```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Name: Robert Jauquet
% Class: COMP3085
% Assignment: Labs7
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Description: Read a sequence of facial images while taking user input on one of
% the images to describe how the face will move around the screne. Compute
% how simmilar each image is, and create a graph of the images. Then, using
% optical flow, create a video using the images as frames of the face
% moving around the screne as close to the desired path as possible.
%
% Arguments:
%
%   path: path of the files
%   prefix: prefix of the filename
%   first: first frame
%   last: last frame
%   digits: number of digits of the frame number
%   suffix: suffix of the filename
%   start: the image to be used as the starting point
%
% This creates the following outputs:
%    video.avi          -   the created sequence
%    actual_path.jpg    -   the path taken
%

function [] = Labs7(path, prefix, first, last, digits, suffix, start)

    load_sequence(path, prefix, first, last, digits, suffix);

    % color information stored in imgsC for use at the end
    imgsC=ans;
    [x y z number] = size(imgsC);

    % create a grayscale double sequence of images
    for i=1:number
        imgs(:,:,i) = double(rgb2gray(imgsC(:,:,:,i)))/255;
    end

    % denotes which image to start with.
    firstNode= start;

    % get user input on gray image. The rest of the computations are done
    % on the converted grayscale image sequence.
    userPoints = getInput(imgs(:,:,firstNode));
    % actualPoints gives the adjusted points used in the process
    actualPoints = userPoints;

    % compute the similarity of the images, and then create a sparse matrix
    S=similarity(imgs);
    S=sparse(S);
```

```
% matrix is of the form (vx,vy,opticalflowimage,im2,im1) where im1
% gives the list of all possible images, and im2 gives the images
% connected to each im1
[x,y,z] = size(imgs);
j=0;
for k=1:z
    % for each node, create a list of nodes it can reach, remove any
    % selfe reference, and sort the list.
    nodes = graphtraverse(S, k,'Depth', 1);
    nodes = nodes(2:end);
    nodes = sort(nodes);
    for m=1:length(nodes)
        % the following if-else statement assumes that the flowImage
        % will not be used, and that the flow from an image X to an
        % image Y is equal in magnitude in both x and y directions as
        % image Y to image X, but in the opposite direction. Actual
        % data recieved from the opticalFlow function dictates that the
        % output is not the same, but is similar. Because the main
        % computational time is spent on the opticalFlow, in the
        % interest of time I felt that this was a safe assumtion.
        j=j+1;
       if k>nodes(m)
           % get the index of the node(m) to k flow to be inverted for
           % the k to node(m) flow
            index = flowNodes(nodes(m),k);
            VX(j,:,:) = VX(index,:,:)*(-1);
            VY(j,:,:) = VY(index,:,:)*(-1);
        %   flowIms is not used in the current algorithm, but might be
        %   useful for improvements to the output sequence.
        %   flowIms(:,:,j) = flowIms(:,:,index);
            flowNodes(k,nodes(m))=j;
       else
            [vx,vy,flowImages] = opticalFlow(imgs(:,:,k),imgs(:,:,nodes(m)));
            flowNodes(k,nodes(m))=j;
            VX(j,:,:)=vx;
            % because the points gathered by the user are situated in
            % the fourth quadrant, X values behave normaly, but all y
            % values will be relativly opposite, so they need to be
            % negated.
            VY(j,:,:)=vy*(-1);
        %   flowIms(:,:,j)=flowImages;
       end
   end
end

%load start image to imPath.
clearvars imgPath endPath
imPath = zeros(length(actualPoints));

imgPath(1)=firstNode;
endPath(1)=firstNode;
inputs = length(actualPoints);

% for each node in imgPath, find three nodes that are the best distance
% away. for each of those nodes, follow the path subtracting the
```

```
    % vectors from the origional X and Y vectors. Pick the path with the
    % values closest to zero.

    for i=1:inputs-1
        % load first point in line segment
        X1=round(actualPoints(i,1));
        Y1=round(actualPoints(i,2));
        % load second point in line segment
        X2=round(actualPoints(i+1,1));
        Y2=round(actualPoints(i+1,2));
        % calculate X vector and Y vector
        Xdir=X2-X1;
        Ydir=Y2-Y1;
        % calculate distance
        len = sqrt(Xdir^2 +Ydir^2);

        % make an array of distances such that the index number is the
        % node, and the value is the distance. Need at least four nodes in
        % a graph. dists(1,1) should return the node with the best
        % distance, while dists(1,2) should return the value.

        distIndex = 1;
        for j=1:z
            if j~= imgPath(i)
                [dis, path, pred]=graphshortestpath(S,imgPath(i),j);

                % instead of using dis, use the sum of all vectors from
                % the optical flow
                dist = 0;
                xx = 0;
                yy = 0;
                for p=1:length(path)-1
                    inx=flowNodes(path(p),path(p+1));
                    xx= xx + VX(inx,X1,Y1);
                    yy= yy + VY(inx,X1,Y1);
                end
                dist = sqrt(xx^2 +yy^2);
                distCheck = dist-len;
                dists(distIndex, 1) = j;
                dists(distIndex, 2) = abs(distCheck);
                distIndex = distIndex+1;
            end
        end
        % dists now contains all the paths leading from the current node,
        % sorted based on aproximate distance to the desired location.
