Assorted DiffuserCam Derivations

Shreyas Parthasarathy and Camille Biscarrat
Advisors: Nick Antipa, Grace Kuo, Laura Waller

Last Modified November 27, 2018

1 Walk-through ADMM Update Solution

In our derivation of the ADMM algorithm, we stopped after showing how the variable splitting gives rise to 4 primal update steps and 3 dual update steps. The 4 primal updates are each a standalone optimization problem. We solve them using typical convex optimization techniques here as the final step in the mathematical formulation of the ADMM algorithm for DiffuserCam. For detailed *implementation* details please see our Jupyter notebook guide on the ADMM algorithm.

To begin, we repeat the primal updates:

$$
\begin{aligned}
u_{k+1} &\leftarrow \underset{u}{\text{argmin}} \ \tau\|u\|_1 + \frac{\mu_2}{2}\|\Psi\mathbf{v}_k - u\|_2^2 + \eta_k^\intercal(\Psi\mathbf{v}_k - u) \\
x_{k+1} &\leftarrow \underset{x}{\text{argmin}} \ \frac{1}{2}\|\mathbf{C}x - \mathbf{b}\|_2^2 + \frac{\mu_1}{2}\|\mathbf{M}\mathbf{v}_k - x\|_2^2 + \xi_k^\intercal(\mathbf{M}\mathbf{v}_k - x) \\
w_{k+1} &\leftarrow \underset{w}{\text{argmin}} \ \frac{\mu_3}{2}\|\mathbf{v}_{k+1} - w\|_2^2 + \rho_k^\intercal(\mathbf{v}_{k+1} - w) + \mathbb{1}_+(w) \\
\mathbf{v}_{k+1} &\leftarrow \underset{\mathbf{v}}{\text{argmin}} \ \frac{\mu_1}{2}\|\mathbf{M}\mathbf{v} - x_{k+1}\|_2^2 + \frac{\mu_2}{2}\|\Psi\mathbf{v} - u_{k+1}\|_2^2 + \frac{\mu_3}{2}\|x_{k+1} - w_{k+1}\|_2^2
\end{aligned}
$$

1.1 u-update

All the relevant Lagrangian terms in the u-update are differentiable except that of the ℓ_1 norm. Luckily, since that term is still convex, we can use the properties of sub-gradients to minimize the Lagrangian with respect to u is a way that is very similar to the usual "take the derivative and set it to zero" method. Instead of setting the non-existent derivative equal to zero, we take the sub-gradient (which is a set) and require that 0 be in that set whenever $u = u^*$, the optimal point:

$$
\mathbf{0} \in \partial \left\{ \tau\|u\|_1 + \frac{\mu_2}{2}\|\Psi\mathbf{v}_k - u\|_2^2 + \eta_k^\intercal(\Psi\mathbf{v}_k - u) \right\} \Big|_{u=u^*}
$$

One can show that the sub-gradient of differentiable functions is a set with just the actual derivative in it, so our condition is really:

$$
\mathbf{0} \in \mu_2(u - \Psi\mathbf{v}_k) - \eta_k + \tau\left(\partial\|u\|_1\right) \Big|_{u=u^*}
$$

Now, if this vector relation holds true, it must hold true for every component. This view is especially useful because the ℓ_1 norm is "separable," meaning that we can actually separate the vector relation into an exactly equivalent set of real-valued relations based on the components:

$$
0 \in \mu_2[u - (\Psi\mathbf{v}_k) + \eta_k]_i + \tau\left(\partial|u_i|\right) \Big|_{u_i=u_i^*}
$$

The derivative of $|u_i|$ exists and is whenever $u_i \neq 0$. It is 1 when $u_i > 0$, -1 when $u_i < 0$. When $u_i = 0$, the subgradient is everything in the closed interval $[-1, 1]$. So, we can split the solution for u_i^* into two cases:

$$\begin{cases} 0 = \mu_2[u^* - (\Psi \mathbf{v}_k) - \eta_k]_i + \tau \text{sign}(u_i^*) & u_i^* \neq 0 \\ 0 \in \mu_2[u^* - (\Psi \mathbf{v}_k) - \eta_k]_i + \tau[-1, 1] & u_i^* = 0, \end{cases}$$

