AnalyzeMultispectral

AnalyzeMultispectral is a modular image analysis pipeline designed for multispectral images, z-stacks, and time lapses.

I. User Guide

Before starting: make sure you have all your images saved in .tif format.

rawXToMask: X being *Slice*, *TimeLapse*, or *ZStack*. This script will read in a set of .tif files and generate a folder for each that contains a cell mask and segmented objects from each channel.

maskToCounts*, *maskTimeLapseToCounts: This script will read in the folders output by *rawXToMask* and assess the number, size, and fraction of objects in each channel.

maskToContacts*:* From the folders output by *rawXToMask*, $i \in [2,n]$ contacts between all permutations up of organelles will be calculated, with n being user defined. Parameters calculated include number of occurrences, number fraction (# contacts / # organelle A), and area fraction (area overlap pixels / total area organelle A).

a. *rawSliceToMask* – for batches of single slice multispectral images

You will be prompted to enter the number of channels, the folder containing your images, and the channel to use for masking the cell[1]. The program will then cycle through all images in the folder, showing you the channel you chose for masking. Use the pointer to draw an ROI around your cell[2].

You will then be asked what the morphology of each channel is. This affects how that channel will be thresholded and segmented. Close the interface at any time to use the defaults[3]. The images are thresholded, segmented, and saved in new subfolders that have been generated for the corresponding files.

b. *rawZStackToMask* – for batches of multi slice multispectral images

You will be prompted to enter the number of channels, the folder containing your images, and the channel to use for masking the cell[1]. The program will then cycle through all images in the folder, showing you the second slice of each channel you chose for masking. Use the pointer to draw an ROI around your cell[2]. You will be asked if you want to use the ROI as a loose mask or if you want to tighten your mask[4].

You will then be asked what the morphology of each channel is. This affects how that channel will be thresholded and segmented. Close the interface at any time to use the defaults[3]. The images are made into max intensity projections, thresholded, segmented, and saved in new subfolders that have been generated for the corresponding files.

[1] Usually the ER channel.
[2] Single click to add vertex. Double click to close polygon. Double click inside polygon to accept.
[3] [particle, large org, sheet, particle, large org, particle] corresponding to [lysosome, mitochondria, ER, peroxisome, Golgi, lipid droplet]
[4] Always tighten if using ER as the mask channel.

c. *rawTimeLapseToMask* – for single multispectral movies

You will be prompted to enter the number of channels, number of frames, the folder containing your images, and the channel to use for masking the cell[1]. The program show the second frame of the channel you chose for masking. Use the pointer to draw an ROI around your cell[2].

You will then be asked what the morphology of each channel is. This affects how that channel will be thresholded and segmented. Close the interface at any time to use the defaults[3]. A subfolder is generated for each frame of the movie; images are thresholded, segmented, and saved in the folder corresponding to their time point.

d. *maskToCounts* - for batches of folders containing cell and organelle masks

You will be prompted to select the folders you want to analyze. Then, you will be asked to assign names to each of the channels. Close the interface at any time to use the defaults[4]. For each channel, the script will calculate number, mean/median area, and area fraction of the organelle. Results are saved in the parent folder as ‘organelle counts.csv’.

e. *maskTimeLapseToCounts* – for batches of folders containing cell and organelle masks of a single cell at different time points

You will be prompted to select the folders you want to analyze. Then, you will be asked to assign names to each of the channels. Close the interface at any time to use the defaults[4]. For each channel of every frame, the script will calculate number, mean/median area, and area fraction of the organelle. Results are saved in the parent folder as ‘organelle counts.csv’.

f. *maskToContacts* – for batches of folders containing cell and organelle masks

You will be prompted to select the folders and channels you want to analyze[5]. Then, you will be asked to assign names to each of the channels. Close the interface at any time to use the defaults[4]. Next, you will be asked the degrees of contact *n* you wish to compute to. Close the interface at any time to use the default of binary.

For every permutation of the selected channels, contacts of degree $i \in [2,n]$ will be assessed for number, number fraction, and area fraction.

[1] Usually the ER channel.
[2] Single click to add vertex. Double click to close polygon. Double click inside polygon to accept.
[3] [particle, large org, sheet, particle, large org, particle] corresponding to [lysosome, mitochondria, ER, peroxisome, Golgi, lipid droplet]
[4] [mask, lysosome, mitochondria, ER, peroxisome, Golgi, lipid droplet]
[5] Saves time if you wish to include only a subset of channels in contact analysis.

II. Technical Guide

a. *rawSliceToMask*

```
clear variables
close all
commandwindow;

%MASKING PARAMETER SETUP
numCh = str2double(input('How many channels are there? \n','s'));
filePath = uigetdir('Select folder with images to mask');
addpath(filePath);
fileList = dir([filePath '/*.tif']);

%IMPORT IMAGES
numFiles = size(fileList,1);
images = cell(numFiles,numCh);
for i = 1:numFiles
    for j = 1:numCh
        images{i,j} = imread(fileList(i).name,j);
    end
end
```

Number of channels saved as string **numCh**.

Folder location saved as string **filePath** and added to working directory. List of .tif files in file path generated as **fileList**.

```
maskCh = str2double(input(['Of the ' num2str(numCh) ' channels, which should be used for masking 
the cell? \n'], 's'));
roi = cell(1,numFiles);
for i = 1:numFiles
    brightImg = imadjust(images{i,maskCh},[0 0.2]);
    imshow(brightImg);
    roi{i} = roipoly;
    if isempty(roi{i})
        try
            roi{i} = imbinarize(ones(size(images{3},1),size(images{3},2)));
        catch
            roi{i} = imbinarize(ones(size(images{1},1),size(images{1},2)));
        end
    end
end
```

User enters the channel being used for cell masking, saved as double **maskCh**. Structure **roi** created with a page for each file.

