1. INTRODUCTION

Given mat5 size as \( N_x \times M_y \) and kernel size as \( N_{xk} \times M_{yk} \) where the center pixel is located at \( \{m, n\} \) in mat5 space and \( m_k = (M_{yk} - 1)/2, n_k = (N_{xk} - 1)/2 \) in the kernel space. For simplicity, we will lexicographically index the kernel values and assume any that are not indicated by the mat5 quality image with \( n = 0, 1, \ldots, (N-1) \). In a second version, the influence of each world coordinate referenced by the kernel is weighted by a Gaussian function based on the Euclidean distance between the kernel center point and each of the kernel point world coordinates. We then fit a plane to the kernel points based on a weighted least squares estimate.

2. KERNEL SCANNING TEMPLATE

The kernel process involves moving a kernel across the input Mat5 format 3-D image. The mechanics of this algorithm is given as well as variable definitions.

Function input arguments are

- void MLSF(unsigned char *I, float *X, float *Y, float *Z, short \( N_x \), short \( M_y \) \n// mat 5 data
- short *KernelMask, short \( N_{xk} \), short \( M_{yk} \) \n// Kernel mask and size of kernel
- float \( r_c \) \n// radius of spatial weighting function
- short ithresh) \n// threshold level for quality matrix I

A template for the kernel process is as follows:
{
    long m, n, i, j, ioff, joff, in, jm, indexKernel;
    long index, indexout, Nkernel, N2;
    Point *kernel;
    Nkernel=(long)Nxk*(long)Myk;
    Kernel=new Point [Nkernel];
    //
    ioff=(Nxk-1)/2;
    joff=(Myk-1)/2;
    for(m=0;m<My;m++)
        for(n=0;n<Nx;n++)
        {
            indexout=(long)m*(long)Nx+n; // point to the output pixel location
            N2=0; // dynamic size of kernel data
            // map points into the kernel
            for(j = -joff; j <= joff; j++)
                for(i = -ioff; i <= ioff; i++)
                {
                    indexKernel=(j+joff)*Nxk+(i+ioff);
                    in = i + n; jm = j + m;
                    if((hm>=0)&&(hm<My)&&(in>=0)&&(in<Nx)) // kernel point inside image space
                        {
                            index=(long)jm*(long)Nx+in; // point to the kernel data
                            if((index>=0)&&(index<in)&&(in<Nx)) // valid pixel
                                {
                                    kernel[N2].x=X[index];
                                    kernel[N2].y=Y[index];
                                    kernel[N2].z=Z[index];
                                    ++N2
                                }
                        }
                } // j,i
            } // m,n
    delete [Nkernel] kernel;
}

3. PLANE FITTING
We will fit a plane to the kernel data by two methods and determine the resulting plane coefficients. In the first method, we weight all points in the kernel equally. In the second method we fit the plane with a de-weighting of point based on distance from the point under analysis. The equation of the plane is

\[ ax + by + cz + d = 0 \] (1a)
The world coordinates of the point under analysis is \( \{x_0, y_0, z_0\} \) and we will address the kernel data in sequence \( n=0, 1, 2 \ldots (N_k-1) \). We let \( c=1 \) and rearrange Eq. (1) to solve for \( a \), \( b \) and \( d \) using a linear algebraic method. The rearranged equation is

\[
ax + by + d = -z \tag{1b}
\]

We define a \((N_k-1)\times3\) coordinate matrix and a \((N_k-1)\times1\) \( p_z \) vector as

\[
\begin{bmatrix}
x_{w,0} & y_{w,0} & 1 \\
x_{w,1} & y_{w,1} & 1 \\
\vdots & \vdots & \vdots \\
x_{w,N_k-1} & y_{w,N_k-1} & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
-z_{w,0} \\
-z_{w,1} \\
\vdots \\
-z_{w,N_k-1}
\end{bmatrix}
\]

The 3x1 plane coefficient vector is

\[
a = \begin{bmatrix} a \\ b \\ d \end{bmatrix} \tag{3}
\]

Method 1:
We form a \((N_k-1)\times1\) vector with element values of \( z_{w,n} \) as \( p_z \). For method 1, the linear algebraic equation for the plane is then

\[
P_{xy} a = p_z \tag{4}
\]

A least squares solution for eq. (4) is then

\[
a = R_{pp}^{-1} P_{xy} p_z \tag{5}
\]

where \( R_{pp} = P_{xy} P_{xy}^T \).

Method 2:
In method 2, we de-weight the contributions in the least square solution based on distance from the analysis point. The weighting scheme is based on a Gaussian function as the coefficient such that

\[
w_n = \exp \left( -\frac{r_n^2}{2\sigma^2} \right) \tag{6}
\]

where \( r_n^2 = (x_{w,n} - x_0)^2 + (y_{w,n} - y_0)^2 + (z_{w,n} - z_0)^2 \).

