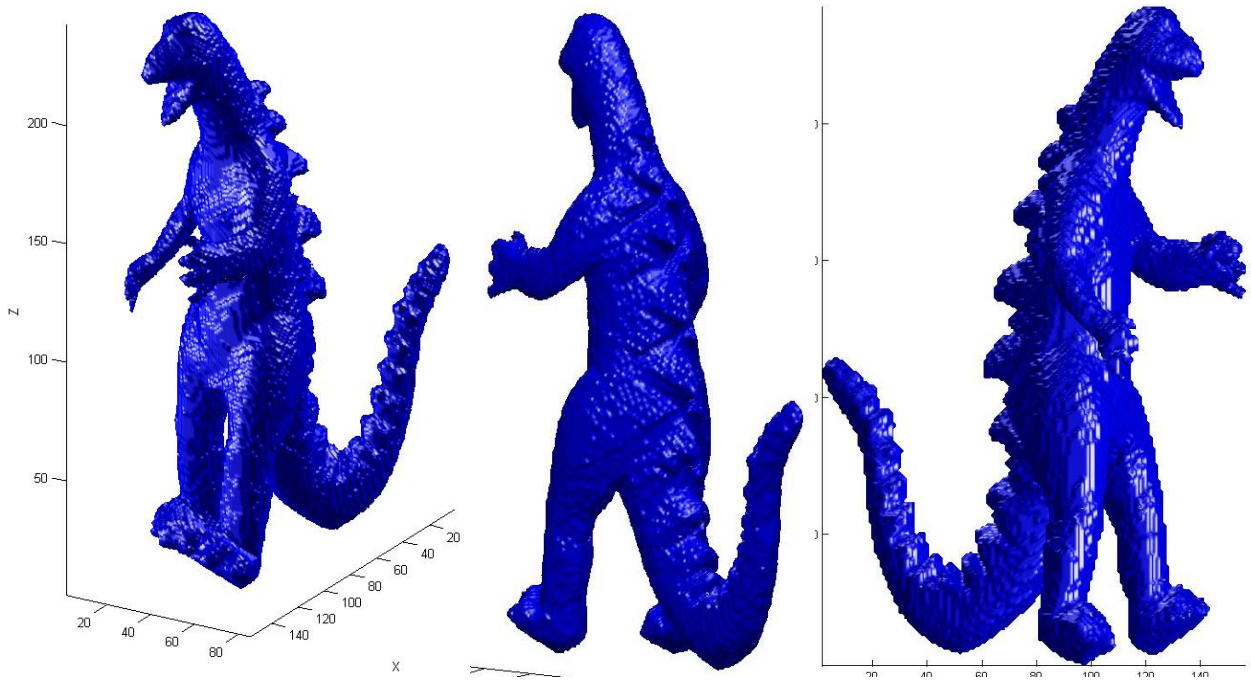
ELEC4630 Assessment Task 4

Of Faces and Dinosaurs – Michael Ball

Task 1: Model a Dinosaur from images and positional data

Given the extrinsic information of any camera it is possible to infer things from the pictures it takes. For this task the extrinsic information for 36 different images were supplied and were used to reconstruct a dinosaur figurine using. This process requires detecting the shape of the dinosaur within each image, mapping each image to a 3D voxel grid, cutting the shape of the dinosaur out of the grid at each of the different camera angles and then finally rendering them for display.



The most computationally intense part of the script is during the preparing to display portion of the code. This part of the code takes as long, if not longer than the rest of the script combined. The time taken there is due to it having to access all of the voxels of the mesh, flipping them to be the 'right' way up and thus producing the final model.

