SIZE];
if (vmax < v) { vmax = v; imax = i; }
}
nav->ref_disp = imax;
}

if (nav->ref_num >= 0)
{
float *refImMag = nav->navImMag[nav->ref_num];
float vmin;

/* first estimate */
vmax = 0;
imax = size/2;
for (i=100; i<size-100; i++)
{
v = tmp[i] - tmp[i-WINDOW_SIZE];
if (vmax < v) { vmax = v; imax = i; }
}

/* correlation */
j = imax;
imax = -1;
for (i=j-SEARCH_SIZE; i<=j+SEARCH_SIZE; i++)
{
v = 0;
for (k=-WINDOW_SIZE; k<WINDOW_SIZE; k++)
{
float d;
d = refImMag[nav->ref_disp + k] - navImMag[i + k];
v += d*d;
}
}

```

```

if (imax<0 || vmin > v) { vmin = v; imax = i; }
}

```

```

}

```

```

nav->position[nav->nav_num-1] = imax;

```

```

free(tmp);
free(src);
}

```

```

void CloseNav(NAV *nav)

```

```

{

```

```

int i;

```

```

char name[80];

```

```

FILE *fp;

```

```

printf("Saving navigator (image space)... "); fflush(stdout);

```

```

sprintf(name, "nav_im%d", nav->nav_num);

```

```

if( (fp=fopen(name, "wb")) == NULL) { printf("Error in open %s\n", name); exit(1);

```

```

for (i=0; i<nav->nav_num; i++)

```

```

{

```

```

fwrite(nav->navIm[i], nav->sizeIm*sizeof(fftw_complex), 1, fp);

```

```

}

```

```

fclose(fp);

```

```

printf("done\n"); fflush(stdout);

```

```

for (i=0; i<nav->nav_num; i++) { free(nav->nav[i]); free(nav->navIm[i]); free(nav->

```

```

free(nav->nav);

```

```

free(nav->navIm);

```

```

free(nav->navImMag);

```

```

free(nav->position);

```

```

free(nav->iBin);

```

```

free(nav->hist);

```

```

free(nav);

```

```

}

```

```

int dw_nav_level;

```

```

int dw_nav_window;
int dw_nav_xpos;
int dw_nav_lastnav;
int displacementindex;

void DisplayNavLine(int size, float *data, int displacement)
{
int i;

for (i=0; i<size/2; i++)
{
float v;
int c;

v = ((data[2*i]+data[2*i+1])/2 - dw_nav_level)/dw_nav_window + (float)0.5;
// c = v*GRAY_NUM;
// if (c<0) c = 0; if (c>GRAY_NUM-1) c = GRAY_NUM-1;
// xsetcolor(gmap[c]);
// xdraw_line(dw_nav, 3*dw_nav_xpos, i, 3*dw_nav_xpos+2, i);
}

i = displacement/2;
// xsetcolor(RED);
// xdraw_line(dw_nav, 3*dw_nav_xpos+1, i, 3*dw_nav_xpos+1, i);
pixel_displacement[displacementindex++] = i;

dw_nav_xpos ++;
}

void InitDisplayNav(NAV *nav)
{
int i;
// xsetcolor(BLACK);
// xfill_rectangle(dw_nav, 0, 0, dw_nav->width, dw_nav->height);
dw_nav_level = 0;
dw_nav_window = 1;
dw_nav_xpos = 0;
dw_nav_lastnav = 0;
}

```

```

void DisplayNav(NAV *nav)
{
if (dw_nav_window==1 && nav->ref_num>0)
{ /* initialization */
int i;
float v = 0;
for (i=0; i<nav->sizeIm; i++)
{
v += nav->navImMag[nav->ref_num][i];
}
v /= nav->sizeIm;

dw_nav_level = (int)v;
dw_nav_window = (int)(2*v);

if (dw_nav_window==0) dw_nav_window = 1;
}

for (; dw_nav_lastnav<nav->nav_num; dw_nav_lastnav++)
{
// if (3*dw_nav_xpos>dw_nav->width)
{
// xsetcolor(BLACK);
// xfill_rectangle(dw_nav, 0, 0, dw_nav->width, dw_nav->height);
dw_nav_xpos = 0;
}
DisplayNavLine(nav->sizeIm, nav->navImMag[dw_nav_lastnav], nav->position[dw_nav_la
}
}

int ExtractDisplacement()
{
int i, iBin;
NAV *nav;
int displacement;
// struct timespec Start, End;

```

```

displacementindex=0;
/***** open connection *****/
while (!InitializeCommunication(&ipg_address, &bam_address))
{
sleep(1);
if (stop_flag) return;
}
GetIPG(&cBlock, ipg_address);
cBlock.finish = 0;
SetIPG(&cBlock, ipg_address);

        /***** open connection end *****/

nav = InitNav(cBlock.rhdaxres, recon_size, nBins, navCoil);
InitDisplayNav(nav);

GetNavigator(nav, &cBlock);
ComputeNavIm(nav, 0);
GetNavigator(nav, &cBlock);
ComputeNavIm(nav, 1);

/***** Main Acquisition Loop *****/
for (i=0; i<MAXIMUM_ACQUISITIONS && !stop_flag; i++)
{
if (!GetNavigator(nav, &cBlock)) break;

ComputeNavIm(nav, 0);
if (pixelsPerBin > 10) displacement = recon_size/2;
else displacement = nav->position[nav->nav_num-1];
iBin = displacement/pixelsPerBin;

if (iBin>=2 && iBin<=nBins-3)
{
nav->iBin[nav->nav_accepted_num++] = iBin;
}

DisplayNav(nav);

}

```

```

/***** Main Acquisition Loop End *****/

cBlock.finish = 1;
SetIPG(&cBlock, ipg_address);          /* Tell PSD that we're done */

CloseNav(nav);

    for(i = 0; i < displacementindex-1; i++) {

// pixel_displacement2[i] = pixel_displacement[i/2+10] - pixel_displacement[2];
pixel_displacement2[i] = pixel_displacement[i+10] - pixel_displacement[2];
}

}

int * argparse(int argc, char ** argv)
{
static int parsed[2];
int PIXEL_SHIFT;
int PFILE;

// Check to see if enuf command line arguments were given
    if (argc < 4)
    {
        // Print error message
        //fprintf(stderr, "ERROR: Not enough arguments to prgm\n");
        //fprintf(stderr, "Usage:\n ./a.out in_file out_file \n");
fprintf(stderr, "Usage: %s in_file out_file [mr/p] [pixel displacement (optional)]\n");
        // Force printing
        fflush(stderr);

        // exit the prgm (since we can't do anything)
        exit(1);
    }

if (!strcmp(argv[3], "mr"))
PFILE = 0;
else if (!strcmp(argv[3], "p"))
PFILE = 1;
else

```

```
{
fprintf(stderr, "Must use .MR or P-files for now.\n");
exit(1);
}

if (argc==4)
PIXEL_SHIFT = 1;
else
PIXEL_SHIFT = atoi(argv[4]);

parsed[0] = PIXEL_SHIFT;
parsed[1] = PFILE;

return parsed;
}

void PAWS_extractor()
{
    FILE *kspacefile;
    int c[BINS], limits[BINS][2], i, j, how_many_lines;
    char buffer[BUFSIZE];

    // Open the file
    kspacefile = fopen(KSPACEDATA_FILE,"rb");

    if (kspacefile == NULL)
    {
        fprintf(stderr, "Cannot open %s", KSPACEDATA_FILE);
        fflush(stderr);

        exit(1);
    }

    // Clear out c
    for (i=0; i<BINS; i++)
    {
        c[i] = 0;
    }
}
```

```
        limits[i][END] = 0;
        limits[i][START] = 0;
    }

    //Reset i & j
    i =0;

    // How many lines does the file have?
    how_many_lines = 0;

    // Read until the end of the file...we need the last BINS numbers
    while (!feof(kspacefile))
    {
        // Store the character
        fscanf(kspacefile, "%d", &j);

        if (!feof(kspacefile))
        {
            c[i] = j;
        }

        // Increment i
        i++;
        i=i%3;
    }

    // At this point, we've got the 3 last integers in the file
    // Lets figure out which one of the bins sample which parts of
// kspace

    // Rewind the file
    rewind(kspacefile);

    // Lets figure out how many lines this file has
    while (!feof(kspacefile))
    {
        // Get a line from the file
        if (fgets(buffer, BUFSIZE, kspacefile) == NULL)
            break;
    }
}
```

```

        how_many_lines++;
    }

    // Rewind the file
    rewind(kspacefile);

    fprintf(stderr, "how many lines = %d\n", how_many_lines);

    fflush(stderr);

    for (j=0; j<how_many_lines-1; j++)
    {
        int bin, ky, match=0;
        // Get a line from the file
        fgets(buffer, BUFSIZE, kspacefile);

        // Read two integers from the buffer
        sscanf(buffer, "%d %d", &bin, &ky);

        if (!feof(kspacefile))
        {
            // Check to see if bin matches any one
            // of the three "chosen" bins
            for (i=0; i<BINS; i++)
            {
                if (c[i] == bin)
                {
                    if (limits[i][START] == 0)
                        limits[i][START] = ky;

                    // limits[i][END]
                    //   =MAX(ky,limits[i][END])
                    limits[i][END]
                        = (limits[i][END] <=ky)
                          ? ky : limits[i][END];