        % Note: because the optical flow and distance between two points is
        % not calculated the same, it's difficult to use these two numbers
        % together (as in the distCheck variable), however in this case, we
        % only need to find which gets closest to this value, so the
        % comparison is between all of the optical flow vectors. This still
        % can create errors in the computation.
        dists = sortrows(dists,2);

        % get the closest three distances to check which is pointing
        % closest to the right direction. More emphasis has been placed on
```

```
        % pointing in the right direction over getting to the right
        % distance. This is because there can be a number of totally
        % incorect images at the right distance that may be facing a
        % completely wrong direction, but a far jump in distance with the
        % face staying relativly in the same direction is less jaring.
        Xs = [dists(1,1) Xdir; dists(2,1) Xdir; dists(3,1) Xdir];
        Ys = [dists(1,1) Ydir; dists(2,1) Ydir; dists(3,1) Ydir];

        % find the shortest path between the current image and the three
        % distances to get the best path.
        for k=1:3
            [dist, path, pred]=graphshortestpath(S,imgPath(i),dists(k,1));
            Xval=X1;
            Yval=Y1;
            for L=1:length(path)-1
                index = flowNodes(path(L),path(L+1));
                vx = VX(index,Xval,Yval);
                vy = VY(index,Xval,Yval);
               % shift the current location to the new pixel value
                Xval = round(Xval + vx);
                Yval = round(Yval + vy);
                % subtract the change from the value to see how much
                % closer this move gets to the desired direction.
                Xs(k,2) = Xs(k,2) - vx;
                Ys(k,2) = Ys(k,2) - vy;
            end
        end

        % get the distance vector of the resulting Xs and Ys and compare
        % them to see how far off they are. Take the smallest magnitude,
        % and add the path from the current node to the calculated node to
        % the output path. If there are more user input points, add the
        % final node in the path to imgPath, and shift the userInput to the
        % calculated end point.

        d1 = dists(1,2) - sqrt(Xs(1,2)^2 + Ys(1,2)^2);
        d2 = dists(2,2) - sqrt(Xs(2,2)^2 + Ys(2,2)^2);
        d3 = dists(3,2) - sqrt(Xs(3,2)^2 + Ys(3,2)^2);

        % create an array of the values and nodes. format = (node,
        % distance, x vector, y vector
        ds = [dists(1,1) d1 Xs(1,2) Ys(1,2); dists(2,1) d2 Xs(2,2) Ys(2,2); dists(3,1)↙
d3 Xs(3,2) Ys(3,2)];
        % sort by row on the second column to keep all values with their
        % respective nodes
        ds = sortrows(ds,2);
        % take the first node, as it will be next to the smallest value.
        node = ds(1,1);
        % add the node to the list of connected nodes.
        endPath(length(endPath)+1) = node;

        [pointsLeft w] = size(actualPoints);
        % if there are points remaining, add the node to the list, and
        % increase the length of im to go through the loop again.
        if pointsLeft > i+1
```

```
            imgPath(i+1) = node;
        end
    end

    clearvars -except endPath S imgs dists imgsC userPoints VX VY flowNodes firstNode;


% get the list read to print
    for i=1:length(endPath)-1
        [dist, path, pred]=graphshortestpath(S,endPath(i),endPath(i+1));
        for j=1:length(path)
            paths(i,j)=path(j);
        end
    end

    [x y] = size(paths);
    num=x*y;
    [t u v] = size(imgs);
    % creat a single row of nums so it's easier to cycle through using a
    % single for loop
    pathSeq = reshape(paths',1,num);


    previous = 0;
    % remove any duplicate framse (assuming they are next to each other) by
    % turning them into zeros. This doesn't check for strings of frames
    % longer than 2. if there are three in a row, the image will be
    % displayed twice, as it is assumed that the face should stay put in
    % this case.
    for i=1:length(pathSeq)
        if(previous == pathSeq(i))
            previous = pathSeq(i);
            pathSeq(i) = 0;
        else
            previous = pathSeq(i);
        end
    end
    % remove zeros ( this removes duplicates and zero frame references.
    pathSeq=pathSeq(pathSeq ~= 0);


    % down select imgsC
    imgsC = imgsC(1:2:end,1:2:end,:,:);

    % now, construst the sequence of images
    for i=1:length(pathSeq)
        sequence(:,:,:,i)=imgsC(:,:,:,pathSeq(i));
    end

    % use sequence to take a frame one at a time and creat an avi movie

    [x,y,z,t] = size(sequence);
    for i=1:t
        fr = sequence(:,:,:,i);
        fr = double(fr);
```

```
        fr = fr/255;
        frame = im2frame(fr);
        frames(i) = frame;
    end
    fps=5;
    n=1;
    % output the video (without audio)
    movie2avi(frames, 'video.avi', 'fps', fps);


    %%%%%% this ends the general computation. The remaining computation is
    %%%%%% used to compute the video with audio added, as well as the
    %%%%%% actual path taken.


