MethodsX 7 (2020) 101023

Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

Semi-automated segmentation of the lateral periventricular regions using diffusion magnetic resonance imaging

Albert M. Isaacs[a,b,*], Rowland H. Han[c], Christopher D. Smyser[d,e,f], David D. Limbrick Jr.[c], Joshua S. Shimony[f]

[a] Department of Biology and Biomedical Sciences, Washington University in St. Louis, St. Louis, MO, USA
[b] Department of Clinical Neurosciences, University of Calgary, Calgary, Alberta, Canada
[c] Department of Neurosurgery, Washington University School of Medicine, St. Louis, MO, USA
[d] Department of Neurology, Washington University School of Medicine, St. Louis, MO, USA
[e] Department of Pediatrics, Washington University School of Medicine, St. Louis, MO, USA
[f] Mallinckrodt Institute of Radiology, Washington University School of Medicine, St. Louis, MO, USA

A B S T R A C T

The lateral ventricular perimeter (LVP) of the brain is a critical region because in addition to housing neural stem cells required for brain development, it facilitates cerebrospinal fluid (CSF) bulk flow and functions as a blood-CSF barrier to protect periventricular white matter (PVWM) and other adjacent regions from injurious toxins. LVP injury is common, particularly among preterm infants who sustain intraventricular hemorrhage or post hemorrhagic hydrocephalus and has been associated with poor neurological outcomes. Assessment of the LVP with diffusion MRI has been challenging, primarily due to issues with partial volume artifacts since the LVP region is in close proximity to CSF and other structures of varying signal intensities that may be inadvertently included in LVP segmentation.

This research method presents:

- A novel MATLAB-based method to segment a homogenous LVP layer using high spatial resolution parameters (voxel size 1.2 × 1.2 × 1.2 mm^3) to only capture the innermost layer of the LVP.
- The segmented LVP is averaged from three contiguous axial slices to increase signal to noise ratio and reduce the effect of any residual volume averaging effect and eliminates manual and inter/intrarater-related errors.

DOI of original article: 10.1016/j.nicl.2019.102031

* Corresponding author at: Department of Biology and Biomedical Sciences, Washington University School of Medicine 425 S Euclid Ave, St. Louis, MO 63110, USA
E-mail address: albert.isaacs@ucalgary.ca (A.M. Isaacs).

https://doi.org/10.1016/j.mex.2020.101023

ARTICLE INFO
Method name: Segmentation of lateral ventricular perimeter regions of interest
Keywords: Diffusion tensor imaging, Hydrocephalus, Intraventricular hemorrhage, Lateral ventricular perimeter, Preterm infant, Subventricular zone, Ventricular zone
Article history: Received 29 March 2020; Accepted 4 August 2020; Available online 20 August 2020

Specifications table

Subject Area	Neuroscience
Method name:	Segmentation of lateral ventricular perimeter regions of interest
More specific subject area	Image processing for brain diffusion MRI
Name and reference of original method	*N/A.*
Resource availability	1. MATLAB (https://www.mathworks.com/products/matlab.html) 2. The *tilefigs.m* program, previously described by Charles Plum [2] was used only to display images already created by the algorithm (https://www.mathworks.com/matlabcentral/fileexchange/328-tilefigs-m.

Method details

The semi-automated algorithm for selecting periventricular ROIs begins by reading standard processed DTI files into MATLAB. Images showing mean diffusivity (MD) are displayed sequentially to allow the user to select slices centered on the foramen of Monro. Once this slice is determined, the algorithm automatically selects three total axial slices (one rostral and one caudal to the center slice) for further analysis. The MD image is used since it provides a sharp demarcation between the brain parenchyma and the CSF.

The script then displays a histogram of the MD values present in the center slice, which appears as a bimodal distribution. The lower peak of this distribution represents MD values in the brain parenchyma, and the higher peak represents the cerebrospinal fluid spaces. Based on this histogram, the user determines a threshold value that best separates parenchyma and CSF MD values. Following this, the script produces an image showing voxels with MD values above the threshold. When properly executed, these steps result in a few predominant regions representing the lateral and third ventricles, along with some small peripheral regions of CSF. Through sequential erode and dilate steps, the selected regions are smoothed, and small extraneous spaces are removed.

Next, the script calculates the number of objects in each slice and the area of each object (blob analysis) and displays this information. A predetermined minimum size (we used 10 voxels) is applied to display only areas that are likely to be involved in the ventricular system. The script also determines the center of mass of each object, with the assumption that true ventricular objects will be close to the center of the brain. At this stage, the user selects the number of objects to be further processed, which should include all objects that represent ventricular spaces. The script produces an image of all objects that are greater than the minimum size, and the user is able to click near the center of mass of each object to be included in the final selection. Once the relevant objects are selected, the script saves those selections and removes extraneous objects from further analysis. This process is repeated for each of the three contiguous slices.

After the ventricular spaces have been selected, the script isolates the periventricular voxels. This is done by performing two sequential dilation steps, incorporating voxels that are one and two spaces away from the ventricles and then subtracting the original ventricle objects from the dilated objects. The final periventricular ROI is then saved, and it can be sampled for individual DTI parameters.

The anterior (frontal) and posterior (occipital) horns of the lateral ventricle may be independently segmented. In order to select only the frontal and occipital horns, the previously segmented "full" periventricular objects are further modified as follows.

- The center point of the periventricular ROIs in the anterior-posterior (A-P) direction is calculated. This divides the brain into an anterior and posterior segment.
- The script then locates the most anterior and most posterior voxels that are also included in the ROI. This identifies the most anterior and posterior aspects of the ventricles which are then used to select all voxels that are within a predetermined A-P distance from these most extreme points for inclusion in the frontal and occipital horn perimeters.
- An additional step separates the left frontal, right frontal, left occipital, and right occipital objects for individual analysis.

Step-by-step guide

All MATLAB scripts required to run the program is included here. To run the program effectively, and to avoid bugs, we recommend the folder/file structure outlined on Supplementary Fig. 1. Should the user decide to use our recommended structure, all the dMRI images to be analyzed may be placed into folders separated by their "**Study Groups**". The following MATLAB scripts can then be used as is to create the 12 files in the "Codes" folder, which are all required for parsing the LVP ("full') and frontal-occipital horn ("corners") regions.