                    // limits[i][START]
                    //   = MIN(ky,limits[i][START])

```

```

limits[i][START]
    = (limits[i][START] >= ky)
    ? ky : limits[i][START];
    }
    }
}

// Now we know the limits of each bin,
// Lets try to figure out which one is which is the center bin
for(i=0; i<SIZE; i++) {
pixel_displacement2[i] = 0;
}

if (BINS == 2)
{
// BINS == 2 is a special case cause the
// edges have TWO starts and ends

// Lets see which one is the center bin
int centerbin;
int edgebin;
int globaldisp;
if (limits[0][START] > limits[1][START])
{
centerbin = 0;
edgebin = 1;
}
else {
centerbin = 1;
edgebin = 0;
}

globaldisp = c[centerbin];
fprintf(stderr, "global shift = %d\n", globaldisp);
for (j=limits[edgebin][START]; j<limits[centerbin][START];j++)
{
pixel_displacement2[2*j] = c[i] - globaldisp;
pixel_displacement2[2*j+1] = c[i] - globaldisp;
}
}

```

```

}
  for (j=limits[centerbin][START]; j<limits[centerbin][END];j++)
    {
      pixel_displacement2[2*j] = c[i] - globaldisp;
      pixel_displacement2[2*j+1] = c[i] - globaldisp;
    }
  for (j=limits[centerbin][END]; j<limits[edgebin][END];j++)
    {
      pixel_displacement2[2*j] = c[i] - globaldisp;
      pixel_displacement2[2*j+1] = c[i] - globaldisp;
    }
}
else
{
int center = -1;
int globaldisp;
int mind[BINS];
int max=0;
for (i=0; i<BINS; i++)
{
mind[i]=0;
}
    for (i=0; i<BINS; i++)
    {
fprintf(stderr, "cbin[%d] = %d\n", i, c[i]);
fprintf(stderr, "limits[%d][START]= %d\n", i,limits[i][START]);
fprintf(stderr, "limits[%d][END]= %d\n", i,limits[i][END]);
if ((limits[i][START]<=64)&&(64<=limits[i][END]))
{ // the bin collects the center ky line
fprintf(stderr,"c[i]=%d\n",c[i]);
mind[i]=64-limits[i][START];
if((limits[i][END]-64)<mind[i]) mind[i]=limits[i][END]-64;
fprintf(stderr,"mind[i]=%d\n",mind[i]);
}
}
for (i=0; i<BINS; i++)
{
if(mind[i]>=max)
{

```

```

max=mind[i];
center = i;
}
}
globaldisp = c[center];
//fprintf(stderr,"global shift = %d\n",c[i]);
fprintf(stderr,"global shift = %d\n", globaldisp);

for (i=0; i<BINS; i++)
    {
        for (j=limits[i][START]; j<=limits[i][END]; j++)
            {
                pixel_displacement2[2*j] = c[i] - globaldisp;
pixel_displacement2[2*j+1] = c[i] - globaldisp;
            }
    }

for (j=limits[center][START]; j<=limits[center][END]; j++)
    {
        pixel_displacement2[2*j] =0;// c[i] - globaldisp;
pixel_displacement2[2*j+1] = 0;//c[i] - globaldisp;
fprintf(stderr, "pixel disp[%d] = %d\n", j,pixel_displacement2[2*j]);
    }

    // Temporary fix for 130 problem
/*
    for (i=129; i<SIZE; i++)
        pixel_displacement2[i]
            = pixel_displacement2[128];
*/