User manually draws ROI for each cell on a contrast adjusted image; ROI saved as binary mask in **roi**. If ROI drawing interface is closed, will auto-generate whole-image ROI from channel 3, or, in lieu of channel 3, from channel 1.

```
%MASK CHANNELS
close(figure(1));
for i = 1:numFiles
    mkdir(fullfile(filePath, strrep(fileList(i).name,'.tif','')));
    savePath = fullfile(filePath, strrep(fileList(i).name,'.tif',''));    %Apply ROI to image and
blur
    roi{i} = immultiply(roi{i},images{i,maskCh});
    roi{i} = medfilt2(roi{i}, [15 15]);
%Calculate threshold from blurred image
    t = double(multithresh(roi{i},3));
    t = t(1)/65535;
%Make binary, fill holes, exclude small objects
    mask = bwareaopen(imfill(imbinarize(roi{i},t),'holes'),150);
%Save mask
    if numFiles > 1
        imwrite(mask,fullfile(savePath, 'cellmask.tif'),'compression','none');
    else
        imwrite(mask,fullfile(savePath,'cellmask.tif'),'compression','none');
    end
%Mask all other channels
    for j = 1:numCh
        images{i,j} = images{i,j}.*uint16(mask);
    end
end
```

Closes ROI drawing interface and begins loop to generate cell masks. Creates subfolder for each image and creates string **savePath**, where the mask will be saved.

Applies ROI to the image, setting all external pixels to 0; then, applies a large, n = 15 median filter to blur the image. A threshold is calculated to binarize the image.

mask is generated by filling holes and deleting objects smaller than 150 pixels. It is saved in directory **savePath**. The final mask is applied to all other image channels.

```
%THRESHOLD PARAMETER SETUP
segIndex = zeros(1,numCh);
for i = 1:numCh
    %Ask how to segment other channels
    imshow(imadjust(images{1,i},[0 0.2]));
    response = questdlg(['What is the morphology of channel ' num2str(i) '? Close to use
defaults.'],...
        'Channel segmentation workflow selection','Particle',...
        'Large organelle','Sheet','Large organelle');
    %Assign segmentation index
    switch response
        case 'Particle'
            segIndex(i) = 0;
        case 'Large organelle'
            segIndex(i) = 1;
        case 'Sheet'
            segIndex(i) = 2;
        otherwise
            if numCh == 6
                segIndex = [0 1 2 0 1 0];
            else
                segIndex = zeros(1,numCh);
            end
            break
    end
end
```

User is asked to identify channel contents. This data is stored in **segIndex**. If the interface is closed at any point, default indices will be chosen.

```
%THRESHOLD AND SAVE
close(figure(1));
ball = strel('disk',1,0);
B = ones(50,50)/(50^2);
for i = 1:numFiles
    disp(['Working on cell ' num2str(i) '...']);
    savePath = fullfile(filePath, strrep(fileList(i).name,'.tif',''));
    for j = 1:numCh
        %Particle segmentation
        if segIndex(j) == 0
        %Gaussian filter and opening
            C = uint16(conv2(images{i,j},B,'same'));
            images{i,j} = imgaussfilt(images{i,j},1.5);
            t = double(multithresh(images{i,j},2));
            t = t(2)/65535;
            images{i,j} = imbinarize(images{i,j},t);
            images{i,j} = imopen(images{i,j},ball);
        %Watershed
            D = -bwdist(~images{i,j});
            M = imextendedmin(D,2);
            D2 = imimposemin(D,M);
            L = watershed(D2,8);
            L(~images{i,j}) = 0;
            images{i,j} = bwareaopen((L>0),10);
        %Large organelle segmentation
        elseif segIndex(j) == 1
            %Median filter and opening
            images{i,j} = medfilt2(images{i,j});
            t = double(multithresh(images{i,j},2));
            t = t(2)/65535;
            images{i,j} = imbinarize(images{i,j},t);
            images{i,j} = imopen(images{i,j},ball);
        %Sheet segmentation
        else
            images{i,j} = medfilt2(images{i,j});
            t = adaptthresh(images{i,j}, 'NeighborhoodSize', 51);
            images{i,j} = imbinarize(images{i,j},t);
        end
        imwrite(images{i,j},fullfile(savePath,['ch0' num2str(j) '.tif']),'Compression','none');
    end
end
```

Closes open image and generates structuring matrices **ball** and **B**.Loops through each image and generates a **savePath**. Begins looping through each channel of the image.

Channels selected as particle-containing are subjected to a background subtraction and Gaussian filter (δ = 0.5). A global threshold is generated from the blurred image and used to binarized the channel. Small objects are eroded with imopen and watershed to split connecting components.

Channels selected as large organelle-containing are subjected to a median filter (n = 1). A global threshold is generated from the blurred image and used to binarized the channel. Small objects are eroded with imopen.

Channels selected as sheet-containing are subjected to a median filter (n = 1). An adaptive threshold is generated from each pixel's neighbors (n = 51) and used to binarize the channel.

The segmented channel image is saved to the **savePath** directory as 'ch0j', j being the current channel.

a. *rawZStackToMask*

```
clear variables
close all
commandwindow;

%MASKING PARAMETER SETUP
numCh = str2double(input('How many channels are there? \n','s'));
filePath = uigetdir('Select folder with images to mask');
addpath(filePath);
fileList = dir([filePath '/*.tif']);

%IMPORT IMAGES
numFiles = size(fileList,1);
numSlices = zeros(1,numFiles);
for i = 1:numFiles
    z = 1;
    while z > 0
        try
            j = rem(z-1,numCh)+1;
            images(:,:,i,j,ceil(z/numCh)) = imread(fileList(i).name,z);
            z = z+1;
        catch
            numSlices(i) = (z-1)/numCh;
            z = 0;
        end
    end
    disp(i);
end
```

Number of channels saved as string **numCh**. Folder location saved as string **filePath** and added to working directory. List of .tif files in file path generated as **fileList**.

images is a 5D matrix that stores X, Y, file number, channel number, and slice number.