1	**add_moments.m**

	% add new pixel to existing blob
	function [blob1, bimg] = add_moments(bimg,i,nblob,blob1,nx)
	
	x = mod((i-1),nx)+1;
	y = floor((i-1)/nx)+1;
	z = 0;
	
	bimg(i) = nblob;
	blob1.m000 = blob1.m000 + 1;
	blob1.m100 = blob1.m100 + x;
	blob1.m010 = blob1.m010 + y;
	blob1.m001 = blob1.m001 + z;
	blob1.m110 = blob1.m110 + x*y;
	blob1.m101 = blob1.m101 + x*z;
	blob1.m011 = blob1.m011 + y*z;
	blob1.m200 = blob1.m200 + x^2;
	blob1.m020 = blob1.m020 + y^2;
	blob1.m002 = blob1.m002 + z^2;
	Return

2	**bindilate.m**

```
% dilate 2d binary image
% N ranges from coarse 1 to fine 7
% change 0 to 1 if #n=1 is >=N
function [out2d] = bindilate(img2d, N)

[nx, ny] = size(img2d);

out2d = zeros(nx, ny);

for i = 2:nx-1
   for j = 2:ny-1

      if img2d(i,j)==1
         out2d(i,j) = 1;
         continue
      end
      sum = img2d(i-1,j-1) + img2d(i-1,j) + img2d(i-1,j+1)+ ...
         img2d(i,j-1) +                img2d(i,j+1)+ ...
         img2d(i+1,j-1) + img2d(i+1,j) + img2d(i+1,j+1);

      if sum >= N
         out2d(i,j) = 1;
      else
         out2d(i,j) = 0;
      end
   end
end
```

3	**binerode.m**

```
% erode 2d binary image
% N ranges from coarse 1 to fine 7
% change 1 to 0 if #n=0 is >=N
function [out2d] = binerode(img2d, N)

[nx, ny] = size(img2d);

out2d = zeros(nx, ny);

for i = 2:nx-1
   for j = 2:ny-1

      if img2d(i,j)==0
         out2d(i,j) = 0;
         continue
      end
      sum = img2d(i-1,j-1) + img2d(i-1,j) + img2d(i-1,j+1)+ ...
         img2d(i,j-1) +                img2d(i,j+1)+ ...
         img2d(i+1,j-1) + img2d(i+1,j) + img2d(i+1,j+1);

      if 8-sum >= N
         out2d(i,j) = 0;
      else
         out2d(i,j) = 1;
      end
   end
end

return
```

4	**calc_blobs.m**

```
% calc_blobs
% convert from raw blobs to real blobs
% keep only objects with area > areamin
% nblob: is the final number of raw blobs
% lub: look up blob, table to point from raw blob to new blob number
% nobj: number of real blobs

function [nobj, lub, objs] = calc_blobs(nblob,areamin, adj_reg,parents,lub,blob,objs);

nobj = 0;

for i = 1:nblob
    if adj_reg(i) == 0
        continue
    end
    lub(i) = i;
    next = adj_reg(i);
    while (next ~= i)
        blob(i).m000 = blob(i).m000 + blob(next).m000;
        blob(i).m100 = blob(i).m100 + blob(next).m100;
        blob(i).m010 = blob(i).m010 + blob(next).m100;
        blob(i).m001 = blob(i).m001 + blob(next).m001;
        blob(i).m110 = blob(i).m110 + blob(next).m110;
        blob(i).m101 = blob(i).m101 + blob(next).m101;
        blob(i).m011 = blob(i).m011 + blob(next).m011;
        blob(i).m200 = blob(i).m200 + blob(next).m200;
        blob(i).m020 = blob(i).m020 + blob(next).m020;
        blob(i).m002 = blob(i).m002 + blob(next).m002;
        tmp = adj_reg(next);
        lub(next) = i;
        adj_reg(next) = 0;
        for j = 1:nblob
            if parents(j) == next
                parents(j) = i;
            end
        end
        next = tmp;
    end

    if blob(i).fgflg==1 && blob(i).m000>=areamin
        nobj = nobj + 1;
        objs(nobj).fgflg = 1;
        objs(nobj).xbeg = blob(i).xbeg;
        objs(nobj).ybeg = blob(i).ybeg;
        objs(nobj).zbeg = blob(i).zbeg;
        objs(nobj).m000 = blob(i).m000;
        objs(nobj).m100 = blob(i).m100;
        objs(nobj).m010 = blob(i).m010;
        objs(nobj).m001 = blob(i).m001;
        objs(nobj).m110 = blob(i).m110;
        objs(nobj).m101 = blob(i).m101;
        objs(nobj).m011 = blob(i).m011;
        objs(nobj).m200 = blob(i).m200;
        objs(nobj).m020 = blob(i).m020;
        objs(nobj).m002 = blob(i).m002;
        objs(nobj).nblob = i;
        objs(nobj).parent = parents(i);
    end
end

return
```

5	**connect_reg.m**

```
% connect_reg
% make note of blobs that are connected
% input n,m of the two blobs to be connected
% make change in adj_reg array to track this info

function [adj_reg] = connect_reg(adj_reg, n, m)

tmp = adj_reg(n);
while (tmp ~= n)
   if tmp == m
      return
   end
   tmp = adj_reg(tmp);
end

tmp = adj_reg(n);
adj_reg(n) = adj_reg(m);
adj_reg(m) = tmp;

return
```