Recall that we are trying to solve for u_i^*, so it's not useful to have the solution be split into cases that depend on what that value is. So, let's see if we can draw conclusions by calculating the optimal point in each case. Suppose our solution $u_i^* > 0$. Then, we fall under the first case and have:

$$\begin{aligned} &\mu_2 u_i^* - \mu_2(\Psi \mathbf{v}_k + \eta_k)_i + \tau = 0 \\ &\implies u_i^* = (\Psi \mathbf{v}_k + \eta_k)_i - \tau/\mu_2 \end{aligned}$$

Let's define $s := \Psi \mathbf{v}_k + \eta_k$. What we've shown is that if $u_i^* > 0$, then $u_i^* = s_i - \tau/\mu_2$. Similarly, if $u_i^* < 0$, we find $u_i^* = s_i + \tau/\mu_2$. Moreover, the condition on u_i^* can now be converted to a condition on s_i:

$$u_i^* = \begin{cases} s_i - \tau/\mu_2 & s_i > \tau/\mu_2 \\ s_i + \tau/\mu_2 & s_i < -\tau/\mu_2 \end{cases}$$

Lastly, if $u_i^* = 0$, the second case tells us

$$\begin{aligned} &0 \in \mu_2 u_i^* - \mu_2(\Psi \mathbf{v}_k + \eta_k)_i + [-\tau, \tau] \\ &0 \in -\mu_2(\Psi \mathbf{v}_k + \eta_k)_i + [-\tau, \tau] \\ &0 \in [s_i - \tau/\mu_2, s_i + \tau/\mu_2] \\ &\implies |s_i| < \tau/\mu_2 \end{aligned}$$

In summary:

$$u_i^* = \begin{cases} s_i - \tau/\mu_2 & s_i > \tau/\mu_2 \\ s_i + \tau/\mu_2 & s_i < -\tau/\mu_2 \\ 0 & |s_i| < \tau/\mu_2 \end{cases}$$

These equations are the definition of soft-thresholding, compactly written as $\boxed{u_i^* = \text{sign}(s_i) \max(0, |s_i| - \tau/\mu_2)}$. In non-component form: $\boxed{u^* = \mathcal{T}_{\frac{\tau}{\mu_2}}(\Psi \mathbf{v}_k + \eta_k)}$

1.2 x-update

This update is simpler because all the terms are differentiable. The derivative of the expression that we are minimizing with respect to x is:

$$\mathbf{C}^H(\mathbf{C}x - \mathbf{b}) - \mu_1(\mathbf{M}\mathbf{v}_l - x) - \xi$$

x_{k+1} is the point that makes this expression 0:

$$\begin{aligned} \mathbf{C}^H(\mathbf{C}x_{k+1} - \mathbf{b}) - \mu_1(\mathbf{M}\mathbf{v} - x_{k+1}) - \xi_k &= 0 \\ \mathbf{C}^H\mathbf{C}x_{k+1} + \mu_1 x_{k+1} - \mathbf{C}^H\mathbf{b} - \mu_1\mathbf{M}\mathbf{v} - \xi_k &= 0 \\ \boxed{x_{k+1} = (\mathbf{C}^H\mathbf{C} + \mu_1 I)^{-1}(\mathbf{C}^H\mathbf{b} + \mu_1\mathbf{M}\mathbf{v}_k + \xi_k)} \end{aligned}$$

1.3 w-update

All the terms are differentiable except the indicator function $\mathbb{1}_+(w)$. Rather than use subgradients again, we observe that the optimal point must have $w_{k+1} \geq 0$. If not, then the objective function would be infinite. So, either $w_{k+1} > 0$ and it minimizes the quantity without the indicator function or $w = 0$. In other words:

$$w_{k+1} = \max\left(\underset{w}{\text{argmin}}\left[\frac{\mu_3}{2}\|\mathbf{v}_{k+1} - w\|_2^2 + \rho_k^\intercal(\mathbf{v}_{k+1} - w)\right], 0\right)$$

To solve the minimization problem, we take the derivative and set it to zero:

$$\begin{aligned} -\mu_3(\mathbf{v}_{k+1} - w_{k+1}) - \rho_k &= 0 \\ \mu_3 w_{k+1} &= \rho_k + \mu_3 \mathbf{v}_{k+1} \\ w_{k+1} &= \frac{\rho_k}{\mu_3} + \mathbf{v}_{k+1} \end{aligned}$$

In summary:

$$\boxed{w_{k+1} = \max\left(\rho_k/\mu_3 + \mathbf{v}_{k+1}, 0\right)}$$

2 Adjoint Operators (Derivations of Ψ^H and $\mathbf{C}^H$)

For readers with enough mathematical background, they can skip to sections 2.3 and 2.4 for a straightforward derivation. The following develops that background.