Stacks of images are read in sequential order prioritizing slice number. All channels from slice z = 1 are stored in **images**. Z is incremented to the next slice and the process is repeated until there are no more images.

```
%MANUAL ROI SELECTION
maskCh = str2double(input(['Of the ' num2str(numCh) ' channels, which should be used for masking
the cell? \n'], 's'));
roi = cell(1,numFiles);
for i = 1:numFiles
    brightImg = imadjust(max(images(:,:,i,maskCh,:),[],5),[0 0.2]);
    imshow(brightImg);
    roi{i} = roipoly;
    if isempty(roi{i})
        try
            roi{i} = imbinarize(ones(size(images(:,:,3),1),size(images(:,:,3),2)));
        catch
            roi{i} = imbinarize(ones(size(images(:,:,1),1),size(images(:,:,1),2)));
        end
    end
end
```

User enters the channel being used for cell masking, saved as double **maskCh**. Structure **roi** created with a page for each file.

User manually draws ROI for each cell on a contrast adjusted image; ROI saved as binary mask in **roi**. If ROI drawing interface is closed, will auto-generate whole-image ROI from channel 3, or, in lieu of channel 3, from channel 1.

```
%MASK CHANNELS
close(figure(1));
maskType = str2double(input('Use selected mask channel for tighter mask? 0 for
Yes, 1 for No. Recommended only if using ER as mask channel. \n'));
for i = 1:numFiles
    mkdir(fullfile(filePath, strrep(fileList(i).name,'.tif','')));
    savePath = fullfile(filePath, strrep(fileList(i).name,'.tif',''));
%Make binary, fill holes, exclude small objects
    if maskType == 1
        mask = roi{i};
    else
        mask = max(images(:,:,i,maskCh,:),[],5);
        try
            mask = medfilt2(mask,[15 15]).*uint16(roi{i});
            disp('uint16');
        catch
            mask = medfilt2(mask,[15 15]).*uint8(roi{i});
            disp('uint8');
        end
        t = adaptthresh(mask,.9, 'NeighborhoodSize', 51);
        mask = imfill(imclose(imbinarize(mask,t),strel('disk',1,0)),'holes');
    end
%Save mask
    if numFiles > 1
        imwrite(mask,fullfile(savePath, 'cellmask.tif'),'Compression','none');
    else
        imwrite(mask,fullfile(savePath,'cellmask.tif'),'Compression','none');
    end
%Mask all other channels
    for j = 1:numCh
        for z = 1:numSlices(i)
            try
                images(:,:,i,j,z) = images(:,:,i,j,z).*uint16(mask);
            catch
                images(:,:,i,j,z) = images(:,:,i,j,z).*uint8(mask);
            end
        end
    end
end
```

Closes ROI drawing interface and begins loop to generate cell masks. Asks for user input on whether to use the drawn ROI as a loose mask, or to threshold the ER and use it as a tighter mask. Creates subfolder for each image and creates string **savePath**, where the mask will be saved.

Applies ROI to the image, setting all external pixels to 0; if a tight mask is desired, a max intensity projection of the ER is created. A large, n = 15 median filter is applied to blur the ER; following, an adaptive threshold is applied to each pixel's neighbors within 51 pixels.

mask is generated by closing gaps and filling holes in the resulting image. It is saved in directory **savePath**. The final mask is applied to all image channels of all other slices.

```
%THRESHOLD PARAMETER SETUP
segIndex = zeros(1,numCh);
for i = 1:numCh
    %Ask how to segment other channels
    imshow(imadjust(images{1,i},[0 0.2]));
    response = questdlg(['What is the morphology of channel ' num2str(i) '? Close to use 
defaults.'],...
        'Channel segmentation workflow selection','Particle',...
        'Large organelle','Sheet','Large organelle');
    %Assign segmentation index
    switch response
        case 'Particle'
            segIndex(i) = 0;
        case 'Large organelle'
            segIndex(i) = 1;
        case 'Sheet'
            segIndex(i) = 2;
        otherwise
            if numCh == 6
                segIndex = [0 1 2 0 1 0];
            else
                segIndex = zeros(1,numCh);
            end
            break
    end
end
```

User is asked to identify channel contents. This data is stored in **segIndex**. If the interface is closed at any point, default indices will be chosen.

```
%THRESHOLD AND SAVE
close(figure(1));
ball = strel('disk',1,0);
B = ones(50,50)/(50^2);
mipImages = cell(numFiles,numCh);
tf = isa(mask,'uint8');
if tf == 1
    div = 255;
else
    div = 65535;
end
```

Closes open image and generates structuring matrix in the shape of a radius-1 circle. Also creates a 50 x 50 matrix for background subtraction and **mipImages** for storing max intensity projections.