6 **corners.m**

```
% Keep corners; does each hemisphere separately, so image must be
% approximately midline
% Keep only most anterior and posterior 'limnum' voxels
% img: input image
% limnum: number of voxels to limit corners
% Nx: size in x direction
% Ny: size in y direction
% select: 1=RF, 2=LF, 3=RO, 4=LO, 5=all 4
```

```
function img_out = corners(img, limnum, Nx, Ny, select)

 img_out = img;
 Nxhalf = floor(Nx/2);

  % first hemisphere
  ymax = Ny/2 + limnum;
  ymin = Ny/2 - limnum;

  for p = 1:Nxhalf
     for q = 1:Ny
        if img(p,q)==0
           continue
        end
        if q > ymax
           ymax = q;
        end
        if q < ymin
           ymin = q;
        end
     end
  end

  for p = 1:Nxhalf
     for q = 1:Ny
        if img(p,q)==0
           continue
        end

        if ismember(select,[1 3])
           img_out(p,q)=0;
        elseif ymax - q >= limnum && q - ymin >= limnum && select==5
           img_out(p,q)=0;
        elseif q - ymin >= limnum && select==2
           img_out(p,q)=0;
        elseif ymax - q >= limnum && select==4
           img_out(p,q)=0;
        end
     end
  end

  % second hemisphere
  ymax = Ny/2 + limnum;
  ymin = Ny/2 - limnum;

  for p = (Nxhalf+1):Nx
     for q = 1:Ny
        if img(p,q)==0
           continue
        end
        if q > ymax
           ymax = q;
        end
        if q < ymin
           ymin = q;
        end
     end
```

```
    end

    for p = (Nxhalf+1):Nx
        for q = 1:Ny
            if img(p,q)==0
                continue
            end

            if ismember(select,[2 4])
                img_out(p,q)=0;
            elseif ymax - q >= limnum && q - ymin >= limnum && select==5
                img_out(p,q)=0;
            elseif q - ymin >= limnum && select==1
                img_out(p,q)=0;
            elseif ymax - q >= limnum && select==3
                img_out(p,q)=0;
            end
        end
    end

end
```

7 **find_blob.m**

```
% blob finding routine
% searches input img2d in column order
% keeps only foreground blobs with area greater than areamin
% returns the number of objects, critical values on onjects
% and final image

function [nobj, objval, out2d] = find_blob(areamin, img2d)

[nx, ny] = size(img2d);
out1d = zeros(1, nx*ny);  % output 1d
out2d = zeros(nx,ny);  % output 2d

Maxblob = 500;
oneblob = zeros(1,1);
initblob = zeros(1,Maxblob);
adj_reg = zeros(1,Maxblob);
parents = zeros(1,Maxblob);

blob = struct('fgflg', initblob, ...
    'm000', initblob, 'nblob', initblob, 'parent', initblob, ...
    'xbeg', initblob, 'ybeg', initblob, 'zbeg', initblob, ...
    'm100', initblob, 'm010', initblob, 'm001', initblob, ...
    'm110', initblob, 'm101', initblob, 'm011', initblob, ...
    'm200', initblob, 'm020', initblob, 'm002', initblob);

blob1 = struct('fgflg', oneblob, ...
    'm000', initblob, 'nblob', initblob, 'parent', initblob, ...
    'xbeg', oneblob, 'ybeg', oneblob, 'zbeg', oneblob, ...
    'm100', oneblob, 'm010', oneblob, 'm001', oneblob, ...
    'm110', oneblob, 'm101', oneblob, 'm011', oneblob, ...
    'm200', oneblob, 'm020', oneblob, 'm002', oneblob);

objs = struct('fgflg', initblob, ...
    'm000', initblob, 'nblob', initblob, 'parent', initblob, ...
    'xbeg', initblob, 'ybeg', initblob, 'zbeg', initblob, ...
    'm100', initblob, 'm010', initblob, 'm001', initblob, ...
    'm110', initblob, 'm101', initblob, 'm011', initblob, ...
```

```
    'm200', initblob, 'm020', initblob, 'm002', initblob);

% open file
bimg = reshape(img2d,[1,nx*ny]);

% background blob starts with 1
nblob = 0;

% create frame as bg area in 1st blob
[nblob, blob(nblob), bimg] = init_moments(1,nx,0,0,nblob,blob1,bimg);

%blob(nblob) = blob1;
adj_reg(nblob) = nblob;
parents(nblob) = 0; % parent of main bg is zero

% fill first column as bg
for i = 1:nx+1
   bimg(i) = nblob;
end

% loop over image
for i = nx+2:nx*ny-1
   ba = i - nx + 1;
   bb = i - nx;
   bc = i - nx - 1;
   bd = i - 1;
   be = i + 1;

   % test patch of the array
%    fprintf('%d:\n',i);
%    fprintf('%3d %3d %3d\n',bimg(bc),bimg(bb),bimg(ba));
%    fprintf('%3d %3d %3d\n',bimg(bd),bimg(i),bimg(be));

   if nblob >= Maxblob
      fprintf('Error: number of raw blobs too large\n');
      return
   end

   if bimg(i) == 1  % fg
      if blob(bimg(ba)).fgflg == 1
         [blob(bimg(ba)), bimg] = add_moments(bimg,i,bimg(ba),blob(bimg(ba)),nx);
         if blob(bimg(bd)).fgflg==1 && bimg(ba)~=bimg(bd)
            adj_reg = connect_reg(adj_reg,bimg(ba),bimg(bd));
         elseif blob(bimg(bc)).fgflg==1 && bimg(ba)~=bimg(bc)
            adj_reg = connect_reg(adj_reg,bimg(ba),bimg(bc));
         end
      elseif blob(bimg(bb)).fgflg == 1
         [blob(bimg(bb)), bimg] = add_moments(bimg,i,bimg(bb),blob(bimg(bb)),nx);
      elseif blob(bimg(bc)).fgflg == 1
         [blob(bimg(bc)), bimg] = add_moments(bimg,i,bimg(bc),blob(bimg(bc)),nx);
      elseif blob(bimg(bd)).fgflg == 1
         [blob(bimg(bd)), bimg] = add_moments(bimg,i,bimg(bd),blob(bimg(bd)),nx);
      else
         [nblob, blob1, bimg] = init_moments(i,nx,1,bimg(bd),nblob,blob1,bimg);
         blob(nblob) = blob1;
         adj_reg(nblob) = nblob;
         parents(nblob) = bimg(bd);
      end
   else  % bg
      if blob(bimg(bb)).fgflg==0
```

```
            [blob(bimg(bb)), bimg] = add_moments(bimg,i,bimg(bb),blob(bimg(bb)),nx);
            if blob(bimg(bd)).fgflg==0 && bimg(bb)~=bimg(bd)
               adj_reg = connect_reg(adj_reg,bimg(bb),bimg(bd));
            end
         elseif blob(bimg(bd)).fgflg==0
            [blob(bimg(bd)), bimg] = add_moments(bimg,i,bimg(bd),blob(bimg(bd)),nx);
         elseif blob(bimg(ba)).fgflg==0 && bimg(be)==0
            [blob(bimg(ba)), bimg] = add_moments(bimg,i,bimg(ba),blob(bimg(ba)),nx);
         else
            [nblob, blob1, bimg] = init_moments(i,nx,0,bimg(bd),nblob,blob1,bimg);
            blob(nblob) = blob1;
            adj_reg(nblob) = nblob;
            parents(nblob) = bimg(bd);
         end
      end

%     for j = 1:nblob
%     fprintf('nblob %d: adj_reg %d parents %d fgflg %d area %d\n', ...
%        j,adj_reg(j),parents(j),blob(j).fgflg,blob(j).m000);
%     end

end

nobj = 0;
out2d = reshape(bimg, [nx,ny]);
% figure
% imshow(out2d, []);

% print out raw blobs
% for i = 1:nblob
%    fprintf('nblob %d: adj_reg %d parents %d fgflg %d area %d\n', ...
%       i,adj_reg(i),parents(i),blob(i).fgflg,blob(i).m000);
% end

% lub = look up blobs, converts from raw blobs to final blobs
lub = zeros(1,nblob+1);
% areamin is input
[nobj, lub, objs] = calc_blobs(nblob,areamin,adj_reg,parents,lub,blob,objs);

% return critical values of:
% 1 blob number, 2 area,
% 3 x center, 4 y center
% 5 sum x2, 6 sum y2 around center, 7 xy around center
objval = zeros(7,nobj);
for i = 1:nobj
   objval(1,i) = objs(i).nblob;
   objval(2,i) = objs(i).m000;
   objval(3,i) = objs(i).m100/objs(i).m000;
   objval(4,i) = objs(i).m010/objs(i).m000;
   objval(5,i) = objs(i).m200/objs(i).m000 - objval(3,i)^2;
   objval(6,i) = objs(i).m020/objs(i).m000 - objval(4,i)^2;
   objval(7,1) = objs(i).m110/objs(i).m000 - objval(3,i)*objval(4,i);

   fprintf('nobj %d: nblob %d prt %d fg %d area %d xcen %f ycen %f x2mom %f y2mom
%f\n', ...
      i,objs(i).nblob,objs(i).parent,objs(i).fgflg,objs(i).m000, ...
      objval(3,i),objval(4,i),objval(5,i),objval(6,i));
end

% for i = 1:nblob+1
```

```
%     fprintf('lub %d -> %d\n',i,lub(i));
% end

% final image
for i = nx+2:nx*ny-1
   out1d(i) = lub(bimg(i));
end
out2d = reshape(out1d, [nx,ny]);

return
```