    % load audio (the audio file is included in the zip folder)
    [y, Fs] = wavread('gong');
    player = audioplayer(y, Fs);
    % use play(player); to play the file. add timing to get the sound to play once for↙
each line segment
    % create a movie and play sound. time = five frames per second.
    frameIndex = 0;
    fps = 5;
    % fix the frame in position
    figure('Position',[35 70 1000 675])
    axis off
    % For every line segment, play the video sequence, and audio, and wait
    % for the audio to finish.
    for i=1:length(endPath) - 1
        if endPath(i) ~=endPath(i+1) && endPath(i) ~=0
            n = [];
            n(1) = 1;
            [dist, path, pred]=graphshortestpath(S,endPath(i),endPath(i+1));
            for j=1:length(path)
                n(j+1) = frameIndex + j;
            end
            frameIndex = frameIndex + length(path) - 1;
            movie(frames,n, fps);
            % pauses computation until audio finishes.
            playblocking(player);
        end
    end

    % finally, save the origional path and new path overlayed ontop.
    [x y] = size(paths);

    for j=1:length(userPoints)
        userX(j)= round(userPoints(j,1));
        userY(j)= round(userPoints(j,2));
    end

    actualX = zeros(y,1);
    actualY = zeros(y,1);
    actualX(1) = userX(1);
    actualY(1) = userY(1);
```

```
    actualIndex = 2;
    % for every pair, get the vx and vy
    % the first for loop is the current node to deviate from, and the
    % second cycles through the individual nodes in the list
    for i=1:x
        for j=1:y-1
            if paths(i,j) ~= 0 && paths(i,j+1) ~= 0
                index = flowNodes(paths(i,j),paths(i,j+1));
                usX = round(userPoints(i+1,1));
                usY = round(userPoints(i+1,2));
                X = VX(index, usX, usY)*10;
                Y = VY(index, usX, usY)*10;
                actualX(actualIndex) = X + usX;
                actualY(actualIndex) = Y + usY;
                actualIndex  = actualIndex +1;
            end
        end
    end

    % print end paths together
    hold on;
    % origional is blue, new is red
    plot(userX, userY,':b');
    plot(actualX, actualY,'r');
    hold off;
    print('-djpeg', 'actual_path');
end


%
% gets the input from the user.
%
% Arguments:
%
%   im: the starting image
%
% Output:
%
%   returns the list of coordinates of the user clicks.
%
function c = getInput(im)
    f=figure;
    % down select image
    im = im(1:2:end,1:2:end,:);
    imshow(im,'InitialMagnification', 100);
    a = get(f,'Children');
    % get clicks
    [x,y] = ginput;
    % display path ontop of image
    hold(a);
    plot(x,y);
    hold off;
    % return path
    c = [x,y];
end
```

```
%
% computes similarity of all images in the sequence
%
% Arguments:
%
%   ims: the image sequence
%
% Output:
%
%   returns the matrix of all the differences.
%
function b = similarity(ims)
    [x,y,z] = size(ims);
    D=zeros(z,z);

    % this number is not perfect for all image sequences. For very small
    % sets of rather similar images, the threshold needs to be lower (@85),
    % but for larger sets of images there can be a lot of redundant
    % information or many clumps of unconnected sections of the graph. To
    % try to get most of the graph connected, higher values should be used
    % on poor data. For better data, lower thresholds can be used to
    % improve the graph.
    thresh=95;

    for i=1:z
        for j=1:z
            val =ims(:,:,i) - ims(:,:,j);
            temp=sqrt(sum(sum(val.^2)));
            if(temp<thresh)
                D(i,j)=temp;
            else
                D(i,j)=0;
            end
        end
    end

% return the similarity matrix
    b=D;
end

%
% computes the optical flow between two images
%
% Arguments:
%
%   im1: the first image
%   im2: the second image
%
% Output:
%
%   vx: x vectors of change between the two images
%   vy: y vectors of change between the two images
%   warpI2: The optical flow output image
%
```

```
function [vx,vy,warpI2]=opticalFlow(im1,im2)
 % down select images
   im1 = im1(1:2:end,1:2:end,:);
   im2 = im2(1:2:end,1:2:end,:);

 % code from demoflow.m

   alpha = 0.012;
   ratio = 0.75;
   minWidth = 20;
   nOuterFPIterations = 7;
   nInnerFPIterations = 1;
   nSORIterations = 30;

   para = [alpha,ratio,minWidth,nOuterFPIterations,nInnerFPIterations,nSORIterations];

% end code from demoflow.m
[vx,vy,warpI2]=Coarse2FineTwoFrames(im1,im2, para);

end
```