Sets divisor for thresholding **div** based on image type **tf**.

```
for i = 1:numFiles
    disp(['Working on cell ' num2str(i) '...']);
    savePath = fullfile(filePath, strrep(fileList(i).name,'.tif',''));
    for j = 1:numCh
            mipImages{i,j} = max(images(:,:,i,j,:),[],5);
            if tf == 1
                C = uint8(conv2(mipImages{i,j},B,'same'));
            else
                C = uint16(conv2(mipImages{i,j},B,'same'));
            end
%Particle segmentation
            if segIndex(j) == 0
%Gaussian filter and opening
                mipImages{i,j} = imgaussfilt(imsubtract(mipImages{i,j},C));
                t = double(multithresh(mipImages{i,j},2));
                t = t(2)/div;
                mipImages{i,j} = imbinarize(mipImages{i,j},t);
                mipImages{i,j} = imopen(mipImages{i,j},'holes');
%Watershed
                D = -bwdist(~mipImages{i,j});
                M = imextendedmin(D,2);
                D2 = imimposemin(D,M);
                L = watershed(D2,8);
                L(~mipImages{i,j}) = 0;
                mipImages{i,j} = bwareaopen((L>0,10);
%Large organelle segmentation
            elseif segIndex(j) == 1
                %Median filter and opening
                mipImages{i,j} = medfilt2(imsubtract(mipImages{i,j},C));
                t = double(multithresh(mipImages{i,j},2));
                t = t(2)/div;
                mipImages{i,j} = imbinarize(mipImages{i,j},t);
                mipImages{i,j} = imopen(mipImages{i,j},ball);
%Sheet segmentation
            else
                mipImages{i,j} = medfilt2(imsubtract(mipImages{i,j},C));
                t = adaptthresh(mipImages{i,j}, 'NeighborhoodSize', 51);
                mipImages{i,j} = imbinarize(mipImages{i,j},t);
            end

        imwrite(mipImages{i,j},fullfile(savePath,['ch0' num2str(j)'.tif']),'Compression','none');
    end
end
```

Loops through each image and generates a **savePath**. Loops through each channel and generates max intensity projections along the Z axis. Also generates background image **C** to be subtracted from each MIP.

Channels selected as particle-containing are subjected to a Gaussian filter (δ = 0.5). A global threshold is generated from the blurred image and used to binarized the channel. Small objects are eroded with imopen and watershed to split connecting objects.

Channels selected as large organelle-containing are subjected to a median filter (n = 1). A global threshold is generated from the blurred image and used to binarized the channel. Small objects are eroded with imopen.

Channels selected as sheet-containing are subjected to a median filter (n = 1). An adaptive threshold is generated from each pixel's neighbors (n = 51) and used to binarize the channel.

The segmented max intensity projection for the channel is saved to **savePath** as 'ch0j', j being the current channel.

c. *rawTimeLapseToMask*

```
clear variables
close all
commandwindow;

%MASKING PARAMETER SETUP
numCh = str2double(input('How many channels are there? \n','s'));
numFrames = str2double(input('How many frames are there? \n','s'));
filePath = uigetdir('Select folder with images to mask');
addpath(filePath);
fileList = dir([filePath '/*.tif']);
%IMPORT IMAGES
numFiles = size(fileList,1);
images = cell(numFiles,numCh);
for i = 1:numFiles
    k = 0;
    h = 1;
    for j = 1:numCh*numFrames
        k = k+1;
        images{i,k,h} = imread(fileList(i).name,j);
        if rem(k,numCh) == 0
            k = 0;
            h = h+1;
        end
    end
end
```

Number of channels saved as string **numCh**. Folder location saved as string **filePath** and added to working directory. List of .tif files in file path generated as **fileList**.

images is a 3D matrix that stores X, Y, and frame number.

Stacks of images are read in sequential order prioritizing frame number h. All channels from frame h = 1 are stored in **images** and k is incremented. At the last channel k is reset to 0 and h is incremented.

```
%MANUAL ROI SELECTION
maskCh = str2double(input(['Of the ' num2str(numCh) ' channels, which should be used for masking
the cell? \n'], 's'));
roi = cell(1,numFiles);
for i = 1:numFiles
    brightImg = imadjust(images{i,maskCh,1},[0 0.2]);
    imshow(brightImg);
    roi{i} = roipoly;
    if isempty(roi{i})
        try
            roi{i} = imbinarize(ones(size(images{1,3,1},1),size(images{1,3,1},2)));
        catch
            roi{i} = imbinarize(ones(size(images{1,1,1},1),size(images{1,1,1},2)));
        end

    end
end
```

User enters the channel being used for cell masking, saved as double **maskCh**. Structure **roi** created with a page for each file.

User manually draws ROI on a contrast adjusted image of frame 1; ROI saved as binary mask in **roi**. If ROI drawing interface is closed, will auto-generate whole-image ROI from channel 3, or, in lieu of channel 3, from channel 1.

```
%MASK CHANNELS
close(figure(1));
for i = 1:numFiles
    mkdir(fullfile(filePath, strrep(fileList(i).name,'.tif','')));
    savePath = fullfile(filePath, strrep(fileList(i).name,'.tif',''));
%Apply ROI to image and blur
    roi{i} = immultiply(roi{i},images{i,maskCh,1});
    roi{i} = medfilt2(roi{i}, [15 15]);
%Calculate threshold from blurred image
    t = double(multithresh(roi{i},3));
    t = t(1)/65535;
%Make binary, fill holes, exclude small objects
    mask = bwareaopen(imfill(imbinarize(roi{i},t),'holes'),150);
%Save mask
    if numFiles > 1
        imwrite(mask,fullfile(savePath, 'cellmask.tif','compression','none'));
    else
        imwrite(mask,fullfile(savePath,'cellmask.tif', 'compression','none'));
    end
%Mask all other channels
    for j = 1:size(images,2)
        for k = 1:size(images,3)
            images{i,j,k} = images{i,j,k}.*uint16(mask);
        end
    end
end
```

Closes ROI drawing interface and begins loop to generate cell masks. Creates subfolder for each image and creates string **savePath**, where the mask will be saved.