8 **ImgDisp.m**

```
% ImgDisp: improved image display
% Assumes image was acquired transversely
% img = input 3D image
% tcs = transverse, sagittal, or coronal
% slicelo, slicehi = range of slices to display
% mag = magnify factor, small integer (1-4)
% colorflag = 0 or 1 to display in color
% imgtitle = title for img
```

```
function ImgDisp(img, tcs, slicelo, slicehi, mag, colorflag, imgtitle)

   [Nx Ny Nz] = size(img);

   imgvec = reshape(img, [1 numel(img)]);

   mag = floor(mag);

   % error checking
   if mag==0
      fprintf('Error(ImgDisp): mag cannot be zero\n');
      return
   end

   % get new dimensions and displacements in fft array
   Nxp = floor(mag*Nx);
   Nyp = floor(mag*Ny);
   Nxd = floor(0.5*Nx*(mag-1));
   Nyd = floor(0.5*Ny*(mag-1));

   Fimg = complex(zeros(Nxp, Nyp), zeros(Nxp, Nyp));
   Lintimg = zeros(mag*[Nx Ny]);

   if strcmp(tcs,'tra')
      if mag==1
         for n = slicelo:slicehi
            figure;
            imshow(squeeze(img(:,:,n)), []);
            % title({imgtitle;['Slice',int2str(n)]});
            title([imgtitle, ';slice',int2str(n)]);
         end
         return
      end

      for n = slicelo:slicehi

         % fourier method commented out
         f1img = fft2(squeeze(img(:,:,n)));
         f2img = fftshift(f1img);
```

```
        for i = 1:Nx
            for j = 1:Ny
                Fimg(i+Nxd, j+Nyd) = f2img(i,j);
            end
        end
        fintimg = ifft2(fftshift(Fimg),'symmetric');

        %figure
        %imshow(fintimg, []);
        %title([imgtitle, ';slice',int2str(n)]);

        % do the linear interpolation method
        for i = 1:Nx-1
            for j = 1:Ny-1

                for k = 1:mag
                    t = (k-1)/mag;
                    for l = 1:mag
                        u = (l-1)/mag;

                        Lintimg(mag*(i-1)+k, mag*(j-1)+l) = ...
                            (1-t)*(1-u)* img(i,j,n) + ...
                            t*(1-u)* img(i+1,j,n) + ...
                            (1-t)*u* img(i,j+1,n) + ...
                            t*u* img(i+1,j+1,n);
%                           if i==1 && j==1
%                               fprintf('%d %d : %f %f %f %f\n',k,l, ...
%                                   (1-t)*(1-u),t*(1-u),(1-t)*u,t*u);
%                           end
                    end
                end
            end
        end

        if colorflag == 0
            figure
            hold on
            imshow(Lintimg, []);
            title([imgtitle, ';slice',int2str(n)]);
            colormap(gray);
            hold off
        else
            figure
            hold on
            maxptile = prctile(imgvec(imgvec~=0),99);
            clims = [0 maxptile];
            imshow(Lintimg, clims);
            title([imgtitle, ';slice',int2str(n)]);
            colormap(hot);
            colorbar;
            hold off
        end
    end

elseif strcmp(tcs,'sag')
    newimg = zeros(mag*[Ny Nz]);

    for n = slicelo:slicehi

    for i = 1:Ny
```

```
            for j = 1:Nz

               for k = 1:mag
                  for l = 1:mag
                     newimg(mag*(i-1)+k, mag*(j-1)+l) = img(n,i,j);
                  end
               end

            end
         end

         figure
         imshow(newimg, []);
         title(imgtitle);

         end

      elseif strcmp(tcs,'cor')
         newimg = zeros(mag*[Nx Nz]);

         for n = slicelo:slicehi

         for i = 1:Nx
            for j = 1:Nz

               for k = 1:mag
                  for l = 1:mag
                     newimg(mag*(i-1)+k, mag*(j-1)+l) = img(i,n,j);
                  end
               end
            end
         end

         figure
         imshow(newimg, []);
         title(imgtitle);

         end
      end
   end
```

