Author: R.J. Nijboer	Date of issue: 2005-Aug-01 Kind of issue: Public	Scope: Project Documentation Doc.nr.: LOFAR-ASTRON-DOC-???	LOFAR
	Status: Draft Revision nr.: 0.1	File: ftp://rnijboer.astron.nl/???	

UVBrick: Predicting uv-data from Patch Images A second round of design and implementation

R.J. Nijboer

Verified:			
Name	Signature	Date	Rev.nr.
J.E. Noordam	o.p.v.	2005-???-??	0.1
J.E. Noordam	....	....	

Accepted:		
Work Package Manager	System Engineering Manager	Program Manager
J.E. Noordam	K. van der Schaaf	J. Reitsma
.... *date:*	.... *date:*	.... *date:*

©ASTRON 2005
All rights are reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

Author: R.J. Nijboer	Date of issue: 2005-Aug-01 Kind of issue: Public	Scope: Project Documentation Doc.nr.: LOFAR-ASTRON-DOC-???	
	Status: Draft Revision nr.: 0.1	File: ftp://rnijboer.astron.nl/???	

Distribution list:

Group:	For Information:
ASTRON	ASTRON
R. Assendorp	J.D. Bregman
M.A. Brentjes	A.G. de Bruijn
W.N. Brouw	M.A. van Haarlem
G. van Diepen	H.A. Holties
M. Mevius	K. van der Schaaf
J.E. Noordam	C.M. de Vos
T.A. Oosterloo	
O.M. Smirnov	
S.B. Yatawatta	
NRC	
A.G. Willis	

Document revision:

Revision	Date	Section	Page(s)	Modification
0.1	2005-Aug-01	-	-	Creation

Author: R.J. Nijboer	Date of issue: 2005-Aug-01 Kind of issue: Public	Scope: Project Documentation Doc.nr.: LOFAR-ASTRON-DOC-???
	Status: Draft Revision nr.: 0.1	File: ftp://rnijboer.astron.nl/???

Abstract

This document describes the design and implementation of the MeqTrees and MeqNodes that are used for the prediction of *uv*-data from LSM Patch Images. In a predict tree the following new nodes are used: MeqPatchComposer, MeqStokes, MeqFFTBrick, and MeqUVInterpol. These nodes are used for predicting *uv*-data from extended sources or from multiple point sources. In order to do so a Sky Image (LSM Patch Image) is FFT'd into an image of *uv*-data. The resulting *uv*-data set is called a UVBrick.

Contents

1 Introduction

LOFAR Self-Calibration is based on adaptive models. Model parameters are fitted to measured uv-data. One of the models is for the Sky. Therefore, uv-data must be predicted for model sources from a Local Sky Model (LSM) [1]. This is achieved by the concept of P-Units (Predict or Peeling Units) which come in two flavors: Point-like or Patch-like. A single point source is a Point-like P-Unit in the LSM and its uv-data is predicted by means of a DFT Node. This node is connected to the LSM and directly fills a Request Cells / Vells with uv-data.

For extended sources or multiple point sources the prediction of uv-data is less trivial. It was decided that, for now, all extended sources and multiple point sources will be dealt with using images. That is, an extended source (be it parameterized or an image of CLEAN components) will be described by means of Patch-like P-Unit, which is a (Temporary) Patch Image [1]. This image will be FFT'd into a uv-image. This uv-image can then be used to fill the Request Cells / Vells.

In the Predict Tree for Patch-like P-Units other Nodes may be inserted. These additional nodes then may implement transformations from intrinsic fluxes into apparent fluxes or they may apply image plane effects. We will not consider such additional nodes now. However, the MeqTree system is flexible enough to add them later.