    fclose(kspacefile);
}
return;
}

```

Appendix B

Data

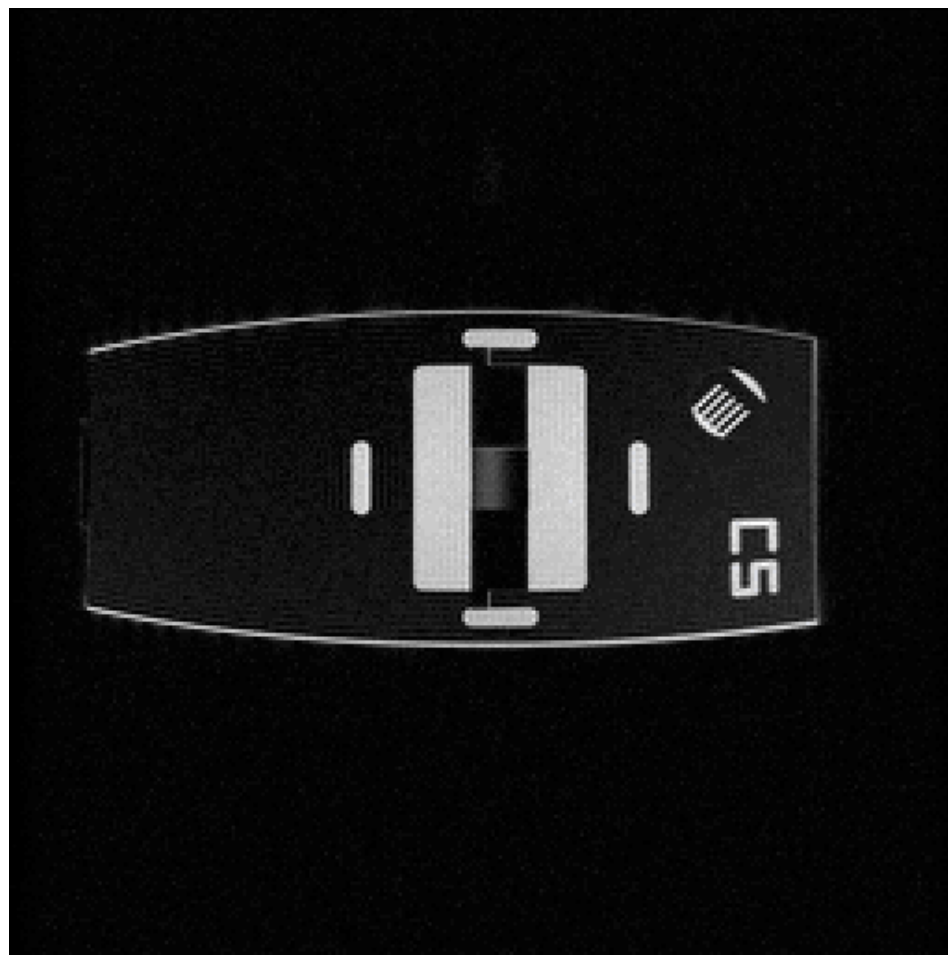


Figure B.1: Original image

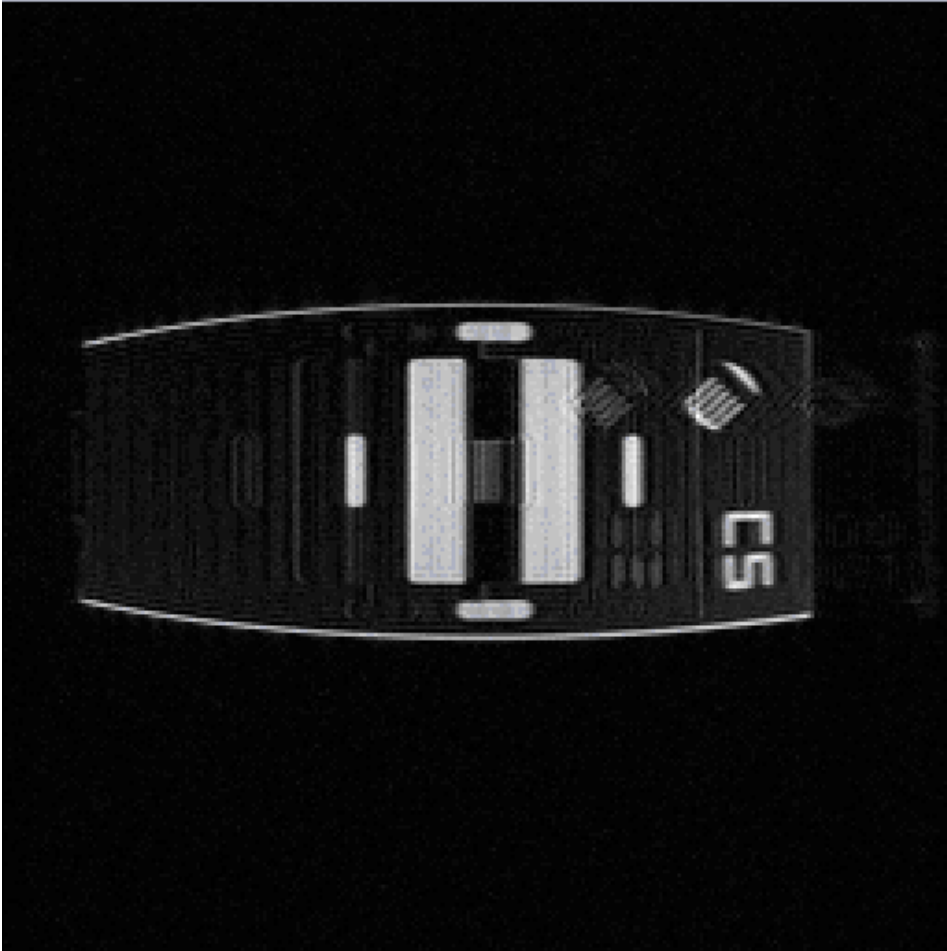


Figure B.2: Cornell PAWS-processed image using navigator *ithaca703*

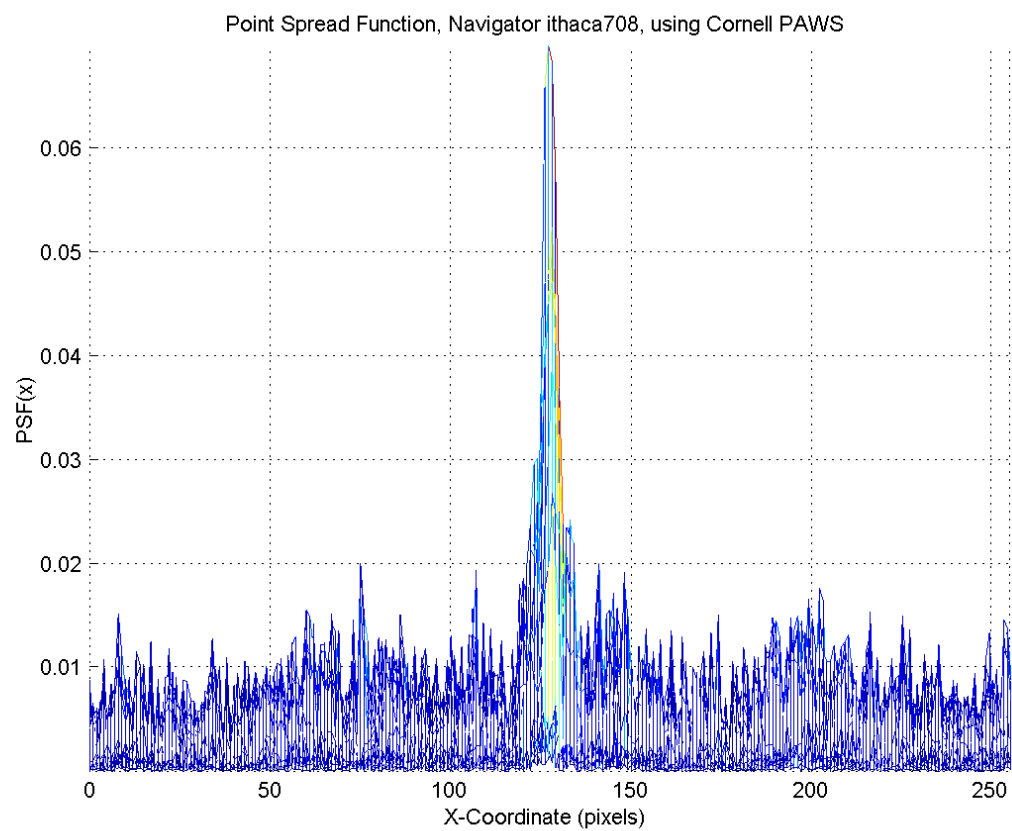


Figure B.3: Point spread function for Figure B.2

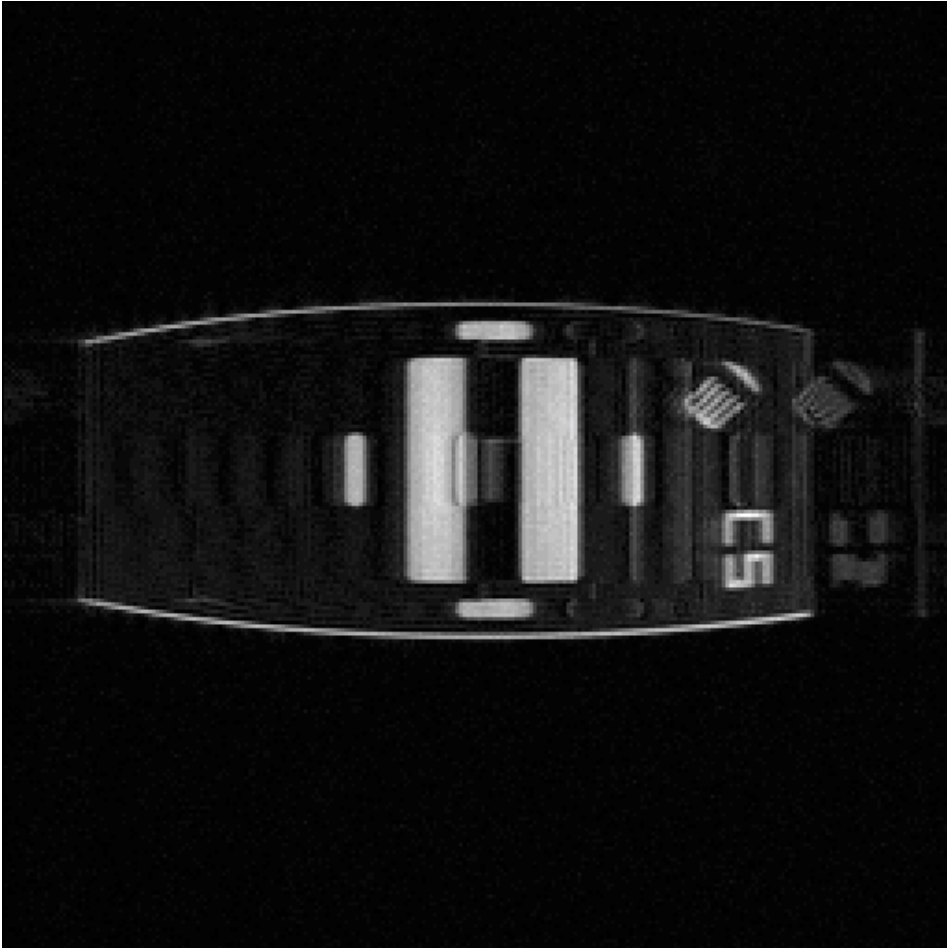


Figure B.4: Jhooti PAWS-processed image using navigator *ithaca703*

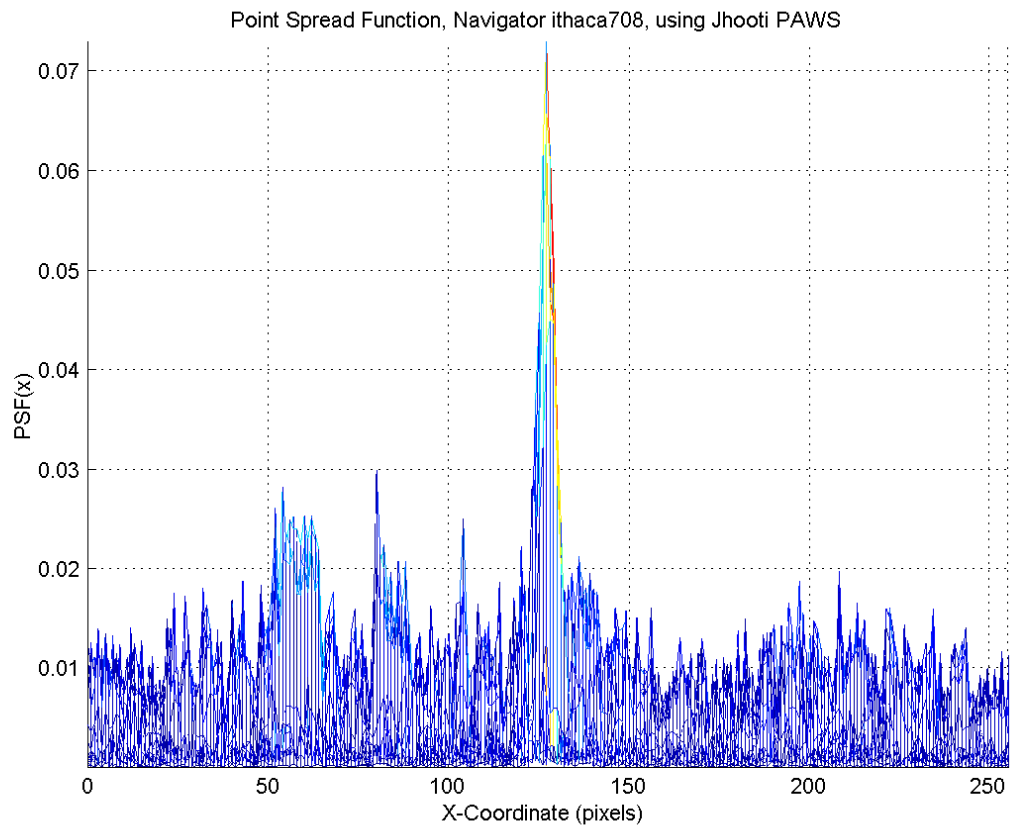


Figure B.5: Point spread function for Figure B.4

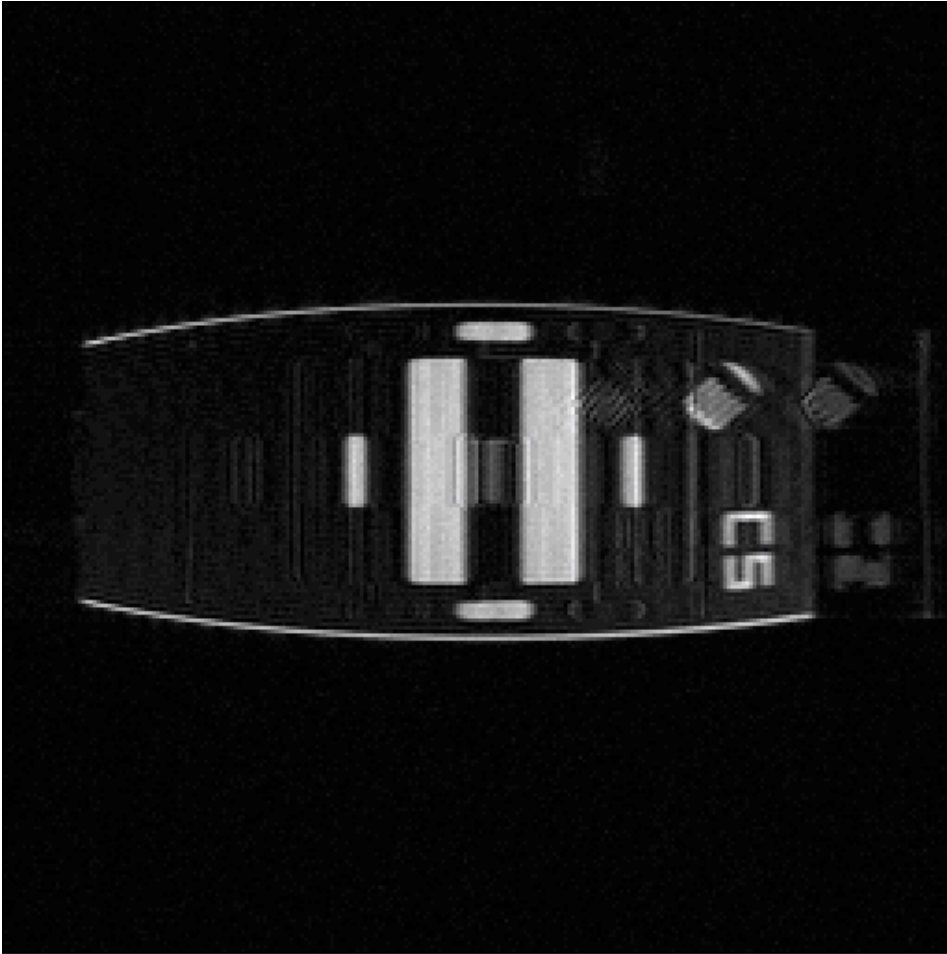