We define a cutoff distance as \( r_c \) such that the de-weighting drops from a maximum of 1 to a value of \( \delta_c \). The de-weighting coefficient has a single parameter \( \sigma^2 \). The relationship for the cutoff value is

\[
\delta_c = \exp \left( -\frac{r_c^2}{2\sigma^2} \right) \tag{7}
\]
The solution for Eq. (7) is
\[ \sigma^2 = -\frac{r^2}{\ln(\delta_c)} \]  
(8)

Let the diagonal weighting matrix be
\[ W = \text{Diag} \left[ w_0, w_1, \ldots, w_{N_x-1} \right] \]  
(9)

so the method 2 relationship is
\[ WP_{xy}a = WP_z \]  
(10)

and the parameter solution is
\[ a = R_{pWW}^{-1}P_{xy}^TR_{WWW}p_z \]  
(11)

where \( R_{pWW} = P_{xy}^TR_{WWW}P_{xy} \) and \( R_{WWW} = WW \).

4. FINDING THE PROJECTION POINT

We know the plane coefficient vector is also the normal vector to the plane. Knowing the normal vector, the analysis point coordinate and the equation of the plane, we determine the projection of the analysis point \( \{x_0, y_0, z_0\} \) onto the plane at point \( \{x_p, y_p, z_p\} \). We can write the relationship between the analysis point and the projection point by scaling the normal vector by \( e \) such that

\[ x_0 = ea + x_p, \]  
(12a)

\[ y_0 = eb + y_p, \]  
(12b)

and

\[ z_0 = ec + z_p. \]  
(12c)

where \( c=1 \). By combining the 3 equations in Eq. (12) with the plane equation in Eq. (1) we set up a linear algebraic relationship between the four unknowns \( \{x_p, y_p, z_p, e\} \) such that

\[ AP_p = p_0 \]  
(13)

where

\[ A = \begin{bmatrix} a & b & 1 & 0 \\ 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 1 \end{bmatrix}, \]  
(14)
\[
\begin{align*}
\mathbf{p}_p &= \begin{bmatrix}
    x_p \\
    y_p \\
    z_p \\
    e 
\end{bmatrix}, \\
\mathbf{p}_0 &= \begin{bmatrix}
    -d \\
    x_0 \\
    y_0 \\
    z_0 
\end{bmatrix}.
\end{align*}
\]

and

The solution for the projection point and scale factor is

\[
\mathbf{p}_p = A^{-1} \mathbf{p}_0
\]

5. ENCODING THE KERNEL RESPONSE VALUE

With all the parameters determined, we determine the output of the kernel process. The output is based on the distance between the analysis point and the projection point and its sign is based on whether analysis point is above or below the projection point along the Z dimension. The Euclidian distance weighted by the sign function is

\[
d_{0p} = \text{sgn}(z_0 - z_p) \sqrt{(x_0 - x_p)^2 + (y_0 - y_p)^2 + (z_0 - z_p)^2}
\]

APPENDIX A: MATLAB

```matlab
function [ C ] = 
MLSF2015(I,X,Y,Z,Nx,My,Kerneltype,Kxside,Kyside,ithresh,clipPos,clipNeg)
% input mat5 matrices 
% Nxk=1+2*Kxside; 
% Myk=1+2*Kyside; 
% Kerneltype:0=full rectangle,1=ellipse to fill Nxk x Myk 
% KernelMask=ones(Myk,Nxk); % kerneltype:0=default 
if Kerneltype==1 % ellipse 
    xc=1+(Nxk-1)/2;yc=1+(Myk-1)/2; 
    % x^2/a^2 + y^2/b^2 = 1 
    A=Nxk-xc;B=Myk-yc; 
    for mk=1:Myk 
        for nk=1:Nxk 
            r2=(nk-xc).^2/(A.^2)+(mk-yc).^2/(B.^2); 
            if r2>1 
                KernelMask(mk,nk)=0; 
            end; 
        end; % for nk 
    end; 
end; % for mk 
```
end; % for mk
end; % if K=1
C=zeros(My,Nx);
ioff=(Nxk-1)/2;
joff=(Myk-1)/2;
KernelX=zeros(Myk*Nxk,1);
KernelY=zeros(Myk*Nxk,1);
KernelZ=zeros(Myk*Nxk,1);
for m=1:My
    for n=1:Nx
        Nk=1;
        if I(m,n)>=ithresh
            x0=X(m,n);y0=Y(m,n);z0=Z(m,n);
            % fill kernel
            for j=-joff:joff
                for i=-ioff:ioff
                    in=i+n;jm=j+m;
                    mk=j+joff+1;
                    nk=i+ioff+1;
                    if jm>=1 & jm<=My
                        if in>=1 & in<=Nx
                            if ((I(jm,in)>=ithresh) | (j==0 & i==0)) &
                                KernelMask(mk,nk)>0
                                %
                                KernelX(Nk,1)=X(jm,in);
                                KernelY(Nk,1)=Y(jm,in);
                                KernelZ(Nk,1)=Z(jm,in);
                                if j==0 & i==0
                                    nk0=Nk;
                                end;
                                Nk=Nk+1;
                            end;
                        end; % if in>=1
                    end; % if jm>=1
                end; % for i
            end; % for j
            Nk=Nk-1;
        end; % if Nk>3
        if Nk>3
            % process kernel
            x0=KernelX(nk0,1);y0=KernelY(nk0,1);z0=KernelZ(nk0,1);
            % Method 1: estimate the plane parameters
            % load matrices to estimate plane parameters
            Pxy=zeros(Nk,3);
pz=zeros(Nk,1);
            for nk=1:Nk;
                % Pxy
                Pxy(nk,1)=KernelX(nk,1);
                Pxy(nk,2)=KernelY(nk,1);
                Pxy(nk,3)=1;
                % pz
                pz(nk)=-KernelZ(nk,1);
            end; % for nk
            %
            Rpp=Pxy'*Pxy;
            if rank(Rpp)==3
                % acoef=[ a, b, d]
                acoef=inv(Rpp)*Pxy'*pz;
            end;
        end;
    end; % for n
end; % for m
end;
a=acoef(1); b=acoef(2); c=1; d=acoef(3);
% Finding the Projection Point
A=zeros(4,4);
A(1,1)=a; A(1,2)=b; A(1,3)=c;
A(2,1)=1; A(2,4)=a;
A(3,2)=1; A(3,4)=b;
A(4,3)=1; A(4,4)=c;
if rank(A)==4
    p0=zeros(4,1);
p0(1)=-d;
p0(2)=x0;
p0(3)=y0;
p0(4)=z0;
    pp=inv(A)*p0;