Script Output:

```
load the pictures
...done
adding the camera data
...done
Finding the outline of the dinosaur in each image
Cutting out the dinosaur! View #1
Cutting out the dinosaur! View #2
Cutting out the dinosaur! View #3
Cutting out the dinosaur! View #4
Cutting out the dinosaur! View #5
Cutting out the dinosaur! View #6
Cutting out the dinosaur! View #7
Cutting out the dinosaur! View #8
Cutting out the dinosaur! View #9
Cutting out the dinosaur! View #10
Cutting out the dinosaur! View #11
Cutting out the dinosaur! View #12
Cutting out the dinosaur! View #13
Cutting out the dinosaur! View #14
Cutting out the dinosaur! View #15
Cutting out the dinosaur! View #16
```

Cutting out the dinosaur! View #17
Cutting out the dinosaur! View #18
Cutting out the dinosaur! View #19
Cutting out the dinosaur! View #20
Cutting out the dinosaur! View #21
Cutting out the dinosaur! View #22
Cutting out the dinosaur! View #23
Cutting out the dinosaur! View #24
Cutting out the dinosaur! View #25
Cutting out the dinosaur! View #26
Cutting out the dinosaur! View #27
Cutting out the dinosaur! View #28
Cutting out the dinosaur! View #29
Cutting out the dinosaur! View #30
Cutting out the dinosaur! View #31
Cutting out the dinosaur! View #32
Cutting out the dinosaur! View #33
Cutting out the dinosaur! View #34
Cutting out the dinosaur! View #35
Cutting out the dinosaur! View #36
...done
Setting up for carving
...done
Starting the carving
Camera #1 carved!
Camera #2 carved!
Camera #3 carved!
Camera #4 carved!
Camera #5 carved!
Camera #6 carved!
Camera #7 carved!
Camera #8 carved!
Camera #9 carved!
Camera #10 carved!
Camera #11 carved!
Camera #12 carved!
Camera #13 carved!
Camera #14 carved!
Camera #15 carved!
Camera #16 carved!
Camera #17 carved!
Camera #18 carved!
Camera #19 carved!
Camera #20 carved!
Camera #21 carved!
Camera #22 carved!
Camera #23 carved!
Camera #24 carved!
Camera #25 carved!
Camera #26 carved!
Camera #27 carved!
Camera #28 carved!
Camera #29 carved!
Camera #30 carved!
Camera #31 carved!
Camera #32 carved!
Camera #33 carved!
Camera #34 carved!
Camera #35 carved!
Camera #36 carved!