Figure B.6: Cornell PAWS-processed image using navigator *mrp1424*

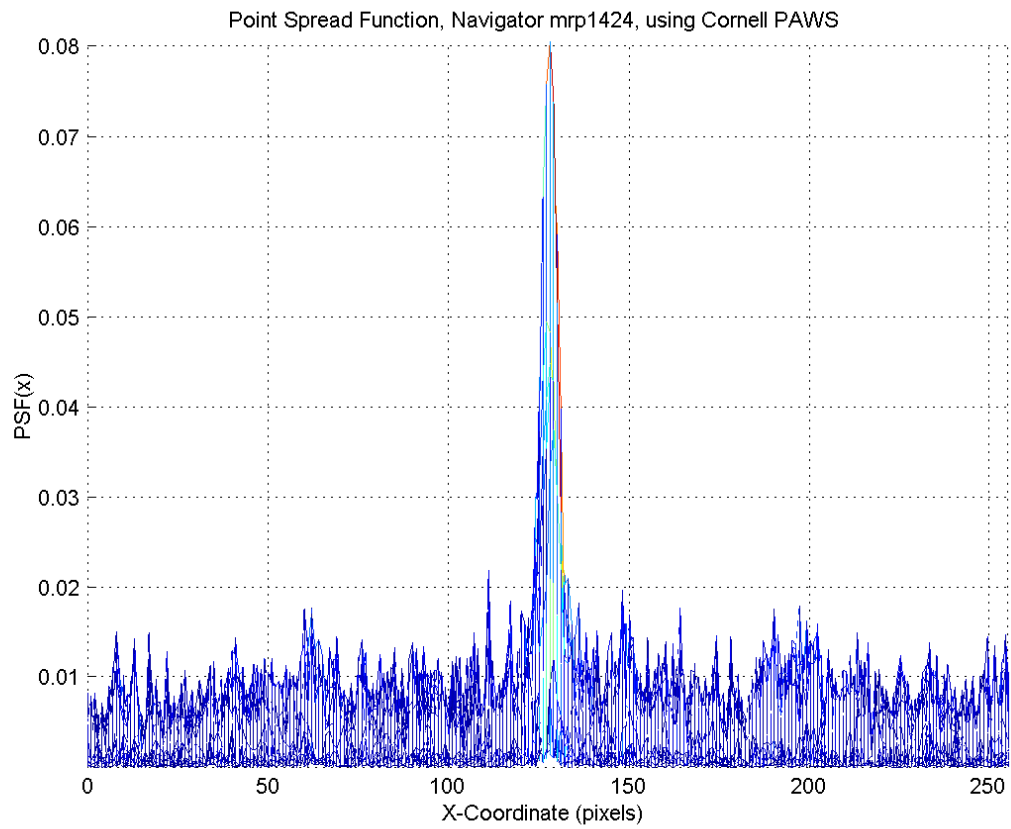


Figure B.7: Point spread function for Figure B.6

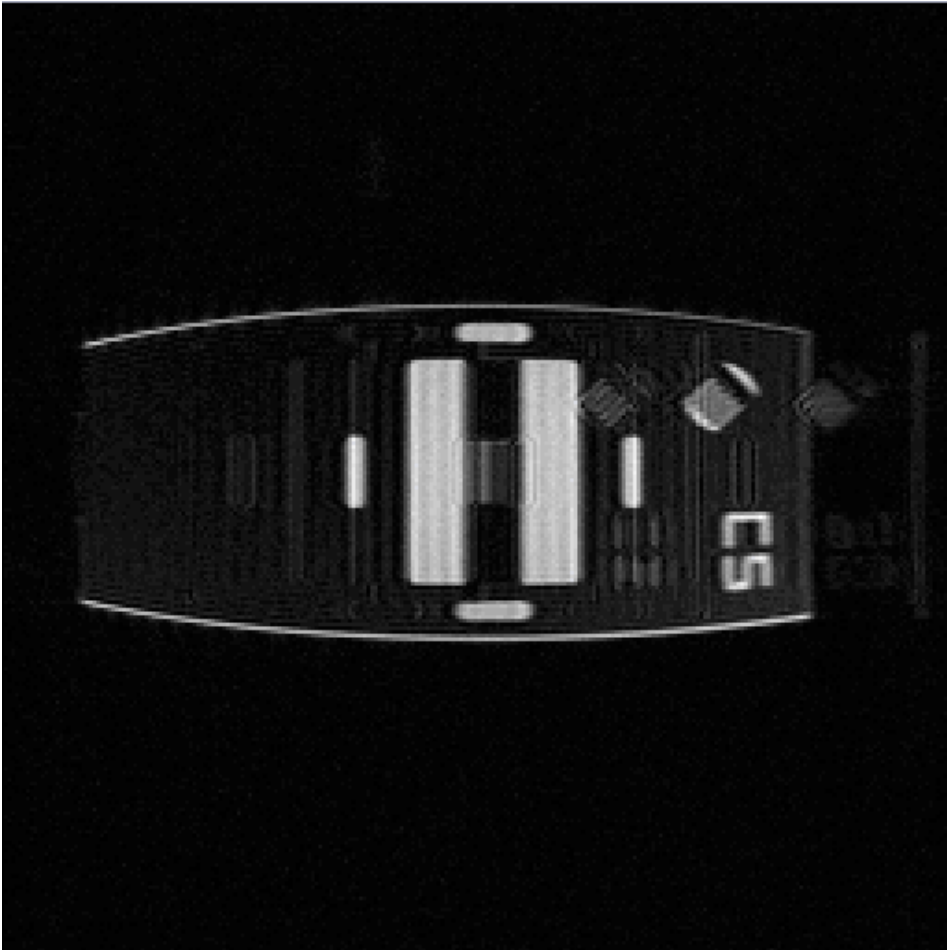


Figure B.8: Jhooti PAWS-processed image using navigator *mrp1424*

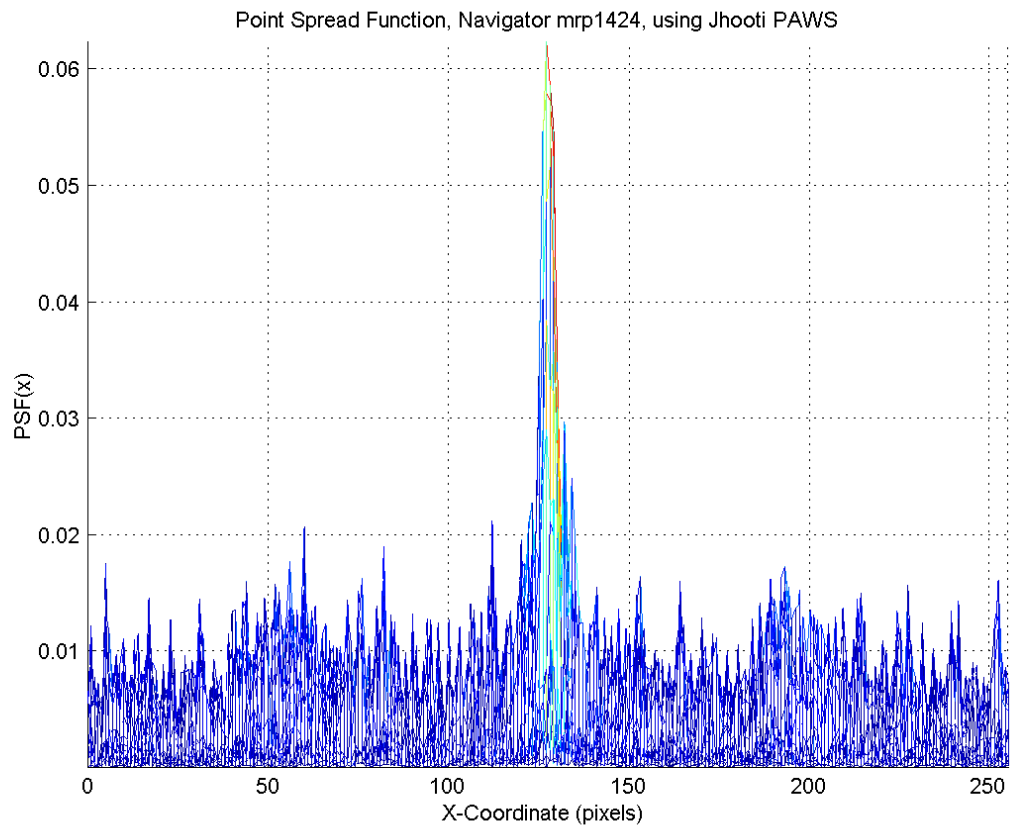


Figure B.9: Point spread function for Figure B.8

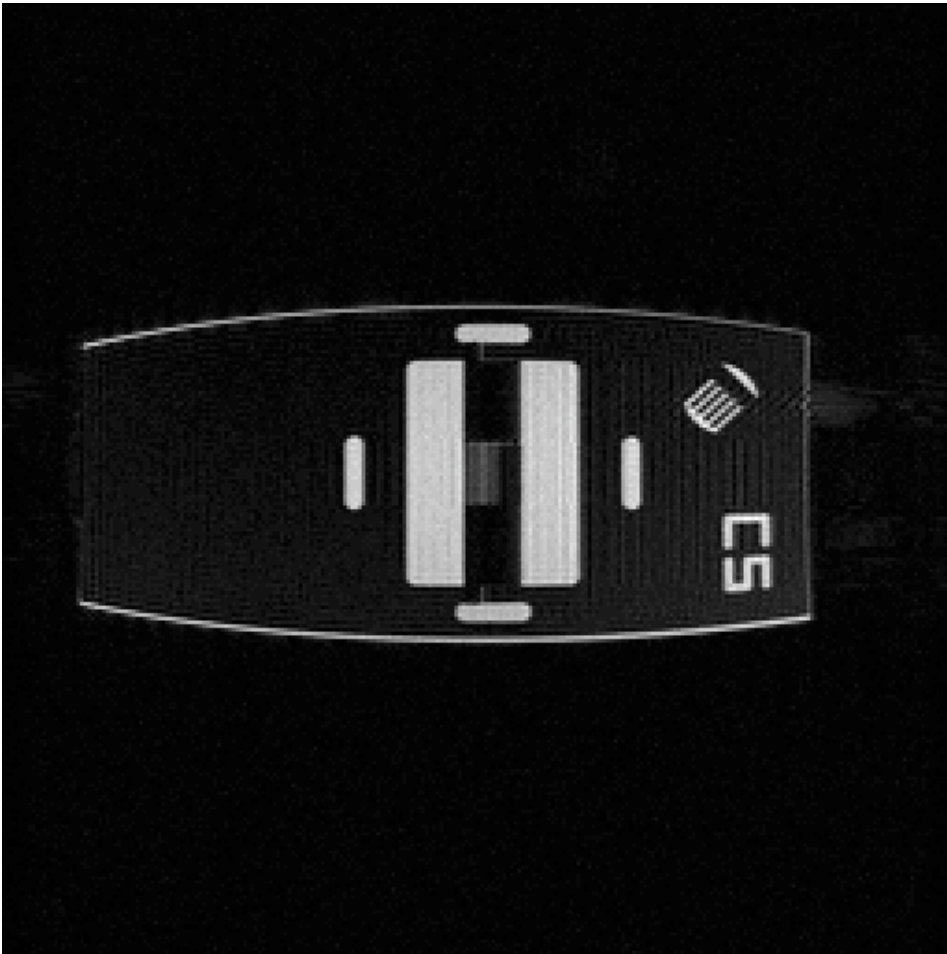


Figure B.10: Cornell PAWS-processed image using navigator *vnk1060*

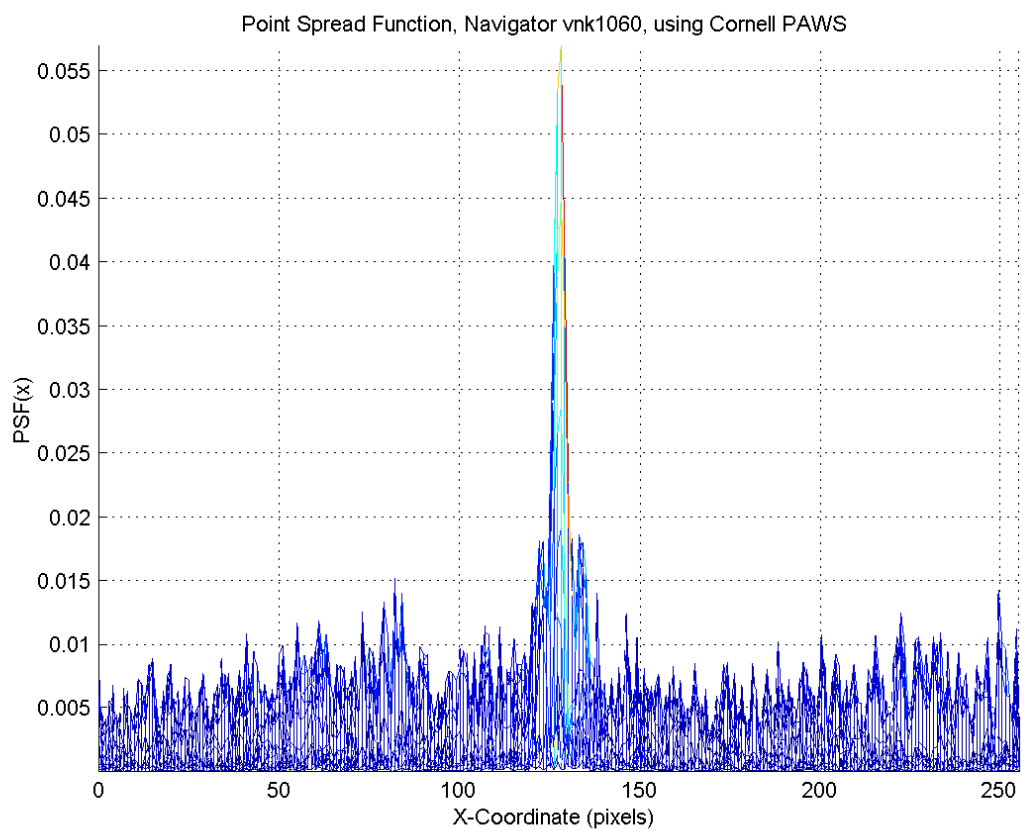


Figure B.11: Point spread function for Figure B.10