9 **init_moments.m**

```
% allocate memory for new blob in blob array
% input-
% bimg: blob image array
% i: first pixel of new blob
% nblob: last blob number, will be incremented and returned
% fgflg: 1=foreground 0=background
% ny: number of columns
% output-
% nblob: new blob number
% blob1: new blob information
% adj_reg, and parents need to be set separately
```

```
function [nblob, blob1, bimg] = init_moments(i,nx,fgflg,parent,nblob,blob1,bimg)

x = mod((i-1),nx)+1;
y = floor((i-1)/nx)+1;
z = 0;
```

```
nblob = nblob + 1;
bimg(i) = nblob;

blob1.fgflg = fgflg;
blob1.m000 = 1;
blob1.nblob = nblob;
blob1.parent = parent;
blob1.xbeg = x;
blob1.ybeg = y;
blob1.zbeg = z;
blob1.m100 = x;
blob1.m010 = y;
blob1.m001 = z;
blob1.m110 = x*y;
blob1.m101 = x*z;
blob1.m011 = y*z;
blob1.m200 = x^2;
blob1.m020 = y^2;
blob1.m002 = z^2;

return
```

10 **pvroi.m**

```
% Extract ROI around the ventricles in DTI data
% inputs
%   imgfile: file name with 'img' extension with processed DTI data
%       program will focus on the first volume MD/ADC
%   slc: slice number to focus on thresh: cutoff for the ventricles, use value based on
histogram in "Figure 2" displayed by the script
```

```
function pvroi(imgfile, slc, MDthresh)

close all

% Set assorted params
MDminarea = 20;
MDcendist = 20;
MDmax2mom = 200;

limnum = 4;

RAthresh = 0.18;
RAminarea = 25;
RAcendist = 25;

% read in file size from the ifh file
% form the file ifh file name
slen = max(size(imgfile));
ifhfile = [imgfile(1:(slen-3)) 'ifh'];
outfileFULL = [imgfile(1:(slen-3)) 'roiFULL'];
outfileCRNS = [imgfile(1:(slen-3)) 'roiCRNS'];
outfileRF = [imgfile(1:(slen-3)) 'roiRF'];
outfileLF = [imgfile(1:(slen-3)) 'roiLF'];
outfileRO = [imgfile(1:(slen-3)) 'roiRO'];
outfileLO = [imgfile(1:(slen-3)) 'roiLO'];
fprintf('Reading ifh file: %s\n',ifhfile);

% read the ifh file
msize = zeros(1,4);
ndim = 1;
```

```
fid = fopen(ifhfile,'r');
for j = 1:100
   tstr = fscanf(fid,'%s',1);
   % fprintf('%d %s\n',j,tstr);
   if strcmp(tstr,'matrix')
      for i = 1:3
         tstr = fscanf(fid,'%s',1);
      end
      msize(ndim) = fscanf(fid,'%d',1);
      ndim = ndim + 1;
   end
   if ndim == 5
      break
   end
end
fclose(fid);

% check img dimensions from the ifh file
fprintf('Input dimensions: %d %d %d %d\n',msize(1),msize(2),msize(3),msize(4));

% basic error checking on slc
if msize(3)<3 || slc<2 || slc>msize(3)-1
   fprintf('Error(pvroi): bad input value in either matrix dim[3] or slice\n');
   return
end

% image size in DTI space
Nx = msize(1);
Ny = msize(2);
Nz = msize(3);
Nt = msize(4);
N3D = Nx*Ny*Nz;

% allocate memory for images
img3d = zeros([Nx,Ny,Nz], 'single');
img2d = zeros([Nx, Ny], 'single');
imgMD = zeros([Nx, Ny], 'single');
imgRA = zeros([Nx, Ny], 'single');
imgtmp = zeros([1 N3D], 'single');
imgtst = zeros([1 N3D], 'single');
imgoutFULL = zeros([Nx,Ny,Nz], 'single');
imgoutCRNS = zeros([Nx,Ny,Nz], 'single');
imgoutRF = zeros([Nx,Ny,Nz], 'single');
imgoutLF = zeros([Nx,Ny,Nz], 'single');
imgoutRO = zeros([Nx,Ny,Nz], 'single');
imgoutLO = zeros([Nx,Ny,Nz], 'single');

imgA = zeros([Nx, Ny], 'single');
imgB = zeros([Nx, Ny], 'single');
imgC = zeros([Nx, Ny], 'single');
imgD = zeros([Nx, Ny], 'single');
imgE = zeros([Nx, Ny], 'single');

imga = zeros([Nx, Ny], 'single');
imgb = zeros([Nx, Ny], 'single');
imgc = zeros([Nx, Ny], 'single');
imgd = zeros([Nx, Ny], 'single');

imgm1 = zeros([Nx, Ny], 'single');
imgm2 = zeros([Nx, Ny], 'single');
```

```
imgm3 = zeros([Nx, Ny], 'single');
imgm4 = zeros([Nx, Ny], 'single');

imgr1 = zeros([Nx, Ny], 'single');
imgr2 = zeros([Nx, Ny], 'single');
imgr3 = zeros([Nx, Ny], 'single');
imgr4 = zeros([Nx, Ny], 'single');

%% Read input files
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = 1:1
   imgtmp = fread(fid, [1, N3D], 'float', 'l');
   DTIdata(i,:) = imgtmp(:);

   img3d = reshape(squeeze(imgtmp(1,:)), [Nx, Ny, Nz]);
   if i == 1
      imgtst = img3d(:,:,slc);
   end
   % display the slice(s) to focus on
   ImgDisp(img3d, 'tra', slc, slc, 2, 0, [imgfile(1:5) '; DTI']);
   % ImgDisp(img3d, 'tra', slc-1, slc+1, 2, 0, 'DTI');
end
fclose(fid);

% From here the program can do 1,2, or 3 slices saving the final ROIs
for ii = slc-1:slc+1

   % Display histogram of MD and RA
   img3d = reshape(squeeze(DTIdata(1,:)), [Nx, Ny, Nz]);
   imgMD = squeeze(img3d(:,:,ii));
   img3d = reshape(squeeze(DTIdata(2,:)), [Nx, Ny, Nz]);
   imgRA = squeeze(img3d(:,:,ii));