...done
Preparing to and then displaying:
...done

Code:

```
% Script made by s4262468 as the solution to part 1 of the 4th assessment
% task of ELEC4630

close all, clear, clc

% Load all the images from all the folders
disp('load the pictures')
imgType = '*.jpg'; % change based on image type
addpath('dino');
imgPath = ['dino/'];
images  = dir([imgPath imgType]);
for idx = 1:length(images)
    [img, map] = imread([imgPath images(idx).name], 'jpg');
    cam{idx}.img = im2double(img);
end
disp('...done')

disp('adding the camera data')
Dino_projection_matrices()
for idx = 1:length(images)
    cam{idx}.p = eval(['P' int2str(idx-1)]);
end

clearvars -except cam
disp('...done')

disp('Finding the outline of the dinosaur in each image')
for i = 1:length(cam),
    disp(['Cutting out the dinosaur! View #' int2str(i)])
    t = rgb2hsv(cam{i}.img);
    cam{i}.cut = t(:,:,1)>0.7 | t(:,:,1)<0.3; %super-simple cutting
    cam{i}.cut = imclearborder(cam{i}.cut);
    cam{i}.cut = imfill(cam{i}.cut, 'holes');
%     figure, imshow(cam{i}.cut)
end
disp('...done')

disp('Setting up for carving')
minX = -180; maxX = 90;
minY = -80; maxY = 70;
minZ = 20; maxZ = 460;

% Volume of spaaaaaace
vol = dist(minX,maxX) * dist(minY,maxY) * dist(minZ,maxZ);

% Number of voxels. if vol==numVox then it's full resolution
numVox = vol/5;
V.res = (vol/numVox)^(1/3); % should be 1 if full res

% create the meshgrid
x = minX:V.res:maxX;
y = minY:V.res:maxY;
z = minZ:V.res:maxZ;

% V is going to be the shape we carve out to make the dino.
[V.X, V.Y, V.Z] = meshgrid(x, y, z);
```

```
V.val = ones(numel(V.X),1);
disp('...done')

disp('Starting the carving')
for i = 1:length(cam),
    c = cam{i}; %

    z = c.p(3,1) * V.X + c.p(3,2) * V.Y ...
      + c.p(3,3) * V.Z + c.p(3,4);

    % Make the flat 'image' seen by the camera.
    y_plane = round((c.p(2,1) * V.X + c.p(2,2) * V.Y ...
    + c.p(2,3) * V.Z + c.p(2,4)) ./ z);

    x_plane = round( (c.p(1,1) * V.X + c.p(1,2) * V.Y ...
    + c.p(1,3) * V.Z + c.p(1,4)) ./ z);

    % Points could be outside of the image we just constructed.
    % Let's get rid of them. >:D
    [h,w,d] = size(c.img);
    keep = find( (x_plane>=1) & (x_plane<=w) & (y_plane>=1) & (y_plane<=h) );
    x_plane = x_plane(keep); % this then masks the image to only be inside w&h
    y_plane = y_plane(keep);

    % Now clear any that are not inside the silhouette
    ind = sub2ind([h,w], round(y_plane), round(x_plane) );
    keep = keep(c.cut(ind) >= 1); % Indices come in hand as the coordinate system here

    V.X = V.X(keep);
    V.Y = V.Y(keep);
    V.Z = V.Z(keep);
    V.val = V.val(keep);

    disp(['Camera #' int2str(i) ' carved!'])
end
disp('...done')

disp('Preparing to and then displaying: ')

% First grid the data
ux = unique(V.X); uy = unique(V.Y); uz = unique(V.Z);

% Expand the model by one voxel in each direction
ux = [ux(1)-V.res; ux; ux(end)+V.res];
uy = [uy(1)-V.res; uy; uy(end)+V.res];
uz = [uz(1)-V.res; uz; uz(end)+V.res];

[X,~,~] = meshgrid( ux, uy, uz ); % we only need X here

% Create an empty voxel grid, then fill only those elements in voxels
v = zeros(size(X));

% Here we flip the model
% This is the most computationally intensive part of the code as we must
% move every single voxel in order to mirror the dino.
for ii=1:numel(V.X)
    ix = (ux == V.X(ii));
    iy = flipud(uy == V.Y(ii));
    iz = flipud(uz == V.Z(ii));
    v(iy,ix,iz) = V.val(ii);
end
```

```
% Now draw it :D
figure; p = patch(isosurface(v, 0.5));
set( p, 'FaceColor', 'b', 'EdgeColor', 'none' );
set(gca,'DataAspectRatio',[1 1 1]);
xlabel('X'); ylabel('Y'); zlabel('Z');
view(-140,22); axis('tight')
lighting('gouraud')
camlight(0,22); camlight(180,22);
disp('...done')
```

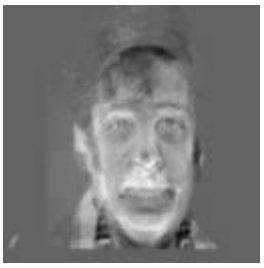
Task 2: Make an accurate Face detection system using Eigenfaces

The eigenface method attempts to match faces to a set of 'training faces' using each eigenvectors. Instead of working in 2-dimensional space each picture involved is converted into a row vector, with the rows of the image stacked end to end. The eigenface method requires the most computation while retrieving the weights of the training data. For an image set of *J* images each with a size of *N*x*m*, matrices of (*N*x*m*)x*J* must be multiplied. Thus for even small images with 100 pixel sides we must deal with matrices with lengths of 10,000 points.

A face is considered matched if it's the minimum distance between its calculated eigenface weights is smaller than a certain threshold and is matched to the training face that is the smallest distance from it in the 'facespace.' In this case the distance between faces is calculated using the following formula:

$$\|\Omega - \Omega^k\| = \sum_{i=1}^{K} (w_i - w_i^k)^2$$

The eigen information obtained from the training set is as follows:


$\lambda_1 = 17.97$


$\lambda_2 = 16.05$


$\lambda_3 = 9.05$


$\lambda_4 = 8.48$


$\lambda_5 = 7.94$


$\lambda_6 = 1.25 \times 10^{-13}$

Each of these 'eigenfaces' are really just eigen vectors that have been transformed back into 2 dimensional space. The images above have been normalised so that the details of the faces are revealed.