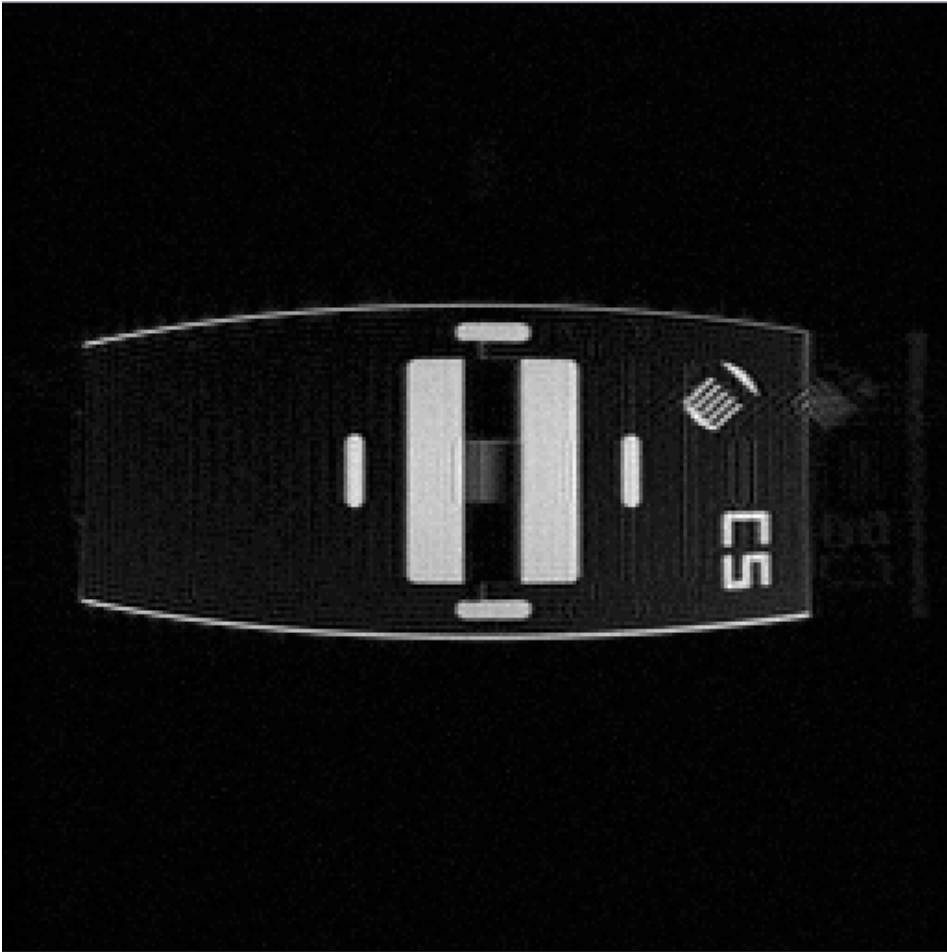


Figure B.12: Jhooti PAWS-processed image using navigator *vnk1060*

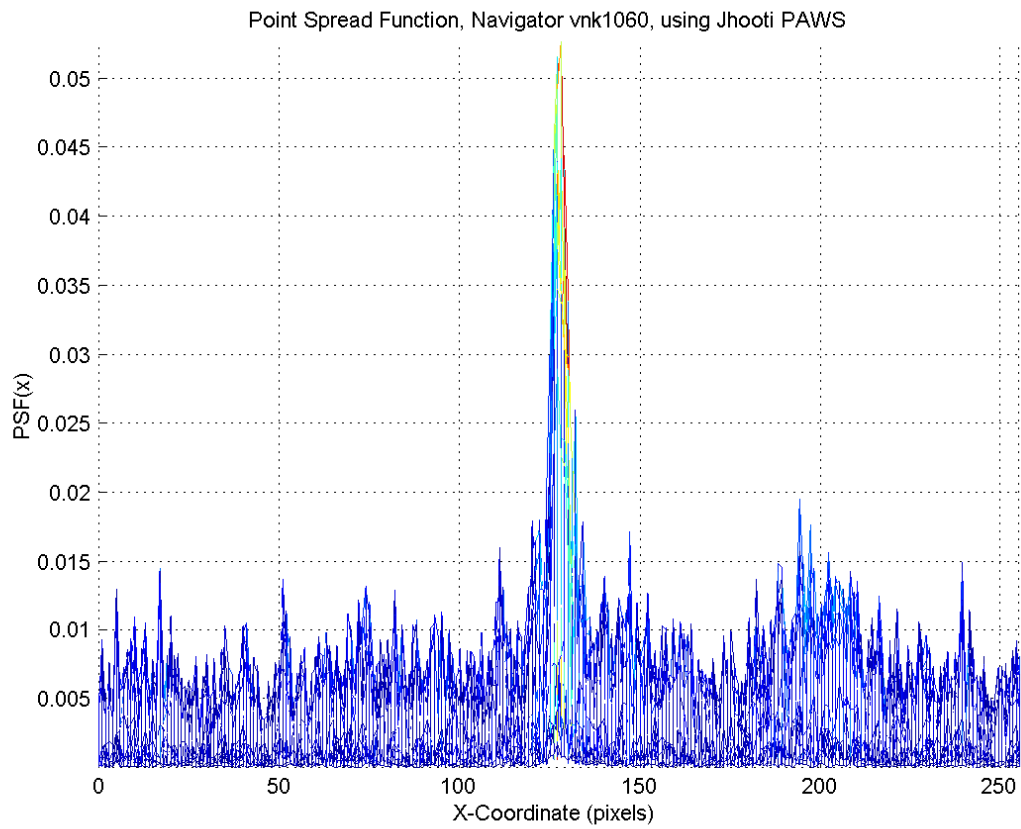


Figure B.13: Point spread function for Figure B.12

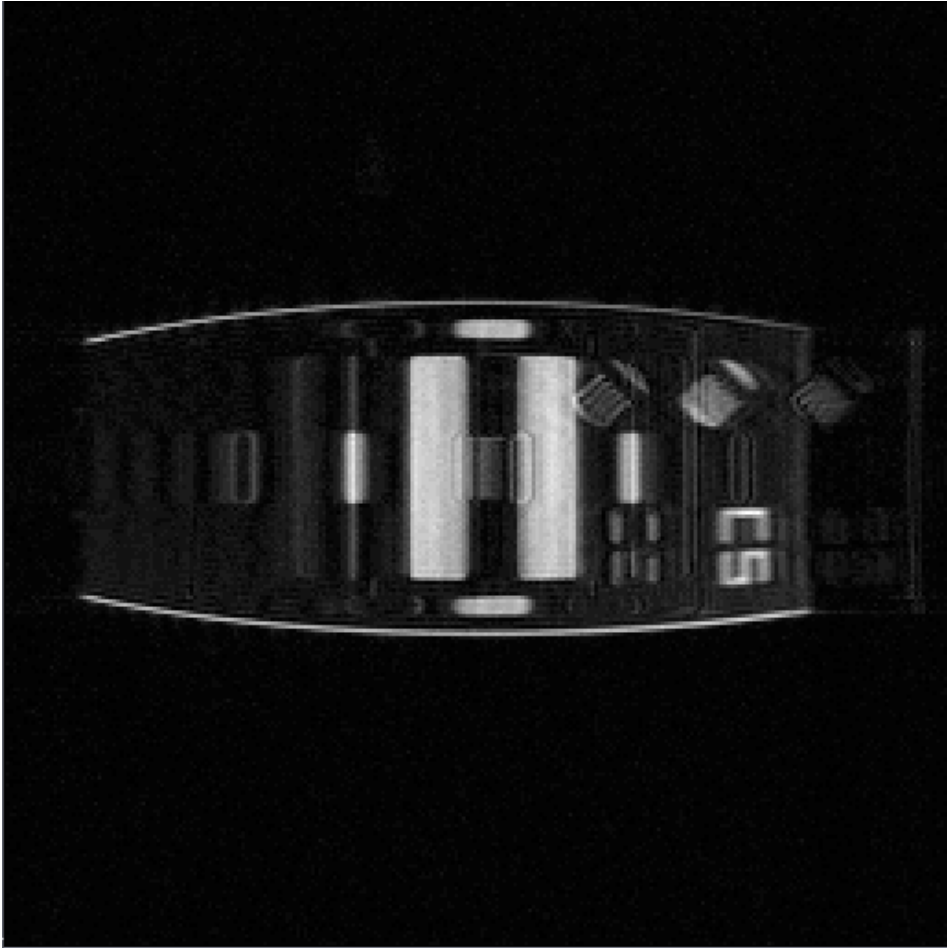


Figure B.14: Cornell PAWS-processed image using navigator *yifeng1128*

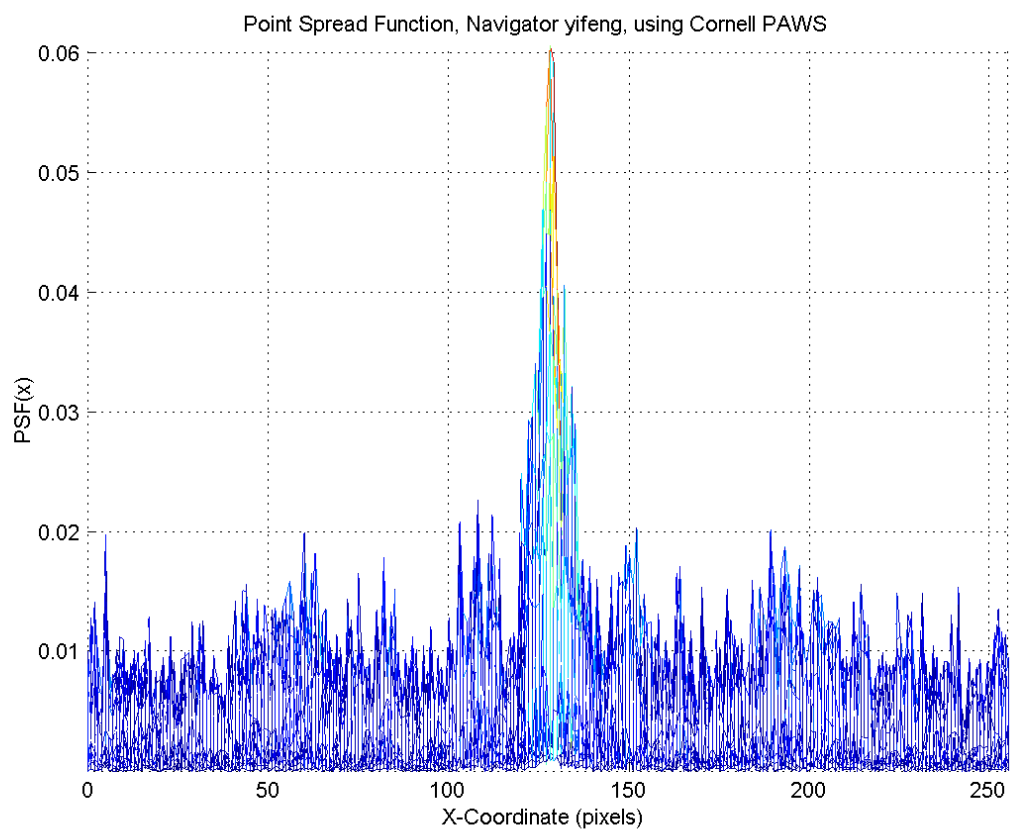


Figure B.15: Point spread function for Figure B.14

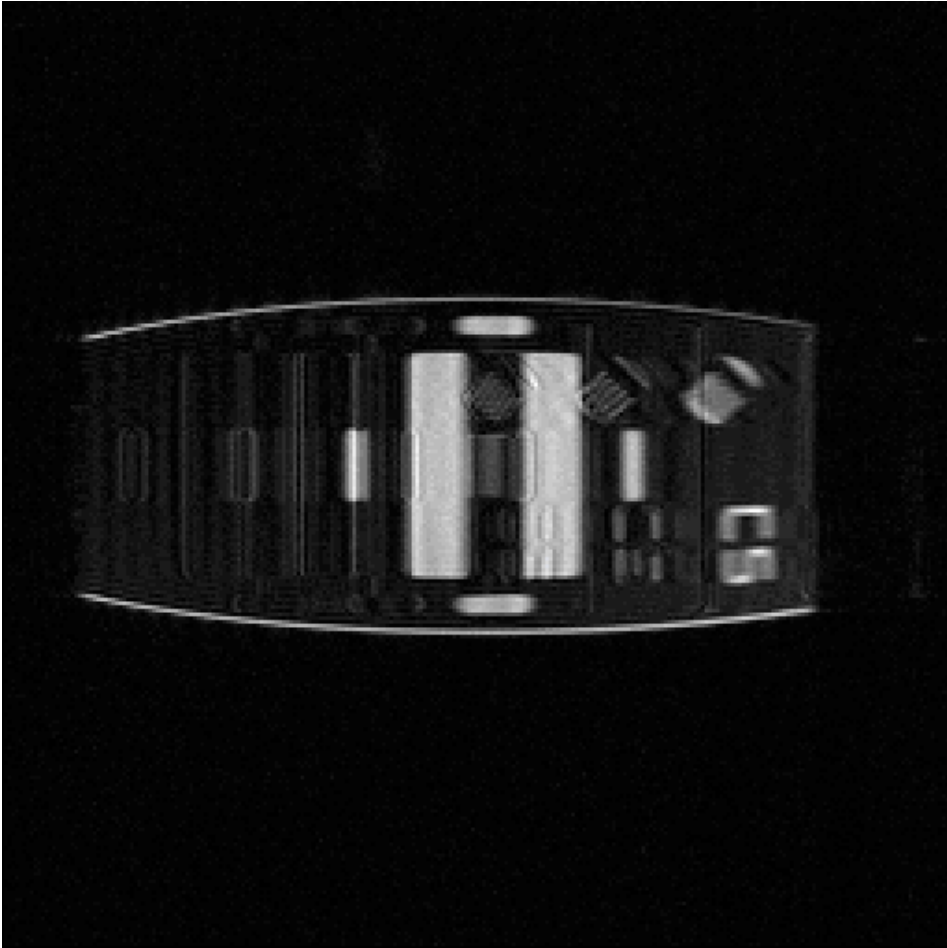


Figure B.16: Jhooti PAWS-processed image using navigator *yifeng1128*