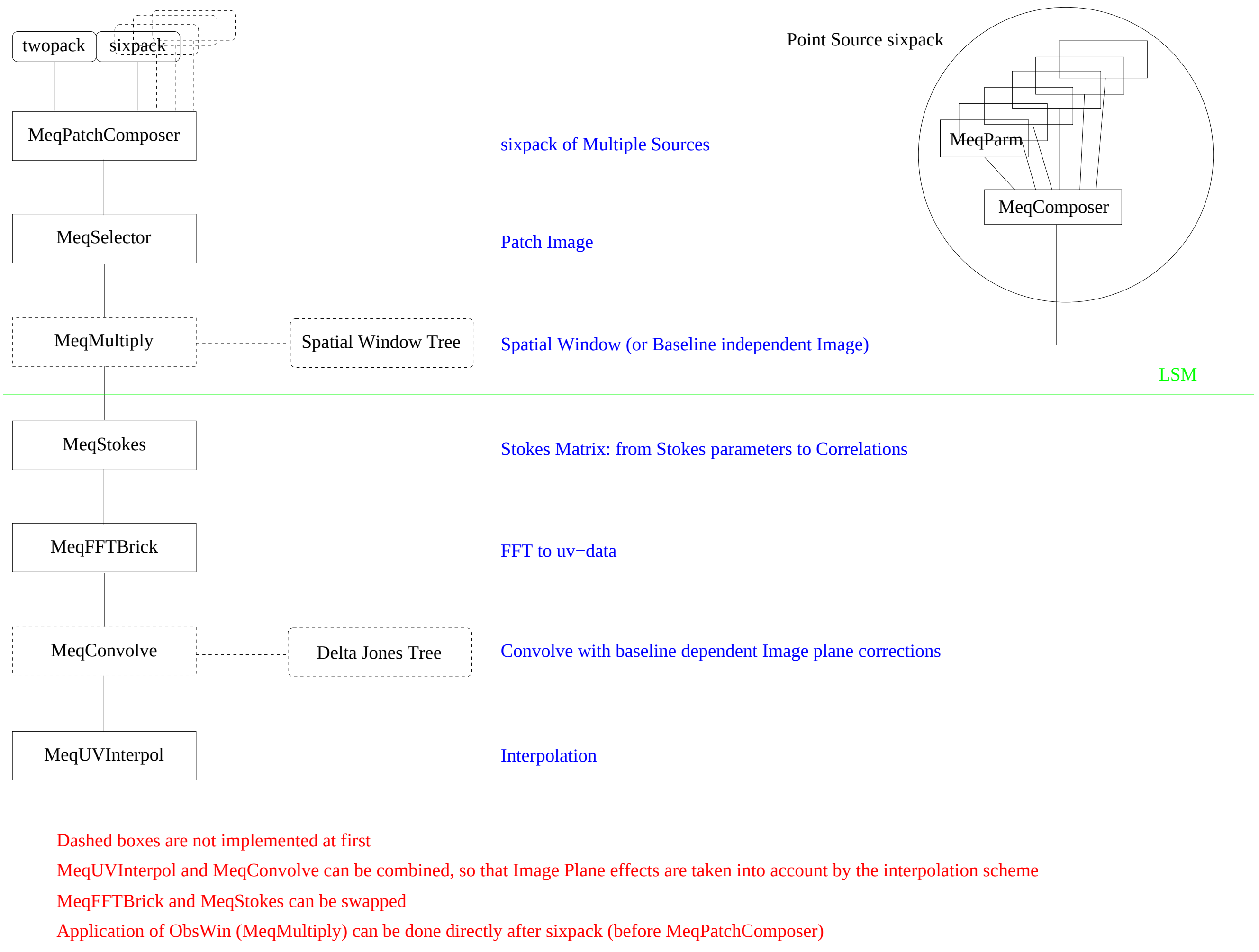
This document describes the design and implementation of the MeqTrees and MeqNodes that are used for the prediction of uv-data from LSM Patch Images. In such a predict tree the following new nodes are used: MeqPatchComposer, MeqStokes, MeqFFTBrick, and MeqUVInterpol. All these nodes are described in the following sections. But first an overview of the Patch Image predict tree is given.

2 Overview of the Predict from Image Patches

In figure 1 a tree for the prediction of uv-data from a Patch Image is shown. The Patch Image is returned in the MeqSelector node. This node has a sixpack as child: the MeqPatchComposer node. The latter node creates a Patch sixpack from a phase center twopack and a point source sixpack for every point source it has as child.

The ObsWin is applied in the MeqMultiply node. Possibly this can also be applied higher up the tree. In any case we do not consider it for now. The border between the Local Sky Model and the User Forest lies between the MeqMultiply node for the ObsWin and the MeqStokes node.