% Response Value
d0p=(x0-pp(1)).^2+(y0-pp(2)).^2+(z0-pp(3)).^2;
d0p=sign(z0-pp(3))*sqrt(d0p);
C(m,n)=d0p;
end;
% rank(A)==4
end; % rank(Rpp)==3
if m==152 & n==164
    
end; % Nk>3
end; % I>=ithresh
end; % for n
end; % for m

% suppress outliers in C based on fractional area
C0=C;
C=ClipBipolar(C0,I,ithresh,clipPos,clipNeg,100);
%
end

ClipBipolar clips the positive and negative values of the C matrix of a mat5 image set.

function [ C2 ] = ClipBipolar(C1,I,ithresh,clipPos,clipNeg,Nhist)
% clipPos is 0<clipPos<1: at 1 no points are clipped
% and 0 all points are set to 0. At 0.5, sorted by value
% the maximum value of the lowest 50% of the points are
% set to that value
% clipNeg is 0<clipNeg<1: performs a similar operation to clipPos
% but only clips negative points.

[My,Nx]=size(C1)
% process positive values
Jpos=find(C1>=0 & I>=ithresh);
hpos=hist(C1(Jpos),Nhist);
minpos=min(min(C1(Jpos)));
maxpos=max(max(C1(Jpos)));
hnorm=sum(hpos);
hpos=hpos/hnorm;
% form accumulative value
for n=2:Nhist
    hpos(n)=hpos(n)+hpos(n-1);
end;
% find closest bin to clipPos
nclose=1;
egclose=abs(hpos(nclose)-clipPos);
for n=2:Nhist
    etemp=abs(hpos(n)-clipPos);
    if(etemp<=eclose)
        nclose=n;
        eclose=etemp;
    end;
end;
maxposclip=nclose*(maxpos-minpos)/Nhist;
Jclip=find(C1>maxposclip);
C1(Jclip)=maxposclip;

%% process negative values
if clipNeg>0
    Jneg=find(C1<0 & I>=ithresh);
    [Mj,Nj]=size(Jneg);
    if Mj*Nj>0
        hneg=hist(abs(C1(Jneg)),Nhist);
        minneg=min(min(abs(C1(Jneg))));
        maxneg=max(max(abs(C1(Jneg))));
        hnorg=sum(hneg);
        if hnorg>0
            hneg=hneg/hnorg;
        else
            hneg=0.*hneg;
        end; % if hnorg
    end; % if Mj*Nj>0
    % form accumulative value
    for n=2:Nhist
        hneg(n)=hneg(n)+hneg(n-1);
    end;
    % find closest bin to clipNeg
    nclose=1;
egclose=abs(hneg(nclose)-clipNeg);
    for n=2:Nhist
        etemp=abs(hneg(n)-clipNeg);
        if(etemp<=eclose)
            nclose=n;
            eclose=etemp;
        end;
    end;
maxnegclip=-nclose*(maxneg-minneg)/Nhist;
Jclip=find(C1>maxnegclip);
C1(Jclip)=maxnegclip;
end; % Check for negative values
end; % if clipNeg>0

%%
C2=C1;
end