In DiffuserCam, all of the objects we deal with can be considered as either elements of a particular vector space or operators acting on those spaces. There is a natural "inner product" that we can associate with all of these vector spaces. For example, an inner product $\langle u, v\rangle$ where u and v are 1-dimensional vectors is usually defined as the dot product: $u \cdot v = u^\intercal v = \sum u_i v_i$ where the sum is over all elements of the vectors. The definition of an adjoint operator depends on these inner products. Let L be a linear operator mapping vectors from space X to space Y. Then L^H is the operator mapping vectors from space Y to space X such that $\langle y, Lx\rangle = \langle L^H y, x\rangle$. Enforcing this property is the most general way to derive the adjoint of the operators we encounter in the ADMM algorithm.

2.1 Continuous version (integration by parts)

Suppose we were looking at the space of integrable functions on $[0, 1]$. The typical inner product used is $\langle u, v\rangle = \int_0^1 u(x)v(x)\mathrm{d}x$. Now consider the derivative operator D. We want to find D^H such that

$$\begin{aligned} \langle D^H u, v\rangle &= \langle u, Dv\rangle \\ &= \int_0^1 u(x) D[v(x)]\mathrm{d}x \\ &= \int_0^1 u(x)v'(x)\mathrm{d}x \\ &= u(x)v(x)\Big|_0^1 - \int_0^1 u'(x)v(x)\mathrm{d}x \end{aligned}$$

If we introduce a boundary condition that $u(0) = u(1) = v(0) = v(1) = 0$ (stronger than needed), then we can conclude that $\langle D^H u, v\rangle = \int_0^1 (-u'(x))\, v(x)\mathrm{d}x$, meaning that $D^H = -D$. This example gives us hints on how to do the discrete case – write out the definition, "transfer" the operator over to the other vector (summation by parts), and introduce a boundary condition.

2.2 1D Forward Difference

Now, instead of a derivative operator, suppose we want to find the adjoint of a first order *finite difference* operator. For example, let's use the **forward difference**:

$$Dv_i \equiv v_{i+1} - v_i$$

Given a vector $v \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$, we define the adjoint $D^\dagger$ such that $\langle D^\dagger u, v\rangle = \langle u, Dv\rangle$, where

$$\begin{aligned}
\langle u, Dv\rangle &\equiv \sum_{i=1}^{n} u_i Dv_i \\
&= \sum_{i=1}^{n} u_i (v_{i+1} - v_i) \\
&= \sum_{i=1}^{n} u_i v_{i+1} - \sum_{i=1}^{n} u_i v_i \qquad (1) \\
&= \sum_{i=2}^{n+1} u_{i-1} v_i - \sum_{i=1}^{n} u_i v_i \\
&= u_n v_{n+1} - u_1 v_1 + \sum_{i=2}^{n} v_i (u_{i-1} - u_i) \qquad (2)
\end{aligned}$$

We're aiming to rewrite the expression above as an inner product, but we have two issues in Eq 2:

1. Missing an $i = 1$ term
2. Two extra terms

We can solve both of these issues by enforcing a **boundary condition**. First let's just set $u_n v_{n+1} - u_1 v_1 \equiv v_1(u_0 - u_1)$, since this solves both issues:

$$u_n v_{n+1} - u_1 v_1 + \sum_{i=2}^{n} v_i (u_{i-1} - u_i) = \boxed{\sum_{i=1}^{n} v_i (u_{i-1} - u_i) \equiv \langle D^\dagger u, v\rangle} \qquad (3)$$

The above indicates that $D^\dagger u \equiv u_{i-1} - u_i$, which is the *negative* of the **backward difference**

So what boundary condition did our assumption enforce? It is a weaker form of the circular boundary condition CBC, which can be readily satisfied by padding the image beforehand:

$$\begin{cases} u_0 = u_n \\ u_1 = u_{n+1} \\ \quad \vdots \end{cases} \qquad \text{similarly for } v$$

Subbing these into the terms gives us $u_n v_{n+1} - u_1 v_1 = u_0 v_1 - u_1 v_1 = v_1(u_0 - u_1)$ as expected. In the more complicated cases below, it turns out the appropriate boundary condition is also CBC, so we will just invoke it when necessary without explicitly showing it to be the case.