The MeqStokes node transforms the Stokes parameters into Sky brightnesses, depending on polarization type. The MeqFFTBrick Fourier transforms the Sky Brightness into coherencies, which may then be convolved with delta Jones matrices in order to apply Image plane effects. The latter node is also not yet implemented. Finally, a frequency-time Vells is filled in the MeqUVInterpol by interpolating the



twopack
sixpack
Point Source sixpack
MeqParm
MeqComposer
MeqPatchComposer
sixpack of Multiple Sources
MeqSelector
Patch Image
MeqMultiply
Spatial Window Tree
Spatial Window (or Baseline independent Image)
LSM
MeqStokes
Stokes Matrix: from Stokes parameters to Correlations
MeqFFTBrick
FFT to uv−data
MeqConvolve
Delta Jones Tree
Convolve with baseline dependent Image plane corrections
MeqUVInterpol
Interpolation
Dashed boxes are not implemented at first
MeqUVInterpol and MeqConvolve can be combined, so that Image Plane effects are taken into account by the interpolation scheme
MeqFFTBrick and MeqStokes can be swapped
Application of ObsWin (MeqMultiply) can be done directly after sixpack (before MeqPatchComposer)

Figure 1: A Tree describing the prediction of uv-data from Patch-like P-Units

uv-plane.

This tree is valid for Patches that are made up from multiple Point sources. By extending the MeqPatch-Composer node extended sources may be treated in the same way. This, however, will be dealt with in a later stage.

3 The Phase Center twopack

The Phase Center twopack is a MeqComposer node returning two Vells planes. Plane 0 contains the RA position of the Phase Center of the Patch Image. Plane 1 contains the Dec position of the Phase Center of the Patch Image. Both positions are measured in radians.

The MeqComposer node will thus have two children: the two sub-trees for the RA and Dec position.

4 The Point Source sixpack

The Point Source sixpack is a MeqComposer node returning six Vells planes. Plane 0 contains the RA position of the Point Source. Plane 1 contains the Dec position of the Point Source. Both positions are measured in radians. Plane 2 to 5 contain the StokesI, StokesQ, StokesU, and StokesV values.

The MeqComposer node will thus have six children: the six sub-trees for RA, Dec, StokesI, StokesQ, StokesU, and StokesV.

Note: It could be that some of the Stokes planes are missing. E.g. for processing intensity data only, or for processing sources having only linear polarization (I,Q,U). Therefore, more generally the sixpack is an n-pack, where n is at least three: RA, Dec, and StokesI.

5 MeqPatchComposer

The MeqPatchComposer node is a node that creates a gridded sixpack from its children: it combines sources (and Patches) into a Patch Image. It generates itself a sixpack, so that it may be used as child in another MeqPatchComposer node.

The node expects a frequency / time Request from its parent and sends a frequency / time / l / m Request to its children. Hence the node has internal information on how to make an l / m grid! When it gets a Result from its children it Phase Shifts and Regrids it to make its own frequency / time / l / m Result. This Result holds the sixpack for the Patch Image.

Note 1: The l / m grid of the Request is just a hint. For a Point Source it is ignored or a one pixel l / m grid is returned. For a Parameterised Extended Source the Request grid is used. An Imaged Extended Source just returns its own grid.

Note 2: l / m coordinates may be replaced by frequency dependent l' / m' coordinates depending whether the UVBrick will be in wavelengths or meters. A first implementation will be based on l' / m'. We have

$$(u, v, w) = f\,(d_x, d_y, d_z)\,/c$$

and

$$(l', m', n') = f\,(l, m, n)\,/c.$$

So

$$ul + vm + wn = d_x l' + d_y m' + d_z n'.$$

Note 3: Internally (RA, Dec) coordinates from the sixpack are phase shifted and transformed into (l,m) coordinates (or (l',m') coordinates):

$$\begin{aligned} l &= \cos(Dec)\sin(RA_0 - RA) \\ m &= \cos(Dec_0)\sin(Dec) - \sin(Dec_0)\cos(Dec)\cos(RA_0 - RA), \end{aligned}$$

where (RA_0, Dec_0) are the RA and Dec coordinates of the Phase Center of the Patch Image.

Note 4: The grid spacing and extent of the final UVBrick (result of the MeqFFTBrick node) is determined in this node!

6 MeqSelector

The MeqSelector node strips off the RA and Dec planes. Or, more general, it strips off planes 0 and 1.

Hence, it just parses the Request.

And, it expects a child Result having more than two Vells planes.

7 MeqMultiply

The MeqMultiply node applies the Spatial Window before the Stokes matrix is applied. The only (station averaged) Jones matrix of which the Kronecker product with itself commutes with the Stokes matrix is a complex gain matrix:

$$\mathsf{J} = \begin{pmatrix} c & 0 \\ 0 & c \end{pmatrix},$$

where c is complex.

This means that by applying this node (having a different value for c for every point in the sky), we can go from intrinsic fluxes to apparent fluxes.