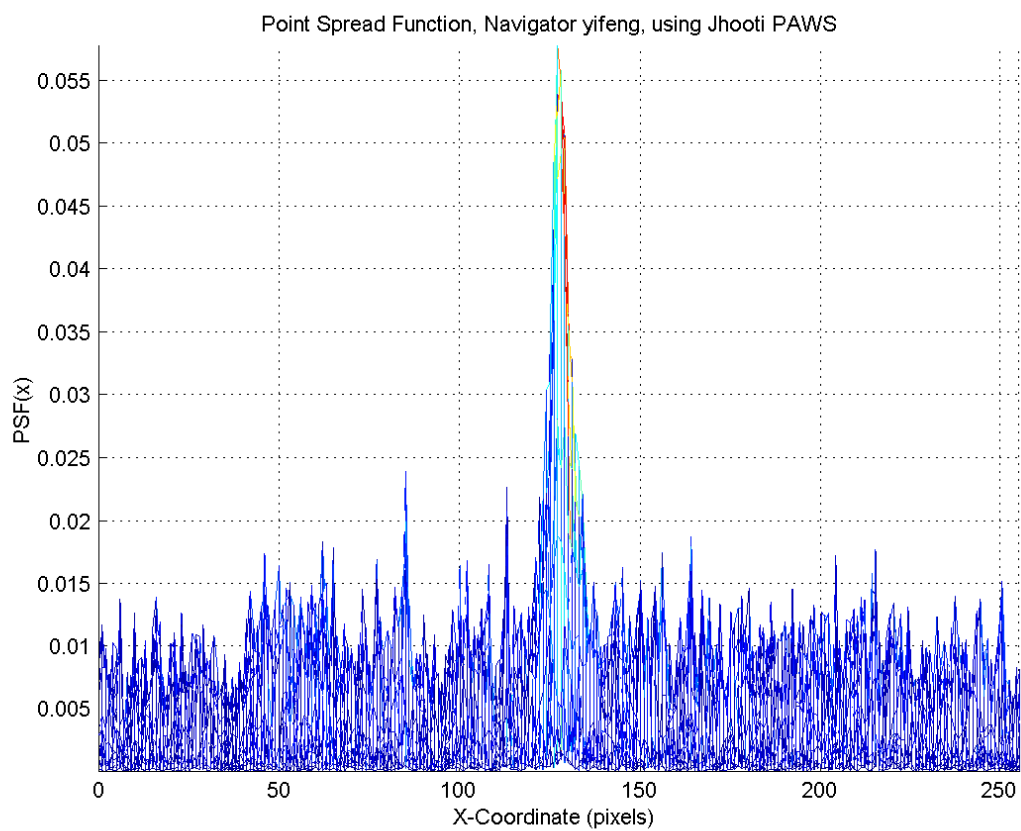


Figure B.17: Point spread function for Figure B.16

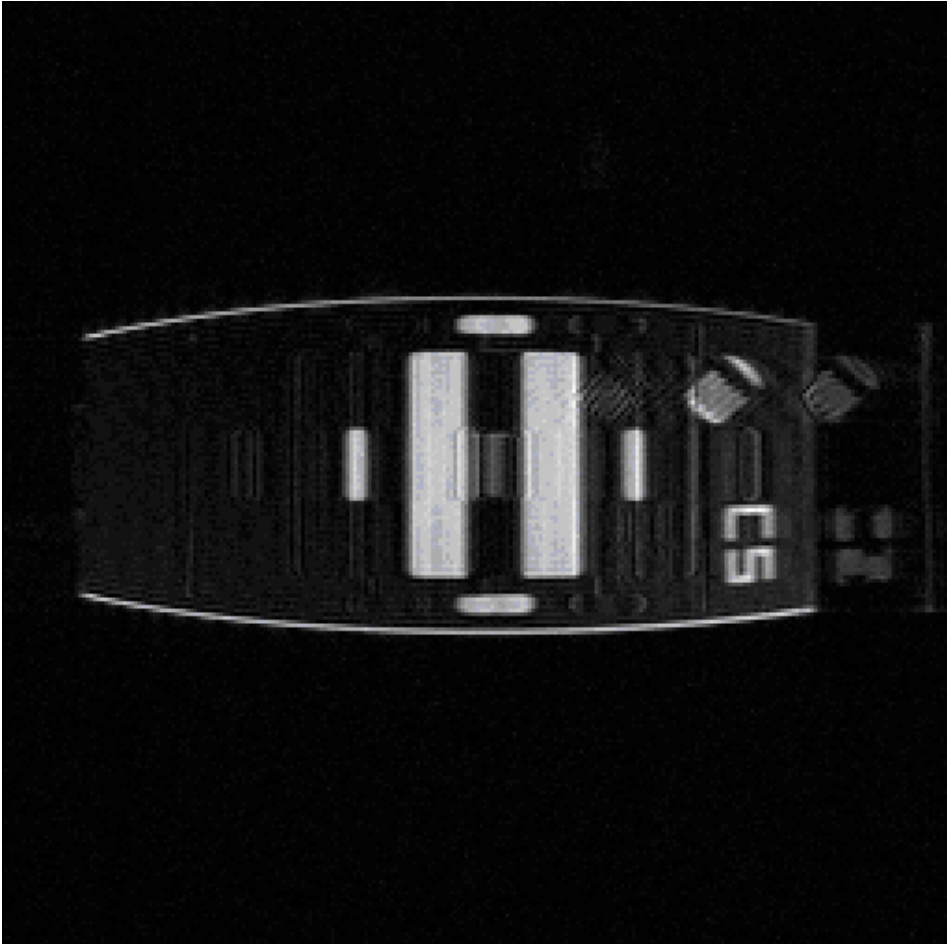


Figure B.18: Cornell PAWS-processed image using navigator *yw1322*

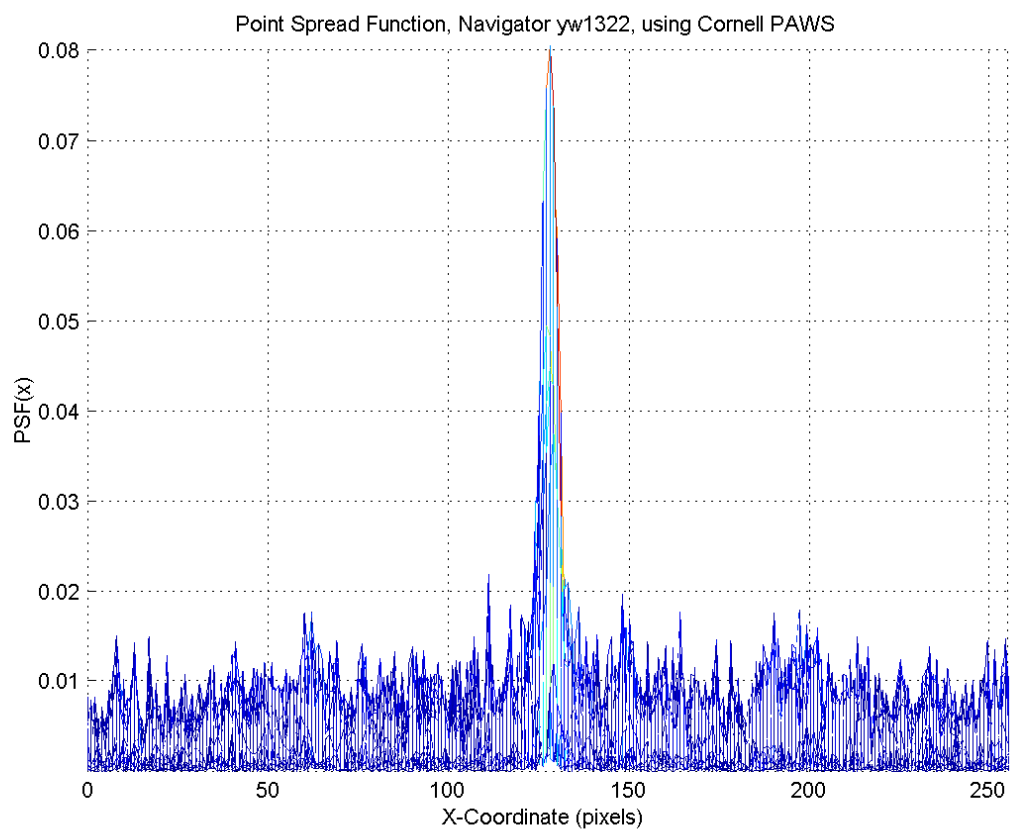


Figure B.19: Point spread function for Figure B.18

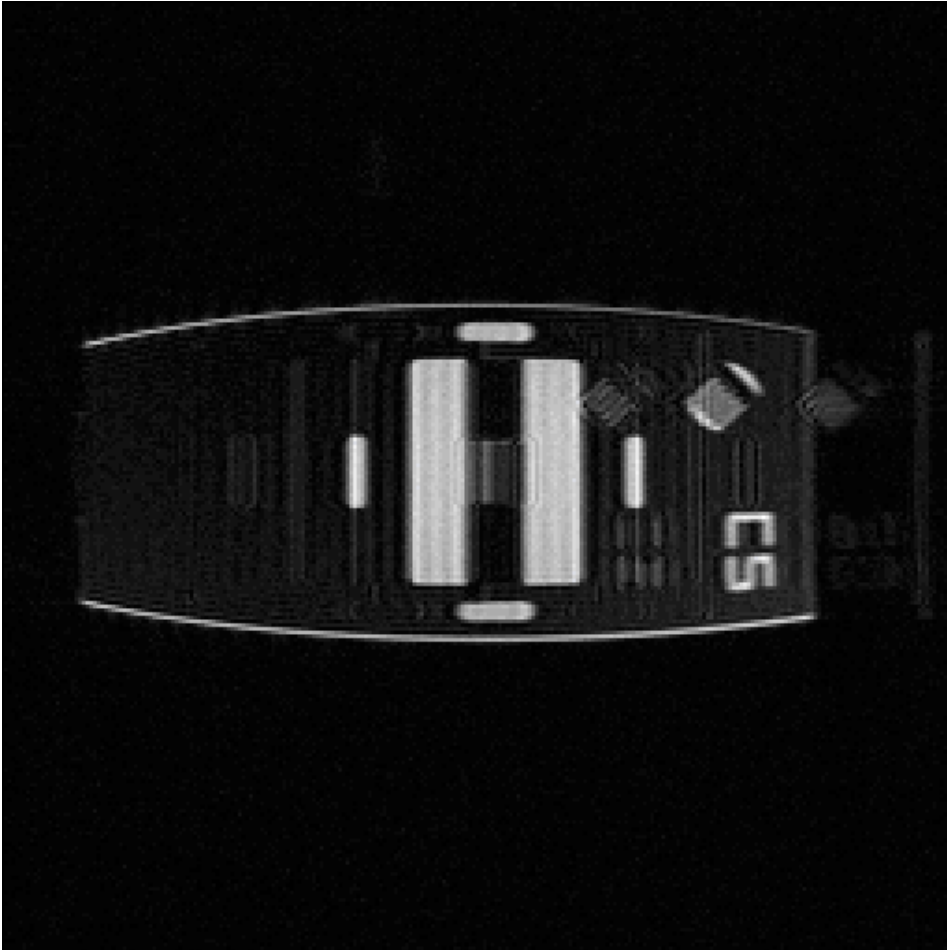


Figure B.20: Jhooti PAWS-processed image using navigator *yw1322*

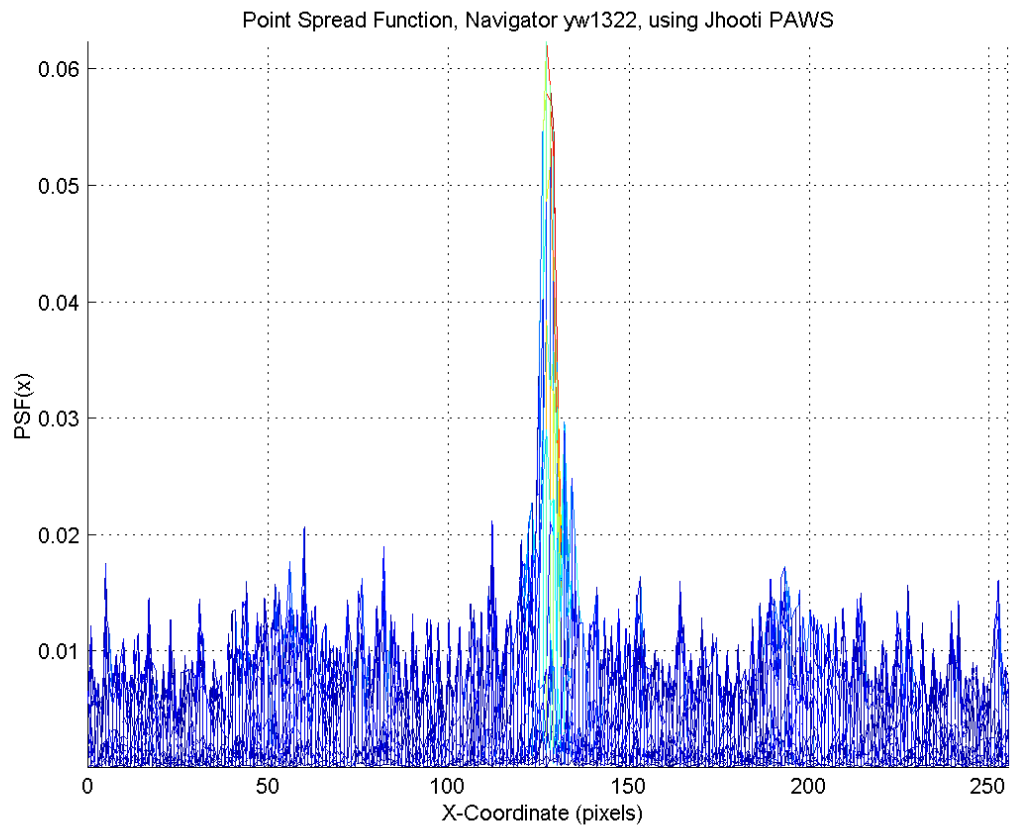


Figure B.21: Point spread function for Figure B.20

Bibliography