Applies ROI to current frame, setting all external pixels to 0; then, applies a large, n = 15 median filter to blur the image. A threshold is calculated to binarize the image.

mask is generated by filling holes and deleting objects smaller than 150 pixels. It is saved in directory **savePath**. The final mask is applied to all channels of every frame.

```
%THRESHOLD PARAMETER SETUP
segIndex = zeros(1,numCh);
for i = 1:numCh
%Ask how to segment other channels
    imshow(imadjust(images{1,i,1},[0 0.2]));
    response = questdlg(['What is the morphology of channel ' num2str(i) '? Close to use
       defaults.'],'Channel segmentation workflow selection','Particle', 'Large
       organelle','Sheet','Large organelle');
%Assign segmentation index
    switch response
        case 'Particle'
            segIndex(i) = 0;
        case 'Large organelle'
            segIndex(i) = 1;
        case 'Sheet'
            segIndex(i) = 2;
        otherwise
            if numCh == 6
                segIndex = [0 1 2 0 1 0];
            else
                segIndex = zeros(1,numCh);
            end
            break
    end
end
```

User is asked to identify channel contents. This data is stored in **segIndex**. If the interface is closed at any point, default indices will be chosen.

```matlab
%THRESHOLD AND SAVE
close(figure(1));
ball = strel('disk',1,0);
for i = 1:numFiles
    disp(['Working on cell ' num2str(i) '...']);
    savePath = fullfile(filePath, strrep(fileList(i).name,'.tif',''));
    for j = 1:size(images,2)
        for k = 1:size(images,3)
%Particle segmentation
            if segIndex(j) == 0
        %Gaussian filter and opening
                images{i,j,k} = imgaussfilt(images{i,j,k});
                t = double(multithresh(images{i,j,k},2));
                t = t(2)/65535;
                images{i,j,k} = imbinarize(images{i,j,k},t);
                images{i,j,k} = imopen(images{i,j,k},ball);
        %Watershed
                D = -bwdist(~images{i,j,k});
                M = imextendedmin(D,2);
                D2 = imimposemin(D,M);
                L = watershed(D2,8);
                L(~images{i,j,k}) = 0;
                images{i,j,k} = bwareaopen((L>0),10);
%Large organelle segmentation
            elseif segIndex(j) == 1
        %Median filter and opening
                images{i,j,k} = medfilt2(images{i,j,k});
                t = double(multithresh(images{i,j,k},2));
                t = t(2)/65535;
                images{i,j,k} = imbinarize(images{i,j,k},t);
                images{i,j,k} = imopen(images{i,j,k},ball);
%Sheet segmentation
            else
                images{i,j,k} = medfilt2(images{i,j,k});
                t = adaptthresh(images{i,j,k}, 'NeighborhoodSize', 51);
                images{i,j,k} = imbinarize(images{i,j,k},t);
            end
%Save image
            if k == 1
            imwrite(images{i,j,k},...
                fullfile(savePath,['ch0' num2str(j) '.tif']),'Compression','none');
            else
                imwrite(images{i,j,k},fullfile(savePath,['ch0' num2str(j) '.tif']),...
                    'Compression','none','WriteMode','append');
            end
        end
    end
end
```

Closes open image and generates structuring matrix in the shape of a radius-1 circle. Loops through each image and generates a **savePath**. Begins looping through each frame of each channel.

Channels selected as particle-containing are subjected to a Gaussian filter (δ = 0.5). A global threshold is generated from the blurred image and used to binarized the channel. Small objects are eroded with imopen and watershed to split connecting components.

Channels selected as large organelle-containing are subjected to a median filter (n = 1). A global threshold is generated from the blurred image and used to binarized the channel. Small objects are eroded with imopen.

Channels selected as sheet-containing are subjected to a median filter (n = 1). An adaptive threshold is generated from each pixel's neighbors (n = 51) and used to binarize the channel. The segmented channel image is saved to the **savePath** directory as 'ch0j', j being the current channel.

d. *maskToCounts*

```
clear variables
close all
commandwindow;

%FOLDER SELECTION
folderList = uigetdir2(0,'Select folders to compute');
addpath(folderList{:});
numFolders = size(folderList,2);
numCh = size(dir([folderList{1} '/*ch*.tif']),1);
images = cell(numFolders,numCh+1);
pixSize = str2double(input('How many pixels per micron? \n','s'))^2;
%READ IN IMAGES
for i = 1:numFolders
    fileList = dir([folderList{i} '/*.tif']);
    for j = 1:numCh+1
            images{i,j} = imread(fullfile(folderList{i},fileList(j).name));
    end
end
```

Selected folders saved as cell **folderList** and added to working directory. Number of folders saved as integer **numFolders**. Number of channels saved as string **numCh**. Pixels per micron saved as pixSize. Images read in and stored in **images**.

```
orgIndex = zeros(1,numCh+1); orgIndex(1) = -1;
orgNames = cell(1,numCh+1); orgNames{1} = 'mask';

%ORGANELLE DESIGNATION
for i = 2:numCh+1
    %Ask how to segment other channels
    %Skip mask in slot 1, assign -1
    imshow(images{1,i});
    if isempty(orgNames{i}) == 1
        try
            orgNames(i) = inputdlg('What will you name this channel? Close to use defaults.');
        catch
            if numCh == 6
                orgIndex = [-1 0 1 2 0 1 0];
                orgNames = {'mask','lysosome','mitochondria','ER','peroxisome','Golgi','lipid
droplet'};
            else
                orgIndex = zeros(1,numCh);
            end
            break
        end
    end
    response = questdlg(['What is the morphology of channel ' num2str(i-1) '?'],...
        'Channel segmentation workflow selection','Particle',...
        'Large organelle','Sheet','Large organelle');
    %Assign segmentation index
    switch response
        case 'Particle'
            orgIndex(i) = 0;
        case 'Large organelle'
            orgIndex(i) = 1;
        case 'Sheet'
            orgIndex(i) = 2;
    end
end
close(figure(1));
```