   % histograms
   figure
   hist(reshape(imgMD,[1,numel(imgMD)]),100);
   ylim([0 1000]);
   title([imgfile(1:5) '; Distribution of MD on slice ' num2str(ii)])

   MDmax = max(imgMD(:));
   MDmin = min(imgMD(:));
   fprintf('MD min %f mode %f max %f\n',MDmin,MDmax);

   % simple threshold for MD and RA calculations
   imgA = (imgMD > MDthresh);
   imga = (imgRA > RAthresh);

   % display after erode and dilate
   imgB = binerode(imgA, 5);
   imgC = bindilate(imgB, 5);
   imgD = zeros([Nx, Ny], 'single');
   imgb = binerode(imga, 5);
   imgc = bindilate(imgb, 5);


   %% MD analysis
   % find the blobs larger than 10 in size
   % nobj1 is the total number blobs, imgB only has the ones > minarea
   % objval 1 nobj 2 mass 3 xcen 4 ycen 5 x2mom 6 y2mom
```

```
[nobj1, objval1, imgB] = find_blob(10, imgC);
% display blobs, 1 is coded as background
figure
imshow(imgB,[1 2]);

np = input('Input number of regions (default=1) then click on or near these regions: ');
if isempty(np) || np<=0
   fprintf('Error: number of regions must be >= 1\n');
   return
end

% IMPORTANT: different x,y convention in matlab
[yp, xp] = ginput(np);
% save blobs close to clicks
for n = 1:np
   distmin = 100.0; k1 = 0;
   for k = 1:nobj1
      dist = sqrt((objval1(3,k) - xp(n))^2 + (objval1(4,k) - yp(n))^2);

      if dist < distmin
         distmin = dist;
         k1 = k;
      end
   end

   % printf the kept blobs
   fprintf('MD nobj=%d area=%f distmin=%f\n',k1,objval1(2,k1),distmin);
   for i = 2:Nx-1
      for j = 2:Ny-2
         if imgB(i,j)==objval1(1,k1)
            imgD(i,j) = 1;
         end
      end
   end

end


% get perimeter
imgA = bindilate(imgD, 2);
imgD = bindilate(imgA, 2);
imgC = bindilate(imgD, 2);
imgB = imgC - imgA;

% regions
close
figure
imshow(imgB,[]);
title([imgfile(1:5) '; Selected regions on slice ' num2str(ii)])


%% RA analysis
RA_flag = 0; % use to suppress RA analysis
if RA_flag ~= 0
[nobj2, objval2, imgb] = find_blob(25, imgc);
figure
imshow(imgb,[]);

% save blobs with area > minarea and near center
keepin = zeros(4,1);
```

```
index = 0;
img2d(Nx/2,Ny/2) = 2;
for k = 1:nobj2
   dcen = sqrt((objval2(3,k) - (Nx/2))^2 + (objval2(4,k) - (Ny/2))^2);
   fprintf('RA nobj=%d area=%f dcen=%f\n',k,objval2(2,k),dcen);

   % keep if area above min and near center
   if objval2(2,k) > RAminarea && dcen < RAcendist
      fprintf('keep ^^^^\n');
      index = index + 1;
      if index >= 5
         fprintf('If index>5 need to fix program a bit: exiting\n');
         return
      end
      keepin(index,1) = k;
      % display all kept regions
      for i = 2:Nx-1
         for j = 2:Ny-2
            if imgb(i,j)==objval2(1,k)
               img2d(i,j) = 1;
            end
         end
      end
      img2d(round(objval2(3,k)),round(objval2(4,k))) = 2;
   end
end

figure
imshow(img2d,[]);

% select the best two
ymin = Ny; ymax = 0;
kmin = 0; kmax = 0;
for i = 1:index
   if objval2(4,keepin(i,1)) < ymin
      ymin = objval2(4,keepin(i,1));
      kmin = keepin(i,1);
   end
   if objval2(4,keepin(i,1)) > ymax
      ymax = objval2(4,keepin(i,1));
      kmax = keepin(i,1);
   end
end

% show kept blobs
% mark the kept blobs + shift them
for i = 2:Nx-1
   for j = 2:Ny-2
      if imgb(i,j)==objval2(1,kmin)
         imgd(i,j-2) = 1;
      end
      if imgb(i,j)==objval2(1,kmax)
         imgd(i,j+2) = 1;
      end
   end
end


figure
imshow(imgd,[]);
```

```
    figure
    imshow(imgtst,[]);

    figure
    imshow(imgtst-imgtst.*imgd,[]);



    end

    % store for output
    for m = 1:Nx
        for n = 1:Ny
            imgoutFULL(m,n,ii) = imgB(m,n);
        end
    end

    imgoutCRNS(:,:,ii) = corners(imgB,limnum,Nx,Ny,5);
    imgoutRF(:,:,ii) = corners(imgB,limnum,Nx,Ny,1);
    imgoutLF(:,:,ii) = corners(imgB,limnum,Nx,Ny,2);
    imgoutRO(:,:,ii) = corners(imgB,limnum,Nx,Ny,3);
    imgoutLO(:,:,ii) = corners(imgB,limnum,Nx,Ny,4);

end

% slice loop on l

% Store the output ROI
fid = fopen(outfileFULL,'w');
fwrite(fid, imgoutFULL, 'float', 'l');
fclose(fid);

fid = fopen(outfileCRNS,'w');
fwrite(fid, imgoutCRNS, 'float', 'l');
fclose(fid);

fid = fopen(outfileRF,'w');
fwrite(fid, imgoutRF, 'float', 'l');
fclose(fid);

fid = fopen(outfileLF,'w');
fwrite(fid, imgoutLF, 'float', 'l');
fclose(fid);

fid = fopen(outfileRO,'w');
fwrite(fid, imgoutRO, 'float', 'l');
fclose(fid);

fid = fopen(outfileLO,'w');
fwrite(fid, imgoutLO, 'float', 'l');
fclose(fid);

tilefigs

return
```