- [1] Jhooti, P., Gatehouse, P.D., Keegan, J., Bunce, N.H., Taylor, A.M., Firmin, D.N. (2000) *Phase Ordering with Automatic Window Selection (PAWS): A Novel Motion-Resistant Technique for 3D Coronary Imaging*, Magnetic Resonance in Medicine 43:470-480
- [2] Fontaine, F. (1999) *Multidimensional Fourier Analysis: 2-D Fourier Transform in polar coordinates, tomography, plane waves*, Dept. of Electrical Engineering, The Cooper Union, NY.
- [3] King, K.F., and Moran, P.R. (1984) *A unified description of NMR imaging, data-collection strategies and reconstruction*, Med Phys., Jan/Feb 11(1) 1-14
- [4] Liang, Z.P., Lauterbur, P.C. (2000) *Principles of Magnetic Resonance Imaging*, IEEE Press, New York.
- [5] Smith, H. J. (1989) *A Non-mathematical Approach to Basic MRI*, Medical Physics Pub. Corp., Madison.
- [6] Nguyen, T.D., Wang, Y., Watts, R., Mitchell, I. (2001) *k-Space Weighted Least-Squares Algorithm for Accurate and Fast Motion Extraction from Magnetic Resonance Navigator Echoes*, Magnetic Resonance in Medicine 46:1037-1040.
- [7] Ahn, C.B., Cho, Z.H. (1987) *A new phase correction method in NMR imaging based on autocorrelation and histogram analysis*, IEEE Trans Med Imaging MI-6:32-36.