Stores organelle names and masking parameter indices in **orgNames** and **orgIndex**. Close during name entry dialog to use default multispectral options. Everything is shifted up one index and the mask is stored in position {1}.

```
%GENERATE RESULT MATRIX
numParam = zeros(1,4);
for i = -1:2
    numParam(i+2) = sum(orgIndex == i);
end
numParam = sum(numParam.*[1 4 4 2]);
results = cell(numParam+1, numFolders+1);
index = 2;
for i = 1:length(orgIndex)
    if orgIndex(i) == -1
        results{index,1} = 'Surface area';
        index = index+1;
    elseif orgIndex(i) == 2
        results{index,1} = [orgNames{i} ' area'];
        results{index+1,1} = [orgNames{i} ' area fraction'];
        index = index+2;
    else
        results{index,1} = [orgNames{i} ' number'];
        results{index+1,1} = [orgNames{i} ' mean area'];
        results{index+2,1} = [orgNames{i} ' median area'];
        results{index+3,1} = [orgNames{i} ' area fraction'];
        index = index+4;
    end
end
for i = 2:size(results,2)
    results{1,i} = fliplr(strtok(fliplr(folderList{i-1}),'\'));
end
```

Generates a table of results with n number of rows for each channel. Rows n is determined by the organelle parameter index of the channel. Labels each column with the original file name.

```
%COMPUTATION
for i = 2:numFolders+1       %new loop every cell
    index = 2;
    for j = 1:numCh+1              %new loop every channel
        if orgIndex(j) == -1    %mask
            results{index,i} = bwarea(images{i-1,j})/pixSize;
            index = index+1;
        elseif orgIndex(j) == 2 %sheet
            results{index,i} = bwarea(images{i-1,j})/pixSize;
            if sum(orgIndex == -1) == 1
                maskIndx = find(orgIndex==-1);
                results{index+1,i} = bwarea(images{i-1,j})/bwarea(images{i-1,maskIndx});
            else
                results{index+1,i} = 'Cannot compute without mask';
            end
            index = index+2;
        else                    %particles and large organelles
            bw = bwconncomp (images{i-1,j});
            results{index,i} = bw.NumObjects;
            areas = zeros(1,bw.NumObjects);
            for k = 1:bw.NumObjects
                areas(k) = length(bw.PixelIdxList{k});
            end
            results{index+1,i} = mean(areas)/pixSize;
            results{index+2,i} = median(areas)/pixSize;
            if sum(orgIndex == -1) == 1
                maskIndx = find(orgIndex==-1);
                results{index+3,i} = bwarea(images{i-1,j})/bwarea(images{i-1,maskIndx});
            else
                results{index+3,i} = 'Cannot compute without mask';
            end
            index = index+4;
        end
    end
end
```

Calculates cell surface area from the cell mask and converts to micron units.

For sheets, calculates area and area fraction and converts to micron units.

For particles and large organelles, calculates total object area, area fraction, and mean/median object area from labeled map generated by bwconncomp.

```
%SAVE RESULT
writetable(cell2table(results),[fliplr(strrep(fliplr(...
    folderList{1}),strtok(fliplr(folderList{1}),'\'),'')) 'organelle counts.csv'],...
    'WriteVariableNames',0);
disp('Complete. Results salved in parent folder.');
```

Saves results in parent folder as organellecounts.csv.

```
%MULTIFOLDER SELECT FUNCTION
```

Details here.

e. *maskTimeLapseToCounts*

```
clear variables
close all
commandwindow;
%FOLDER SELECTION
numFrames = str2double(input('How many frames are there? \n','s'));
folderList = uigetdir2(0,'Select folders to compute');
addpath(folderList{:});
numFolders = size(folderList,2);
numCh = size(dir([folderList{1} '/*ch*.tif']),1);
images = cell(numFolders,numCh+1);
pixSize = str2double(input('How many pixels per micron? \n','s'))^2;
%READ IN IMAGES
for i = 1:numFolders
    fileList = dir([folderList{i} '/*.tif']);
    for j = 1:numCh+1
        if j ==1
            images{i,j,1} = imread(fullfile(folderList{i},fileList(j).name));
        else
            for k = 1:numFrames
                images{i,j,k} = imread(fullfile(folderList{i},fileList(j).name),k);
            end
        end
    end
end
```

Number of frames saved as integer **numFrames**. Selected folders saved as cell **folderList** and added to working directory. Number of folders saved as integer **numFolders**. Number of channels saved as string **numCh**. Pixels per micron saved as double **pixSize**. Images read in and stored in **images**.

```
orgIndex = zeros(1,numCh+1); orgIndex(1) = -1; orgNames = cell(1,numCh+1); orgNames{1} = 'mask';
%ORGANELLE DESIGNATION
for i = 2:numCh+1
    imshow(images{1,i});
    if isempty(orgNames{i}) == 1
        try
            orgNames(i) = inputdlg('What will you name this channel? Close to use defaults.');
        catch
            if numCh == 6
                orgIndex = [-1 0 1 2 0 1 0];
                orgNames = {'mask','lysosome','mitochondria','ER','peroxisome','Golgi','lipid
droplet'};
            else
                orgIndex = zeros(1,numCh);
            end
            break
        end
    end
    response = questdlg(['What is the morphology of channel ' num2str(i-1) '?'],...
        'Channel segmentation workflow selection','Particle',...
        'Large organelle','Sheet','Large organelle');
    switch response
        case 'Particle'
            orgIndex(i) = 0;
        case 'Large organelle'
            orgIndex(i) = 1;
        case 'Sheet'
            orgIndex(i) = 2;
    end
end
close(figure(1));
```