11	**samroi.m** **Sample ROI on DTI data set** **% inputs**

```
%   imgfile: file name with 'img' extension with processed DTI data
%      program will sample roi on each of the volumes
%   roi file: binary roi file with same root and extension 'roi'
%      that will be used to sample each one of the volume of 'img'
%   slc: slice number to focus on for display
```

```
function samroi(imgfile, slc)

close all

% read in file size from the ifh file
% form the file ifh file name
slen = max(size(imgfile));
ifhfile = [imgfile(1:(slen-3)) 'ifh'];
roifileFULL = [imgfile(1:(slen-3)) 'roiFULL'];
roifileCRNS = [imgfile(1:(slen-3)) 'roiCRNS'];
roifileRF = [imgfile(1:(slen-3)) 'roiRF'];
roifileLF = [imgfile(1:(slen-3)) 'roiLF'];
roifileRO = [imgfile(1:(slen-3)) 'roiRO'];
roifileLO = [imgfile(1:(slen-3)) 'roiLO'];
fprintf('Reading ifh file: %s\n',ifhfile);

msize = zeros(1,4);
ndim = 1;
fid = fopen(ifhfile,'r');
for j = 1:100
   tstr = fscanf(fid,'%s',1);
   % fprintf('%d %s\n',j,tstr);
   if strcmp(tstr,'matrix')
      for i = 1:3
         tstr = fscanf(fid,'%s',1);
      end
      msize(ndim) = fscanf(fid,'%d',1);
      ndim = ndim + 1;
   end
   if ndim == 5
      break
   end
end
fclose(fid);

% check img dimensions from the ifh file
fprintf('Input dimensions: %d %d %d %d\n',msize(1),msize(2),msize(3),msize(4));

% image size in DTI space
Nx = msize(1);
Ny = msize(2);
Nz = msize(3);
Nv = msize(4);
N3D = Nx*Ny*Nz;

% allocate memory for images
imgtmp = zeros([1, N3D], 'single');
imgroi = zeros([1, N3D], 'single');
imgsum = zeros([1, N3D], 'single');

%% Read input files - Full trace
% read in roi file first
fid = fopen(roifileFULL,'r');
imgroi = fread(fid, [1, N3D], 'float', 'l');
```

```
fclose(fid);

% number of fg pixels, assumes all=1
npix = sum(imgroi(:));
fprintf('Full trace npix = %d\n',npix);

% read the different volume in loop and sample each
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = [1 9 12 13]
   imgtmp = fread(fid, [1, N3D], 'float', 'l');
   DTIdata(i,:) = imgtmp(:);

   imgsum = imgtmp .* imgroi;
   imgconf = imgtmp .* (1-imgroi);
   nsum = sum(imgsum(:));
   fprintf('Full trace ROI sampled on vol %d = %f\n',i,nsum/npix);

   %!!! displays for volumes of interest
   if any(i == 1)
      % display heat maps for the 3 slices

      img3dinv = reshape(squeeze(imgconf(1,:)), [Nx, Ny, Nz]);
      ImgDisp(img3dinv, 'tra', slc-1, slc+1, 2, 1, [imgfile(1:5) ' full; volume ' num2str(i)]);

   end

end
fclose(fid);

%% Read input files - Corners
% read in roi file first
fid = fopen(roifileCRNS,'r');
imgroi = fread(fid, [1, N3D], 'float', 'l');
fclose(fid);

% number of fg pixels, assumes all=1
npix = sum(imgroi(:));
fprintf('Corners npix = %d\n',npix);

% read the different volume in loop and sample each
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = [1 9 12 13]
   imgtmp = fread(fid, [1, N3D], 'float', 'l');
   DTIdata(i,:) = imgtmp(:);

   imgsum = imgtmp .* imgroi;
   imgconf = imgtmp .* (1-imgroi);
   nsum = sum(imgsum(:));
   fprintf('Corners ROI sampled on vol %d = %f\n',i,nsum/npix);

   %!!! displays for volumes of interest
   if any(i == 1)

      img3dinv = reshape(squeeze(imgconf(1,:)), [Nx, Ny, Nz]);
      ImgDisp(img3dinv, 'tra', slc-1, slc+1, 2, 1, [imgfile(1:5) ' corners; volume ' num2str(i)]);

   end
```

```
end
fclose(fid);


%% Read input files - RF
% read in roi file first
fid = fopen(roifileRF,'r');
imgroi = fread(fid, [1, N3D], 'float', 'l');
fclose(fid);

% number of fg pixels, assumes all=1
npix = sum(imgroi(:));
fprintf('RF npix = %d\n',npix);

% read the different volume in loop and sample each
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = [1 9 12 13]
   imgtmp = fread(fid, [1, N3D], 'float', 'l');
   DTIdata(i,:) = imgtmp(:);

   imgsum = imgtmp .* imgroi;
   imgconf = imgtmp .* (1-imgroi);
   nsum = sum(imgsum(:));
   fprintf('RF ROI sampled on vol %d = %f\n',i,nsum/npix);

   %!!! displays for volumes of interest
   if any(i == 1)

      img3dinv = reshape(squeeze(imgconf(1,:)), [Nx, Ny, Nz]);
      ImgDisp(img3dinv, 'tra', slc-1, slc+1, 2, 1, [imgfile(1:5) ' RF; volume ' num2str(i)]);

   end

end
fclose(fid);


%% Read input files - LF
% read in roi file first
fid = fopen(roifileLF,'r');
imgroi = fread(fid, [1, N3D], 'float', 'l');
fclose(fid);

% number of fg pixels, assumes all=1
npix = sum(imgroi(:));
fprintf('LF npix = %d\n',npix);

% read the different volume in loop and sample each
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = [1 9 12 13]
   imgtmp = fread(fid, [1, N3D], 'float', 'l');
   DTIdata(i,:) = imgtmp(:);

   imgsum = imgtmp .* imgroi;
   imgconf = imgtmp .* (1-imgroi);
   nsum = sum(imgsum(:));
   fprintf('LF ROI sampled on vol %d = %f\n',i,nsum/npix);
```

```
    %!!! displays for volumes of interest
    if any(i == 1)

        img3dinv = reshape(squeeze(imgconf(1,:)), [Nx, Ny, Nz]);
        ImgDisp(img3dinv, 'tra', slc-1, slc+1, 2, 1, [imgfile(1:5) ' LF; volume ' num2str(i)]);

    end

end
fclose(fid);


%% Read input files - RO
% read in roi file first
fid = fopen(roifileRO,'r');
imgroi = fread(fid, [1, N3D], 'float', 'l');
fclose(fid);

% number of fg pixels, assumes all=1
npix = sum(imgroi(:));
fprintf('RO npix = %d\n',npix);

% read the different volume in loop and sample each
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = [1 9 12 13]
    imgtmp = fread(fid, [1, N3D], 'float', 'l');
    DTIdata(i,:) = imgtmp(:);

    imgsum = imgtmp .* imgroi;
    imgconf = imgtmp .* (1-imgroi);
    nsum = sum(imgsum(:));
    fprintf('RO ROI sampled on vol %d = %f\n',i,nsum/npix);

    %!!! displays for volumes of interest
    if any(i == 1)

        img3dinv = reshape(squeeze(imgconf(1,:)), [Nx, Ny, Nz]);
        ImgDisp(img3dinv, 'tra', slc-1, slc+1, 2, 1, [imgfile(1:5) ' RO; volume ' num2str(i)]);

    end

end
fclose(fid);


%% Read input files - LO
% read in roi file first
fid = fopen(roifileLO,'r');
imgroi = fread(fid, [1, N3D], 'float', 'l');
fclose(fid);

% number of fg pixels, assumes all=1
npix = sum(imgroi(:));
fprintf('LO npix = %d\n',npix);

% read the different volume in loop and sample each
DTIdata = zeros([6, N3D], 'single');
fid = fopen(imgfile,'r');
for i = [1 9 12 13]
```

```
    imgtmp = fread(fid, [1, N3D], 'float', 'l');
    DTIdata(i,:) = imgtmp(:);

    imgsum = imgtmp .* imgroi;
    imgconf = imgtmp .* (1-imgroi);
    nsum = sum(imgsum(:));
    fprintf('LO ROI sampled on vol %d = %f\n',i,nsum/npix);

    %!!! displays for volumes of interest
    if any(i == 1)

      img3dinv = reshape(squeeze(imgconf(1,:)), [Nx, Ny, Nz]);
      ImgDisp(img3dinv, 'tra', slc-1, slc+1, 2, 1, [imgfile(1:5) ' LO; volume ' num2str(i)]);

    end

end
fclose(fid);

tilefigs
```

