% Convert image to pixel segments

[segmentsPix,xLimPix,yLimPix] = imageToPixelSegments(img);

| function [segmentsPix,xLimPix,yLimPix] = imageToPixelSegments(img)  % Extract line traces from the image  img2 = ~imbinarize(rgb2gray(img), 'adaptive' , 'ForegroundPolarity' , 'dark' );  img3 = bwmorph(img2, 'clean' );  img4 = bwmorph(img3, 'thin' ,inf);  % Extract pixels in order  coordsPix = getCoords(img4);   | function curvePoints = getCoords(shapeImage)  % Call the function `bwboundaries` to obtain the boundaries of each unique region.  [curves,~,N] = bwboundaries(shapeImage);  curves = curves(1:N); % Ignore hole boundaries  % Get the pixel points from the boundary detected  curvePoints = cell2mat(curves);  % Remove all duplicate points from the curve  curvePoints = unique(curvePoints,'rows','stable');  % In the image you're analyzing, remove all the pixels that were captured  % by `bwboundaries`. There may still be pixels remaining in the image.  % Remove curves from the image  curveInd = sub2ind(size(shapeImage),curvePoints(:,1),curvePoints(:,2));  shapeImage(curveInd) = 0;  % Go through the image to check for any remaining pixels.  % If no pixels remain, return the `curvePoints` found.  % If there are more pixels, then call this function again and  % append `curvePoints` to the additional pixels found by the recursive call.  % Call getCoords recursively if there are other curves remaining  if any(shapeImage(:))  curvePoints = [curvePoints; getCoords(shapeImage)];  end | | --- |   % Break coordinates list into contiguous segments  segmentsPix = coords2segments(coordsPix);   | function segmentsPix = coords2segments(coordsPix)  % First, find the locations where consecutive pixels are not adjacent.  % Create a breaks variable that is a logical index with value 1 where there is a break and 0 where pixels are adjacent.  consecutiveDistance = abs(diff(coordsPix));  breaks = any(consecutiveDistance > [1 1],2);  % Use the breaks variable to construct the start and end indices of each segment.  % Then loop through the segments and create a cell array segmentsPix, where each cell contains all the pixel coordinates for that segment.  % Build cell array of each segment of adjacent pixel coordinates  numSegments = sum(breaks)+1;  segmentsPix = cell(numSegments,1);  breakInds = [0; find(breaks); size(coordsPix,1)];  for ii = 1:numSegments  segmentsPix{ii} = coordsPix(breakInds(ii)+1:breakInds(ii+1),:);  end | | --- |   % Clean data and merge connected segments  segmentsPix = connectSegments(segmentsPix);   | function segments = connectSegments(segments)  % If a segment intersects itself, such as in the letter "O,"  % it will end at a point that is adjacent to another pixel in that segment.  % This will create a small gap when drawn on the whiteboard.  % To correct this, add the final adjacent pixel to the end of that segment so the line is closed.  % This section of code looks at each segment, finds any self-intersections, and closes them.  % p1 is the first point in a segment. pN is the last point in the segment.  % Add any points near p1 to the beginning of the segment.  % Add any points near pN to the end of the segment.  for ii = 1:length(segments)  points = segments{ii};  p1 = points(1,:);  pN = points(end,:);  % Add any points near p1 to beginning of segment ii  nearP1 = isadjacent(p1,points(4:end,:)); %Don't check first 3 points  if any(nearP1)  idx = find(nearP1,1)+3;  points = [points(idx,:); points]; %#ok<\*AGROW>  end  % Add any points near pN to end of segment ii  nearPN = isadjacent(pN,points(1:end-3,:)); %Don't check last 3 points  if any(nearPN)  idx = find(nearPN,1);  points = [points; points(idx,:)];  end  segments{ii} = points;  end  % This code merges any segments with endpoints that are adjacent. It looks  % at every pair of segments ii and jj. Take the first and last point of  % each segment in the pair (four points total), and check whether any are  % adjacent. If any adjacencies are found, replace the two segments with a  % single segment describing a continuous path through both segments.  for ii = 1:length(segments)-1  jj = ii + 1;  % Check all combinations of 2 segments ii and jj  while jj <= length(segments)  points\_i = segments{ii};  points\_j = segments{jj};  pi1 = points\_i(1,:);  piN = points\_i(end,:);  pj1 = points\_j(1,:);  pjN = points\_j(end,:);  % Compare points 1 and N from segments ii and jj  if isadjacent(pi1,pj1)  segments{ii} = [flipud(points\_j); points\_i];  segments(jj) = [];  elseif isadjacent(pi1,pjN)  segments{ii} = [points\_j; points\_i];  segments(jj) = [];  elseif isadjacent(piN,pj1)  segments{ii} = [points\_i; points\_j];  segments(jj) = [];  elseif isadjacent(piN,pjN)  segments{ii} = [points\_i; flipud(points\_j)];  segments(jj) = [];  end  jj = jj + 1;  end  end  end  % The sub-function isadjacent creates a convenient way to check whether two  % points are adjacent. As a sub-function, it can only be called within the  % connectSegments function.  function tf = isadjacent(p1,p2)  tf = all(abs(p1-p2) <= [1 1],2);  end | | --- |   % Store x and y pixel limits  xLimPix = [min(coordsPix(:,2)) max(coordsPix(:,2))];  yLimPix = [min(coordsPix(:,1)) max(coordsPix(:,1))]; |
| --- | --- | --- | --- |