Stores organelle names and masking parameter indices in **orgNames** and **orgIndex**. Close during name entry dialog to use default multispectral options. Everything is shifted up one index and the mask is stored in position {1}.

```
%GENERATE RESULT MATRIX
numParam = zeros(1,4);
for i = -1:2
    numParam(i+2) = sum(orgIndex == i);
end
numParam = sum(numParam.*[1 4 4 2]);
results = cell(numFolders,numParam+1,numFrames+1);
for i = 1:size(results,1)
    index = 2;
    results{i,1,1} = fliplr(strtok(fliplr(folderList{i}),'\'));
    for j = 1:length(orgIndex)
        if orgIndex(j) == -1
            results{i,index,1} = 'Surface area';
            index = index+1;
        elseif orgIndex(j) == 2
            results{i,index,1} = [orgNames{j} ' area'];
            results{i,index+1,1} = [orgNames{j} ' area fraction'];
            index = index+2;
        else
            results{i,index,1} = [orgNames{j} ' number'];
            results{i,index+1,1} = [orgNames{j} ' mean area'];
            results{i,index+2,1} = [orgNames{j} ' median area'];
            results{i,index+3,1} = [orgNames{j} ' area fraction'];
            index = index+4;
        end
    end

    for j = 2:size(results,3)
        results{i,1,j} = ['frame ' num2str(j-1)];
    end
end
```

Generates a table of results with n number of rows for each channel. Rows n is determined by the organelle parameter index of the channel. Labels each column with the frame number.

```
% COMPUTATION
for i = 1:numFolders       %new loop every cell
    index = 2;
    for j = 1:numCh+1              %new loop every channel
        for k = 2:numFrames+1
            if orgIndex(j) == -1 && k == 2    %mask
                results{i,index,k} = bwarea(images{i,j,k-1})/pixSize;
            elseif orgIndex(j) == 2 %sheet
                results{i,index,k} = bwarea(images{i,j,k-1})/pixSize;
                if sum(orgIndex == -1) == 1
                    maskIndx = find(orgIndex==-1);
                    results{i,index+1,k} = bwarea(images{i,j,k-1})/bwarea(images{i,maskIndx,1});
                else
                    results{i,index+1,k} = 'Cannot compute without mask';
                end
            else                        %particles and large organelles
                bw = bwconncomp (images{i,j,k-1});
                results{i,index,k} = bw.NumObjects;
                areas = zeros(1,bw.NumObjects);
                for l = 1:bw.NumObjects
                    areas(l) = length(bw.PixelIdxList{l});
                end
                results{i,index+1,k} = mean(areas)/pixSize;
                results{i,index+2,k} = median(areas)/pixSize;
                if sum(orgIndex == -1) == 1
                    maskIndx = find(orgIndex==-1);
                    results{i,index+3,k} = bwarea(images{i,j,k-1})/bwarea(images{i,maskIndx,1});
                else
                    results{i,index+3,k} = 'Cannot compute without mask';
                end
            end
        end
        if orgIndex(j) == -1
            index = index+1;
        elseif orgIndex(j) == 2
            index = index+2;
        else
            index = index+4;
        end
    end
end
```

Loops through each frame of each channel of each cell. Calculates cell surface area from the cell mask and converts to micron units.

For sheets, calculates area and area fraction and converts to micron units.

For particles and large organelles, calculates total object area, area fraction, and mean/median object area from labeled map generated by bwconncomp.

```
%SAVE RESULT
for i = 1:numFolders
    writetable(cell2table(squeeze(results(i,:,:))),[fliplr(strrep(fliplr(...
        folderList{1}),strtok(fliplr(folderList{1}),'\'),'')) 'timelapse ' num2str(i) ' organelle
counts.csv'], 'WriteVariableNames',0);
end
disp('Complete. Results salved in parent folder.');
```

Saves results in parent folder as organellecounts.csv.

```
%MULTIFOLDER SELECT FUNCTION
```

Details here.

f. *maskToContacts*

```
clear variables
close all
commandwindow;

%FOLDER SELECTION
folderList = uigetdir2(0,'Select folders to compute');
addpath(folderList{:});
numFolders = size(folderList,2);

%WARNING: uigetfile has issues with long file names. See comment in line 3.
chList = uigetfile(fullfile(folderList{1},'/*ch*.tif'),...
    'Select channels to compute contacts for.','Multiselect','on');
numCh = size(chList,2);
chListNum = zeros(1,numCh);
for i = 1:numCh
    %searches for integers in channel i and returns their location(s)
    temp = regexp(chList{i},'\d');
    %extremely bad section heavily dependent on filename
    %literally: temp looks something like [3 4]
    %takes characters 3 and 4 from chList{i}, which looks like 'ch04.tif'
    %returns '04' --str2double--> 4
    chListNum(i) = str2double(chList{i}(temp(1):temp(2)));
end
fileList = dir([folderList{1} '/*ch*.tif']);
images = cell(1,numCh);
```

Selected folders saved as cell **folderList** and added to working directory. Number of folders saved as integer **numFolders**. Number of channels saved as integer **numCh**. Selected channels stored in **chListNum**.

```
%NAME ASSIGNMENT
defaultNames = cell(1,numCh);
for i = 1:numCh
    realCh = strrep(fliplr(strtok(fliplr(chList{i}),'_')),'.tif','');
    switch realCh
        case 'ch01'
            defaultNames{i} = ' lysosome ';
        case 'ch02'
            defaultNames{i} = ' mitochondria ';
        case 'ch03'
            defaultNames{i} = ' ER ';
        case 'ch04'
            defaultNames{i} = ' peroxisome ';
        case 'ch05'
            defaultNames{i} =  ' Golgi ';
        case 'ch06'
            defaultNames{i} =  ' lipid droplet ';
        otherwise
            defaultNames{i} = ' organelle ';
    end
end
```