Lastly, note that for 1D vectors, we can represent D as a matrix, where CBC determines the last row:

$$\mathbf{D} = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \dots & -1 \end{bmatrix}$$

Once we write down that matrix, the adjoint can easily be found as the conjugate transpose of $\mathbf{D}$:

$$\mathbf{D}^\dagger = \begin{bmatrix} -1 & 0 & 0 & \dots & 1 \\ 1 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -1 \end{bmatrix},$$

which is readily seen as the negative backward difference. However, for 2D and more complicated inner products, the sum formulas above will prove more useful.

2.3 2D Forward Difference (Ψ^H)

In this case, note that the discrete gradient $D : \mathbb{R} \to \mathbb{R}^2$, so $D^\dagger : \mathbb{R}^2 \to \mathbb{R}$. Therefore, let

$$Dv_{ij} \equiv \begin{bmatrix} v_{i+1,j} - v_{i,j} \\ v_{i,j+1} - v_{i,j} \end{bmatrix},$$

the discrete 2D forward difference. Define

$$U_{ij} \equiv \begin{bmatrix} U^x \\ U^y \end{bmatrix}$$

. Let's also assume that v is an $n \times m$ image. Then the inner product is:

$$\begin{aligned} \langle U, Dv \rangle &\equiv \sum_{i=1}^{n} \sum_{j=1}^{m} U_{ij} \bullet Dv_{ij} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \left(U^x_{ij}(v_{i+1,j} - v_{i,j}) + U^y_{ij}(v_{i,j+1} - v_{i,j}) \right) \\ &= \sum_{j=1}^{m} \left[\sum_{i=1}^{n} U^x_i (v_{i+1} - v_i) \right]_j + \sum_{i=1}^{n} \left[\sum_{j=1}^{m} U^y_j (v_{j+1} - v_j) \right]_i \end{aligned}$$

The common indices i and j have been removed and applied outside the sum term to emphasize that each bracketed term is the same form as Eq 1. Pattern matching yields

$$\langle U, Dv \rangle = \sum_{j=1}^{m} \left[U^x_n v_{n+1} - U^x_1 v_1 + \sum_{i=2}^{n} v_i (U^x_{i-1} - U^x_i) \right]_j + \sum_{i=1}^{n} [\dots]_i$$

The 1D circular boundary condition in Eq. 3 can be applied to each row and column of the image:

$$\begin{aligned} \langle U, Dv \rangle &= \sum_{j=1}^{m} \sum_{i=1}^{n} \left[v_i (U^y_{i-1} - U^y_i) \right]_j + \left[v_j (U^x_{j-1} - U^x_j) \right]_i \\ &= \boxed{\sum_{i=1}^{n} \sum_{j=1}^{m} v_{ij} \underbrace{\left[(U_{i-1,j} - U_{ij})_x + (U_{i,j-1} - U_{ij})_y \right]}_{D^\dagger U_{ij}}} \end{aligned}$$

2.4 2D Crop ($\mathbf{C}^H$)

Finding the adjoint of the 2D crop operator is also straightforward using this definition. Let $\mathbf{C} : \mathbb{R}^{n \times m} \to \mathbb{R}^{n' \times m'}$, with $n' < n$ and $m' < m$, denote the operator that takes an $n \times m$ image and crops it to size $n' \times m'$

(for notation simplicity, assume the crop takes the top left part of the image). Suppose we have an arbitrary $n \times m$ image v and $n' \times m'$ image u. We want $\mathbf{C}^H$ such that the two inner products below are the same:

$$\langle u', \mathbf{C}v \rangle \equiv \sum_{i=1}^{n'} \sum_{j=1}^{m'} u'_{ij} \, [\mathbf{C}v]_{ij}$$

$$\langle \mathbf{C}^H u', v \rangle \equiv \sum_{i=1}^{n} \sum_{j=1}^{m} [\mathbf{C}^H u']_{ij} \, v_{ij}$$

But the last inner product is a sum over nm pixels and the first one is a sum over the $n'm'$ cropped pixels. The only way these sums can always be the same is if all the extra terms in the second product are zero. So, $\mathbf{C}^H$ turns u' into a size $n \times m$ image, but with every new pixel set to zero – this is the zero-pad operator!

3 Calculating Large Matrix inverses

While we have largely avoided dealing with the actual matrices used to represent operators such as cropping, convolution, and padding, here we consider what they look like, because inverting the operators efficiently will implicitly use properties of the matrix representation.

Let's go through the process with a linear function (or operator) O that operates on images. Let $\mathbf{O}$ be the matrix corresponding to that operator. Recall from (writeup) that if O operates on $m \times n$ images, then $\mathbf{O}$ is an $mn \times mn$-dimensional matrix, operating on vectorized images of length mn.