8 MeqStokes

The MeqStokes node transforms Stokes (I,Q,U,V) into Sky Brightnesses. For an instrument having feeds based on linear polarization the transformation reads

$$(I,Q,U,V) \rightarrow (B_{11}, B_{12}, B_{21}, B_{22}) = (I+Q, U+iV, U-iV, I-Q)/2$$

For circular polarization feeds it reads

$$(I,Q,U,V) \rightarrow (B_{11}, B_{12}, B_{21}, B_{22}) = (I+V, Q+iU, Q-iU, I-V)/2$$

Note: We really have to check definitions of signs and factors of 2!

Hence, the node gets four planes as input and combines them into four new planes, depending on polarization type.

So, the node parses the Request.

The node expects a Result having four planes and combines them into a two by two tensor:

$$\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

However, sometimes planes may be missing. E.g., only the StokesI plane is present, or the StokesQ/U/V planes are empty. It that case the node may return a scalar result: $I/2$. We will consider all exceptions in more detail in a later stage.

Note: the Stokes matrix transforms the real Sky into a complex Brightness. Hence, by applying the Stokes matrix before the FFT, we have to do a complex-to-complex FFT, instead of a real-to-complex FFT. The alternative would be to apply the Stokes matrix after the FFT.

9 MeqFFTBrick

The MeqFFT Node performs a 2D FFT from l(') / m(') to u (d_x) / v (d_y) for every frequency and brightness cross-section. Apart from the FFT also data for the (numerical) derivatives (using finite differences) is stored in additional planes. This additional data is only needed when a Bi-Cubic Hermite interpolation is used in the MeqUVInterpol node.

Because of the additional information, the MeqFFTBrick could be implemented as a special case of a more general MeqFFT node.

The FFT will be implemented using standard library routines. For instance, the FFTW implementation. We will have to look into this.

In order to have enough points per wavelength for the subsequent interpolation in the MeqUVInterpol node, the l(') / m(') may be padded with zero's. This zero padding also reduces the effect of aliasing due to periodic continuation of the Sky image.

Smart caching must be used in order to stop Requests from going up, whenever the contents of the UVBrick is still valid to satisfy a Request. This can be obtained with a `cache_policy` set equal to -1, 0, or 1 (see the Timba Wiki Page).

The MeqFFTBrick can have a generalized implementation. Requests are parsed. A Result is expected to have two designated axes on which the FFT is going to be applied. All other axes are sliced through. This could just be a MeqFFT node. Optionally, additional planes containing information on the derivatives of the Fourier Transformed function may be added. This is the information needed for efficient use of the Bi-Cubic Hermite interpolation scheme. This would make the node more specific than a general MeqFFT node.

Note: We start with a complex-to-complex FFT. Later on we may decide what other FFT transforms are needed.

10 MeqUVInterpol

This node interpolates UVBrick data. It first transforms frequency / time cells into u / v (d_x/d_y) cells using elliptical trajectories. Then the center value is used to get an interpolation value. Interpolating the UVBrick in meters means that the interpolation point is the same for every frequency plane and every correlation plane. So this is very efficient.

In our initial version we have options for Bi-linear interpolation, Bi-Cubic Hermite interpolation and 4th-order polynomial interpolation. The Bi-Cubic Hermite version uses the additional planes for the derivatives from the UVBrick.

Linear interpolation is first order accurate (the error is of second order). Bi-Cubic interpolation is third order accurate (the error is of fourth order). For the Bi-Cubic interpolation we need to store 4 times as many data. For the same amount of data the linear interpolation could work on images that are twice as large (2D). Still, for increasing image sizes the bi-cubic interpolation will be more accurate due to the higher order. It is still good to check the performance though. Both in accuracy and speed.

The MeqUVInterpol node parses the Request on.

The node expects a Result (a UVBrick) from its child that has at least u (d_x) and v (d_y) axes. This Result is interpolated onto a frequency / time grid. The resulting Result is send to its parent.

Note: Sampled data vs. Integrated data. Right now the MeqUVInterpol node generates sampled data. That is, for every frequency / time cell the center value is used to obtain an interpolation value. Is there a need for integrated data? I.e., do we have to take more samples within the frequency / time cell and integrate the result? This has performace implications, since it means multiple interpolations per cell! Integration might be pretty expensive in this way, especially if convergence is needed.

A Implementation

(The contents of this appendix will eventually be copied into the coreesponding sections. For now this is treated as personal implementation notes.)