Generates array of default channel names if user does not enter their own.

```
orgNames = cell(1,numCh);
for i = 1:numCh
    imshow(imread(fullfile(folderList{1},chList{i})));
    response = inputdlg(...
        ['What will you name this channel? Close at any point to use default array of'...
        defaultNames{:}]);
    if isempty(response) == 0
        orgNames{i} = response{1};
    else
        orgNames = defaultNames;
        break;
    end
end
close(figure(1));
```

Asks user to input organelle names; if exited, will use **defaultNames** instead.

```
%DEGREE OF CONTACT SELECTION
degList = 2:numCh;
opts = cell(1,length(degList));
for i = 1:length(degList)
    opts{i} = num2str(degList(i));
end
opts = {cat(1,opts{:})};
[indx,tf] = listdlg('Name','Contact degree','PromptString','How many degrees of contact?',...
    'ListString',opts,'SelectionMode','single','ListSize',[175 75],...
    'OKString','Apply','CancelString','Binary only');
if isempty(indx) == 1 || tf == 0
    contactDegree = 2;
else
    contactDegree = degList(indx);
end
```

Asks user to select degree of contact analysis. Higher degrees increase run time. Stores user selection as integer **contactDegree**; default value is 2, or binary.

```
%CONTACT ANALYSIS
results = {''};
for i = 1:size(folderList,2)
    disp(['Working on cell ' num2str(i)]);
    index = 2;                    %Tracks current row of result matrix
    if i == 1
        results{1,1} = 'cell 1';
    else
        results{1,end+1} = ['cell ' num2str(i)];
    end
    results{1,end+1} =  'number';
    results{1,end+1} = 'number fraction';
    results{1,end+1} = 'area fraction';
    for j = 1:numCh
        fileName = fullfile(folderList{i},fileList(chListNum(j)).name);
        images{j} = imread(fileName);
    end
    for j = 2:contactDegree
        allPerms = npermutek(1:numCh,j);    %do not change
        for k = size(allPerms,1):-1:1   %count down
            if length(unique(allPerms(k,:))) ~= size(allPerms,2) == 1
                allPerms(k,:) = [];
            end
        end
        if j>2
            uniquePerms = unique(sort(allPerms(:,2:end),2),'rows');
            c = 1;
            for k = 1:numCh
                for l = 1:size(uniquePerms,1)
                    if ismember(k,uniquePerms(l,:)) == 0
                        if c == 1
                            usePerms = [k uniquePerms(l,:)];
                            c = c+1;
                        else
                            usePerms(c,:) = [k uniquePerms(l,:)];
                            c = c+1;
                        end
                    end
                end
            end
        else
            usePerms = allPerms;
        end
        for k = 1:size(usePerms,1)
            [results{index,(i-1)*4+1}, results{index,(i-1)*4+2}, results{index,(i-1)*4+3},
results{index,(i-1)*4+4}] = ...
                computeContacts(images,orgNames,usePerms(k,:));
            index = index+1;
        end

    end
end
```

Loops through every cell folder to read selected channels. Starting at degree 2, calls npermutek to compute permutations. Starting from the back of the matrix, deletes any permutations where the number of elements differs from the current contact degree.

For contact degrees above 2, sorts permutations by their first element and eliminates duplicate combinations of other elements. Loops through the remaining permutations and deletes sets where the first element is duplicated in the other elements. For degree 2 contacts this is skipped.

Generates a result matrix of appropriate size. Uses the remaining permutations **usePerms** along with **images** and **orgNames** to call computeContacts, which computes number, number fraction, and area fraction of each permutation.

```
%SAVE RESULTS
writetable(cell2table(results),[fliplr(strrep(fliplr(...
    folderList{1}),strtok(fliplr(folderList{1}),'\'),'')) 'organelle interactome.csv'],...
    'WriteVariableNames',0);
disp('Complete. Results salved in parent folder.');
```

Saves results in parent folder as organellecontacts.csv.

```
%CONTACT COMPUTATION FUNCTION
function [name, number, nFrac, aFrac] = computeContacts(images, names, indices)
    name = [names{indices(1)} ':' names{indices(2:end)}];
    baseArea = bwarea(~images{indices(1)});
    imageBottom = ones(size(images{1},1),size(images{1},2));
    for n = 2:length(indices)
        imageBottom = imageBottom & images{indices(n)};
    end
    imageContact = bwconncomp(imdilate(images{indices(1)},strel('disk',1))...
        & imageBottom,4);
    baseNumber = max(max(bwlabel(imdilate(images{indices(1)},strel('disk',1)),4)));
    number = imageContact.NumObjects;
    cc = bwlabel(imdilate(images{indices(1)},strel('disk',1)),4);
    aaPix = [imageContact.PixelIdxList];
    nFrac = 0;
    aFrac = 0;
    if ~isempty(aaPix)
        aaIdxList = 0;
        for z = 1:length(aaPix)
            aaIdxList(z) = aaPix{z}(1);
          end
        bb = cc(aaIdxList);
        bb = unique(bb);
        nFrac = double(length(bb)/baseNumber);
        aFrac = bwarea(~images{indices(1)} & imageBottom)/baseArea;
    end
end
```

Computes **baseArea** from channel of interest. Generates **imageBottom** from all channels being computed except the channel of interest. Generates **imageContact** from the overlap between imageBottom and 1-pixel dilated channel of interest.

Creates labeled image from bwlabel and stores object number in integer **baseNumber**. The same is done for the dilated image. Each pixel in imageContact is mapped back to the bwlabel image; the subsequent list is reduced down to its unique elements.

The list of unique object contacts is divided by baseNumber to calculate **nFrac**. The total non-dilated object area of the overlap between imageContact and the channel of interest is divided by baseArea to calculate **aFrac**.

```
%PERMUTATION FUNCTION
```

Details here.

```
%MULTIFOLDER SELECT FUNCTION
```

Details here.