Now consider the image $y = O(x)$. In matrix vector form, we can write this as $\mathbf{y} = \mathbf{O}\mathbf{x}$. This equation tells us explicitly that every pixel $\mathbf{y}_j$ is a linear combination of the pixels in $\mathbf{x}$: $\mathbf{y}_j = \sum_i \mathbf{O}_{ij} x_i$.

So, if $\mathbf{O}$ is diagonal, then we find $\mathbf{y}_j = \mathbf{O}_{jj} x_j$. In other words, a diagonal operator O *scales the jth pixel by factor* $\mathbf{O}_{jj}$! Importantly, the inverse of a diagonal matrix is also diagonal, with all the entries inverted. So, the inverse operator O^{-1} also scales each pixel individually, now by factor $\dfrac{1}{\mathbf{O}_{jj}}$ (remember that j is an index we give to the pixel based on the vectorized image). The main idea is that we can multiply each pixel by the appropriate weight without regard to other pixels in the image, which can easily be done in parallel.

In general, large matrix inverses are the main bottleneck in an algorithm; we need to be careful to make sure that all inverses involved in our algorithm are easily calculated via some method that doesn't require instantiating the matrix. In the following case, we are lucky because *both* $\mathbf{C}^H\mathbf{C}$ and $\mu_1 I$ are diagonalizable *in the same standard basis*, so multiplying by its inverse is the same as dividing by the summed entries. Our only option later will be to find a different basis where all the terms are diagonalizable.

3.1 $(\mathbf{C}^H\mathbf{C} + \mu_1 I)^{-1}$

$\mathbf{C}^H\mathbf{C}$ is a crop down to an image of size `sensor_size` and then a zero-pad back up to `full_size`. In other words, it zeros out any pixels outside of the cropped region, and leaves everything inside the cropped region untouched. We know this corresponds to a diagonal matrix because it is pointwise multiplication of each pixel by either a 0 or a 1. So, $\mathbf{O}$ corresponds to a pointwise multiplication of every pixel by either $0 + \mu_1$ or $1 + \mu_1$. The inverse of this operator is thus pointwise multiplication of every pixel by either $\dfrac{1}{\mu_1}$ or $\dfrac{1}{1 + \mu_1}$.

Now, we will see an example of inversion where the operator is diagonalizable in a different, non-standard basis (not pixel-wise) – the basic idea will remain the same.

3.2 $(\mu_1 \mathbf{M}^H \mathbf{M} + \mu_2 \Psi^H \Psi + \mu_3 I)^{-1}$

If we try to calculate the matrix entries for acting $\mu_1 \mathbf{M}^H \mathbf{M}$ on a vectorized image like before, we find that the product is not diagonal, because convolutions can depend on many different pixels at once by definition.

In fact, the matrix corresponding to $\mathbf{M}^H \mathbf{M}$ is not diagonal in the standard basis precisely because it is diagonal in a different basis! As we showed in the Gradient Descent notebook:

$$\begin{aligned} \mathbf{M}^H \mathbf{M} \mathbf{v} &= \mathbf{F}^{-1} \text{diag}(\mathbf{F}\mathbf{h})^* \ \text{diag}(\mathbf{F}\mathbf{h}) \ \mathbf{F}\mathbf{v} \\ &= \mathcal{F}^{-1} \left\{ \mid \mathcal{F}h \mid^2 \cdot (\mathcal{F}v) \right\}, \end{aligned}$$

we find that it performs pixel-wise multiplication in the *Fourier* (frequency) space. In other words, $\mathbf{M}^H \mathbf{M}$ is diagonal if we take a Fourier transform first. If we wanted to efficiently calculate the inverse $(\mu_1 \mathbf{M}^H \mathbf{M})^{-1}$, we would do a DFT to switch to frequency space, do division by the values of $\mid \mathcal{F}h \mid^2$ in that space, and then inverse DFT to get back to standard (pixel) basis. However, the inverse we wish to calculate has other terms as well, so we can only use this method if $\Psi^H \Psi$ and I are also diagonalizable by a Fourier transform. Since $\mathcal{F}^{-1} I \mathcal{F} = I$, the identity is diagonalizable.