For the Patch Predict four new nodes, MeqPatchComposer, MeqStokes, MeqFFTBrick, and MeqUVInterpol, are to be implemented. The MeqStokes node is a fairly straightforward node. The MeqUVInterpol is available from the initial implementation, altough it should be extended to multiple frequency planes and interpolation using the UVBrick in meters. Most work is involved in the implementation of the MeqPatchComposer node and the MeqFFTBrick node. These two nodes are also very much related: choices in implementation in the one node directly affect the implementation of the other.

MeqPatchComposer
The MeqPatchComposer returns a grid containing a Patch Image of several Point Sources. In a sense it returns a sixpack that is composed of other sixpacks that are Phase Shifted and Regridded.

The grid that is returned is in frequency dependent l' / m' coordinates. The extent of the grid is determined by the source that is farthest away from the Patch Phase Center:

$$\begin{aligned} l'_{max} &= f_{max} l_{max}/c \\ m'_{max} &= f_{max} m_{max}/c \end{aligned}$$

The grid spacing is determined by the longest baseline (in meters)

$$dl' = dm' = \frac{1}{2d_{max}}, \qquad d_{max} = \max\left\{\sqrt{d_x^2 + d_y^2 + d_z^2}\right\}.$$

For the WSRT $d_{max} = 2700 meters$. The number of grid points is integer and rounded off to above such that the grid encompasses $[-l'_{max}, +l'_{max}] \times [-m'_{max}, +m'_{max}]$, i.e.

$$nl = 2 * int(l'_{max}/dl') + 1, \qquad nm = 2 * int(m'_{max}/dm') + 1$$

Furthermore, the grid is constructed in such a way that Image values are given at cell centers. Hence, the Phase Center or origin if the grid is situated in the center of the grid cell

$$[-\frac{dl'}{2}, +\frac{dl'}{2}] \times [-\frac{dm'}{2}, +\frac{dm'}{2}]$$

This means that if

$$l' = f * l/c, \qquad m' = f * m/c$$

then the grid position is given by

$$\begin{aligned}
l' > 0 : i &= (nl-1)/2 + 1 + int(l'/dl' + 0.5) - 1 \\
l' < 0 : i &= (nl-1)/2 + 1 + int(l'/dl' - 0.5) - 1 \\
m' > 0 : j &= (nm-1)/2 + 1 + int(m'/dm + 0.5) - 1 \\
m' < 0 : j &= (nm-1)/2 + 1 + int(m'/dm - 0.5) - 1
\end{aligned}$$

When the to be returned grid is determined, the sixpacks should be regridded onto it. The question is 'What is the best way to regrid making sure that the total flux is conserved?' We have to look more carefully into this subject. Another question: do the units affect this choice (Jy vs. Jy / beam)?

We see a number of possibilities for the regridding:

- Nearest Neighbour
- (Linear) Interpolation
- something based on inverse Linear Interpolation?
- ...

We will start with implementing a Nearest Neighbour solution.

MeqFFTBrick
The MeqFFTBrick returns the Fourier transform of the recieved Patch Image. Since we first apply the MeqStokes node we Fourier transform complex Sky Brightnesses. This means that we need a 2D complex-to-complex FFT.

Consider

- The Patch Image grid has an odd number of grid points the way it is defined now.
- The Patch Image grid should be padded with zero's in order to obtain the correct number of Points per Wavelength (PPW) for the MeqInterpol node. 10 PPW corresponds with a factor of ten zero's to be added.

©ASTRON 2005

- The final number of grid points should be a power of two.
- The gris should be transformed in such a way that it is optimal for the FFT

In the end we might want to implement our own FFT library. In order to get a first version running soon, we will use the AIPS++ FFT library. This is somewhat inefficient, since the Cells / Vells should first be transformed into an AIPS++ Image (imagefromarray) and after the FFT be transformed back into a Cells / Vells. However, this way has the advantage that the AIPS++ FFT server has a docmented input / output interface which makes it less likely to make mistakes.

References

[1] J.E. Noordam, R.J. Nijboer, *Functional description of the Local Sky Model*, LOFAR document (in progress) 2005.

[2] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes, in C or Fortran, The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1986 - 1992.

[3] G.B. Taylor, C.L. Carilli, R.A. Perley, *Synthesis Imaging in Radio Astronomy II*, Astronomical Society of the Pacific Conference Series Volume 180, 1999.

[4] A.R. Thompson, J.M. Moran, G.W. Swenson, Jr., *Interferometry and Synthesis in Radio Astronomy*, 2nd edition, John Wiley & Sons, Inc., New York, 2001.