12	tilefigs.m Plum's[2] tilefigs.m program (https://www.mathworks.com/matlabcentral/fileexchange/328-tilefigs-m may be used to display the generated images.

Data validation

The methods presented herein have been validated in a prospective cohort of human infants who were born preterm and sustained intraventricular hemorrhage with or without Posthemorrhagic hydrocephalus, in comparison to preterm infants with no identifiable brain injury and full term healthy infants [1]. To assess for potential differences between the algorithm and manually placed ROIs, dMRI of 20 neonates of varying ventricular morphologies were selected and manual ROIs were placed by an expert rater. Pearson correlation analyses between dMRI measures from both approaches indicated there was a strong positive association between the automated and manual segmentations fractional anisotropy ($r(20) = .91$, $p < .001$) and MD ($r(20) = .89$, $p < .001$) measures.

Conclusion

We describe a novel semi-automated segmentation algorithm that is able to generate ROIs isolating the perimeter of the lateral ventricles in diffusion-based brain MRI. The algorithm has robust fidelity for locating periventricular regions compared to manually produced ROIs and performs well in various situations of high-grade brain injury and ventriculomegaly. An additional step enables isolation of frontal and occipital regions of the periventricular ROI for separate analysis. Importantly, this method is able to extract regions associated with the ventricular system from other CSF spaces (cysts, cisterns, subarachnoid spaces), which has not been previously possible using traditional approaches.

Acknowledgments

This work was supported by the Vanier Canada Graduate Scholarship [grant number 396212]; National Institutes of Health [grant numbers K02 NS089852, K23 NS075151-01A1, K23 MH105179, TL1 TR002344, P30 NS098577, R01 HD061619, and R01 HD057098]; Child Neurology Foundation; Cerebral Palsy International Research Foundation; The Dana Foundation; March of Dimes Prematurity Research Center at Washington University; The Doris Duke Charitable Foundation; and the Eunice Kennedy Shriver National Institute of Child Health & Human Development of the National Institutes of Health [grant number U54 HD087011]. None of the organizations listed had any role in the study design, data collection, data analysis, data interpretation, writing or decision to submit the report for publication.

We thank Mr. Dimitrios Alexopolous for his help with the manual segmentation and, Ms Tara Smyser and Ms Jeanette Kenley for their help with many aspects of data acquisition and analyses.

Declaration of Competing Interest

Dr. Limbrick receives research funds and/or research equipment for unrelated projects from Medtronic, Inc., Karl Storz, Inc., and Microbot Medical, Inc. Dr. Limbrick has received philanthropic equipment contributions for humanitarian relief work from Karl Storz, Inc. and Aesculap, Inc. The authors have no personal, financial, or institutional interest in any of the materials, or devices described in this article.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.mex.2020.101023.

References

[1] A.M. Isaacs, C.D. Smyser, R.E. Lean, D. Alexopoulos, R.H. Han, J.J. Neil, et al., MR diffusion changes in the perimeter of the lateral ventricles demonstrate periventricular injury in post-hemorrhagic hydrocephalus of prematurity, NeuroImage: Clin. (2019).
[2] C. Plum, tilefigs.m - File Exchange - MATLAB Central, in, 1998, Vol 2017.