We knew $\mathbf{M}^H \mathbf{M}$ was diagonalizable by a Fourier Transform because it was a convolution. So, to show $\Psi^H \Psi$ is also diagonalizable by a Fourier transform we must write $\Psi^H \Psi \mathbf{v} = \Psi^H (\Psi \mathbf{v})$ as a convolution between $\mathbf{v}$ and a fixed kernel. To that end, in 2D the forward difference acted on the i, jth pixel looks like:

$$\begin{aligned} \Psi v_{ij} &= \begin{bmatrix} v_{i+1,j} - v_{i,j} \\ v_{i,j+1} - v_{i,j} \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{i-1,j} & v_{i,j} & v_{i+1,j} \end{bmatrix} \\ \\ \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} v_{i,j-1} \\ v_{i,j} \\ v_{i,j+1} \end{bmatrix} \end{bmatrix} \end{aligned}$$

The above formula holds for every pixel in the image (assuming the same circular boundary conditions that are used to derive Ψ^H itself). This formula is also the pointwise definition of cross-correlation with the *non-vectorized* image v:

$$\Psi \mathbf{v} = \begin{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \star v \\ \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \star v \end{bmatrix},$$

where we have written both the row-wise and column-wise pixel differences in terms of a cross-correlation, just with two different "kernels."

Let's call the row (top) kernel k_{RF} and the column (bottom) kernel k_{CF}, where the "F" stands for forward difference.

Using exactly the same process with Ψ^H (and the stack of two images u defined as above):

$$\Psi^H u = -k_{RB} \star u^x - k_{CB} \star u^y,$$

where the "B" stands for backwards difference.

So,

$$\begin{aligned}\Psi^H\Psi\mathbf{v} &= \Psi^H \begin{bmatrix} k_{RF} \star v \\ k_{CF} \star v \end{bmatrix} \\ &= -k_{RB} \star (k_{RF} \star v) - k_{CB} \star (k_{CF} \star v)\end{aligned}$$

At this point, we cannot simplify the operations because cross-correlations are not associative – we need to reformulate the expression in terms of *convolutions*. So, we use the property that cross-correlation with a kernel k is equivalent to convolution with the *flipped* kernel k' where we flip horizontally *and* vertically. For example, if $k = \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$, then $k' = \begin{bmatrix} 1 & -1 & 0 \end{bmatrix}$. If $k = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$ then $k' = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$. Flipping the kernels and using associativity of convolution, we find:

$$\begin{aligned}\Psi^H\Psi\mathbf{v} &= -k_{RB} \star (k_{RF} \star v) - k_{CB} \star (k_{CF} \star v) \\ &= -k'_{RB} * (k'_{RF} * v) - k'_{CB} * (k'_{CF} * v) \\ &= -\left[(k'_{RB} * k'_{RF}) + (k'_{CB} * k'_{CF})\right] * v\end{aligned}$$

Finally, we again use the convolution theorem: for any k, v, $k * v = \mathcal{F}^{-1}\{\mathcal{F}(k) \cdot \mathcal{F}(v)\}$. Thus, setting $k = -(k'_{RB} * k'_{RF}) - (k'_{CB} * k'_{CF})$, we have that

$$\Psi^H\Psi\mathbf{v} = \mathcal{F}^{-1}\{\mathcal{F}(k) \cdot \mathcal{F}(v)\},$$

which corresponds to pixel-wise multiplication by $\mathcal{F}(k)$ in the Fourier space.

To summarize, in matrix form, the inverse term can be written as

$$\begin{aligned}(\mu_1\mathbf{M}^H\mathbf{M} + \mu_2\Psi^H\Psi + \mu_3 I)^{-1} &= (\mu_1\mathbf{F}^{-1}D_M\mathbf{F} + \mu_2\mathbf{F}^{-1}D_\Psi\mathbf{F} + \mu_3\mathbf{F}^{-1}\mathbf{F})^{-1} \\ &= \mathbf{F}^{-1}(\mu_1 D_M + \mu_2 D_\Psi + \mu_3)^{-1}\mathbf{F},\end{aligned}$$

where D_M and D_Ψ are the diagonal operators corresponding to $\mathbf{M}^H\mathbf{M}$ and $\Psi^H\Psi$ implemented using pixel-wise multiplication in the Fourier space:

$$\begin{cases} D_M = \mathrm{diag}(|\mathcal{F}(h)|^2) \\ D_\Psi = \mathrm{diag}(\mathcal{F}(\texttt{pad}(k))), \end{cases}$$

where the `pad` is used just to make the kernel the same size as the image.

What is that kernel?

$$\begin{aligned}k &= -[(k'_{RB} * k'_{RF}) + (k'_{CB} * k'_{CF})] \\ &= -\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right) - \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right) \\ &= -\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\end{aligned}$$