GRAPPA navigators: motion correction with parallel imaging

P. Roder[1], and J. Willig-Onwuachi[1]

[1]Department of Physics, Grinnell College, Grinnell, Iowa, United States

Introduction: Typically, spatial encoding in MRI is performed entirely with gradient fields, while the use of multiple RF coils is primarily to extend anatomical coverage and boost SNR. However, signals acquired by different coils, with localized sensitivities, inherently contain spatial information. Partially parallel imaging (PPI) in MRI makes use of this built-in redundant spatial information for a number of possible benefits such as reducing scan time, increasing SNR or SNR efficiency, removing image artifacts, and replacing corrupt data. This project uses the SMASH navigator algorithm [1] for motion correction by comparing each successive k-space (phase-encode) line with predictions from previously acquired data. When significant difference is present, motion corruption is suspected and a second algorithm is used that attempts to correct it. A technique such as this based on correcting motion in successive lines progressively through k-space relies heavily on the accuracy of each correction step. The quality of the prediction of adjacent lines is crucial because it is used as the reference in the correction algorithm and because all subsequent corrections are based upon it. If one correction is not quite right, the resulting errors will propagate through the rest of the image and may result in severe image artifacts or even a failed attempt to correct a later instance of motion corruption. So, particularly in applications where multiple points of motion are expected during image acquisition, a more robust prediction method is crucial.

The SMASH navigators technique uses simultaneous acquisition of spatial harmonics (SMASH) [2] to predict one composite k-space line at a time. For example, the lines indexed by k_{PE}+1 from N coils are used to predict a single composite line indexed by k_{PE} for comparison with the composite zero-harmonic reconstruction of acquired k_{PE} lines [1]. SMASH reconstruction is relatively simple and straightforward to implement in this method, however, it is more prone to reconstruction errors than other k-space reconstruction techniques. Use of tailored SMASH [3] or generalized SMASH [4] may help reduce errors, but these methods do not share the reputation for robustness held by GRAPPA [5]. Traditionally GRAPPA uses two or more acquired phase encode lines on either side of a target line for reconstruction and the reconstruction weights are determined by fitting a kernel of adjacent data points from all coils to each point for separately acquired auto calibration signals (ACS). In this work, a one-sided variation of GRAPPA is applied that uses a kernel of points completely 'above' or 'below' the target phase encoding line—for example using lines k_{PE} +1 and k_{PE} +2 to predict line k_{PE} (see Fig. 1). Because GRAPPA weights are determined at multiple points along the read direction, it should be able to more accurately predict lines– an essential component for successful detection and correction of motion errors. The predicted lines are constructed point-by-point for each coil separately (as opposed to a single composite k-space line as in the SMASH-based techniques).

Methods: All simulations were performed using MATLAB 7.1 [Natick, MA, USA] on an Intel® Pentium® M processor running at 1.86 GHz with 1 GB RAM on a Microsoft Windows XP (SP2) Tablet PC Edition 2005 Gateway CX200X laptop. Data was simulated using a library image [Durer] and a simulated coil sensitivity set from a four-coil linear array (four co-planar 30x6 cm rectangular coils spaced by 9 cm in the narrow dimension—also oriented along the phase encoding direction—separated by 9 cm from the image plane). The simulation setup included a dataset with 128x128 resolution and a variance of Gaussian white noise as a fraction of the global maximum of the dataset at 0.02. Manual corruptions were made to the dataset corresponding to object motion in both the PE and FE directions. For the SMASH motion correction algorithm, it is assumed that the first data line (k_{PE} = 63*)* is correct. For the GRAPPA motion correction algorithm, it is assumed that the first two lines *(*k_{PE} = 63 & k_{PE} = 62*)* are correct. All motion was 2D translational motion between 'acquisition' of phase-encoding lines.

For simplicity, when needed, a zero-harmonic SMASH reconstruction was used to combine lines of data from multiple coils into a single composite line for analysis, comparison, or image reconstruction. Each line considered in succession was compared to predictions made from previously considered lines. Differences between the line under consideration and the predicted line were attributed to motion corruption. If a corruption was found, a multi-parameter least squares fit was used to determine the parameters required to correct the motion. For the GRAPPA simulations, the middle 64 k_{PE} lines of were used as ACS lines with independent random Gaussian noise of 0.02 to better simulate separate acquisition. Once GRAPPA and SMASH motion correction algorithms were run, a pixel-by-pixel RMS difference between the reconstructed image and the original image was taken to determine the error in the respective reconstruction algorithm.

Results and Discussion: Figs. 2 and 3 show the reference and corrupted images, respectively. Figs. 4 and 5 are examples of the resulting reconstructions after the corrupted image was run through the detection and correction algorithm using SMASH and GRAPPA, respectively. Note the GRAPPA image appears to be slightly sharper than the SMASH image. The RMS errors (Table 1) associated with each image show that SMASH tended to generate errors twice as big as GRAPPA. This led us to believe that GRAPPA was indeed a more robust method than SMASH for this application.

Atkinson, et al., showed in their SMASH Navigators paper how the corrupted data, having been Fourier transformed in the k_{FE} direction, could be corrected by applying a shift of magnitude Δx in the *x* (frequency-encoding) direction and a phase of ϕ_y in the *y* (phase-encoding) direction. However, we found that an obvious but novel method which applied a phase in both k_{PE} and k_{FE} directions using data that had not been Fourier transformed seemed to work much more efficiently.

Both SMASH and GRAPPA motion correction algorithms worked well progressing from the maximum k_{PE} to the minimum of k_{PE} (top to bottom). The SMASH algorithm worked consistently faster, probably due to the fact that the GRAPPA weight generation and motion correction programs take longer to execute in MATLAB than the SMASH versions, particularly for smaller tolerance (0.7). However, as the tolerance approaches 0.9, the reconstruction times became nearly identical.

The GRAPPA algorithm consistently produced a crisper image than the SMASH algorithm. Also, as the error tolerance increased, GRAPPA seemed to need fewer error corrections than SMASH, suggesting improved accuracy over SMASH. Recall that any small error in the correction gets propagated to subsequent lines of data, so that eventually the corrected data is sufficiently off to appear motion corrupted, requiring correction again even if no corruption has occurred.

With artifact reduction being the most important factor, GRAPPA seemed to consistently work more effectively than SMASH. However, there are certain factors that were not taken into account during the simulation. First, in vivo motion corrupted data was not used and thus it is hard to accurately tell which algorithm would be more fitting for real life application. Lastly, the motion that was simulated corresponds only to motion between the acquisition of adjacent lines of k-space, not during the acquisition of a read-out line.

References: [1] MRM 49:493-500 (2003), [2] MRM 38:591-603 (1987), [3] MRM 44:243-251 (200), [4] MRM 47:160-170 (2002), [5] MRM 47:1202-1210 (2002).

Tol	Corrections SM/GR	Time (s) SM/GR	SMASH Error	GRAPPA Error
.7	12/18	10.13/19.59	.194	.373
.75	6/8	7.03/12.72	.197	.084
.8	5/4	6.30/9.55	.183	.085
.85	4/2	5.55/8.14	.202	.083
.9	4/1	5.68/7.36	.198	.079

Table 1

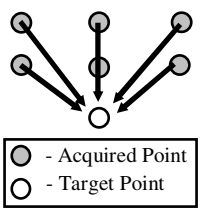

Figure 1

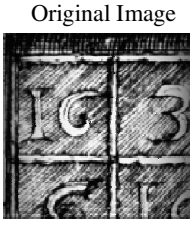

Figure 2

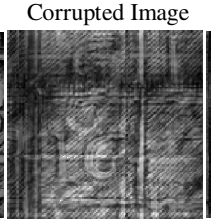

Figure 3

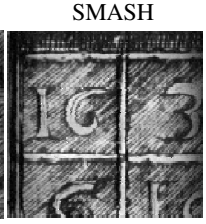

Figure 4

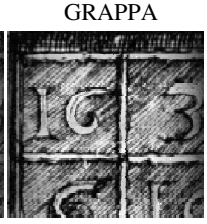

Figure 5