The distance formula mentioned above is a modified version of the Mahalanobis distance. One of the resources used while implementing the eigenface system[1] suggested that the Mahalanobis distance performed better in this application. Due to the way the suggested distance formula used eigenvalues and the weights of each face, incorrect recognition occurred. The tiny size of λ_6 meant that any face being tested against face 6 would have an enormous calculated distance in the facespace. This problem was resolved by modifying the distance formula to remove its reliance upon eigenvalues.

A rudimentary GUI was implemented that allows a user to choose a new picture to compare through the use of a file selection window. The picture is then matched against the training faces, shown with its match in a figure and the distance in facespace between the 2 printed to the console. Another dialog box then asks the user if they'd like to test another picture and closes if 'no' or 'cancel' are selected.

[1] (M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991, hard copy) http://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf

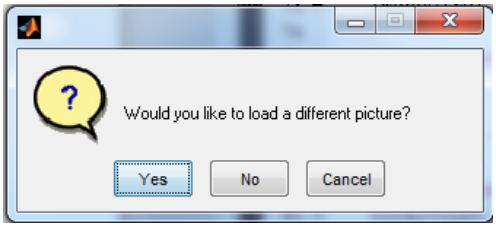


Figure 1: Asks whether you'd like to check other pictures

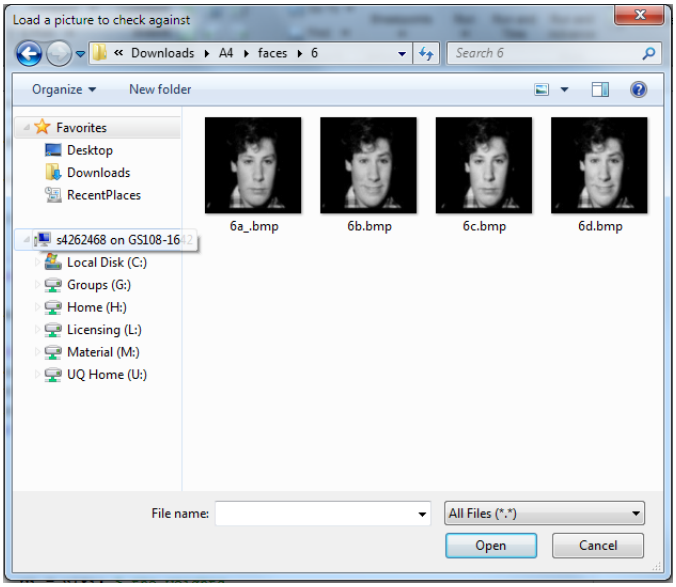


Figure 2: The picture selection window

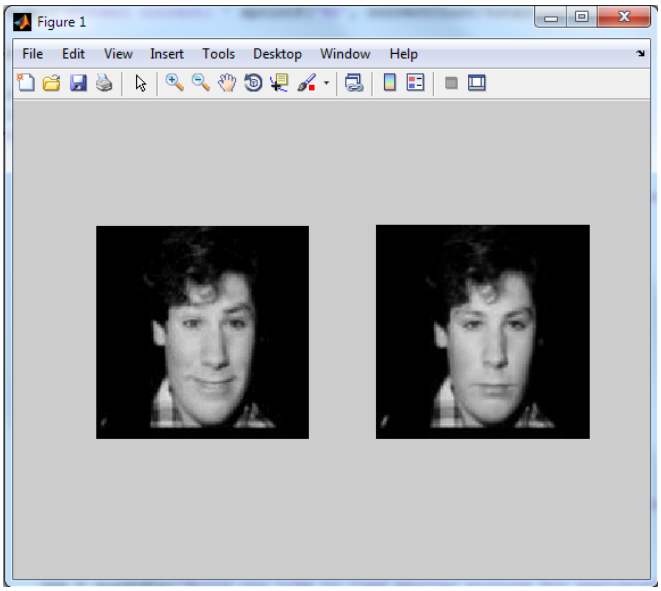


Figure 3: Comparison of selected and matched faces

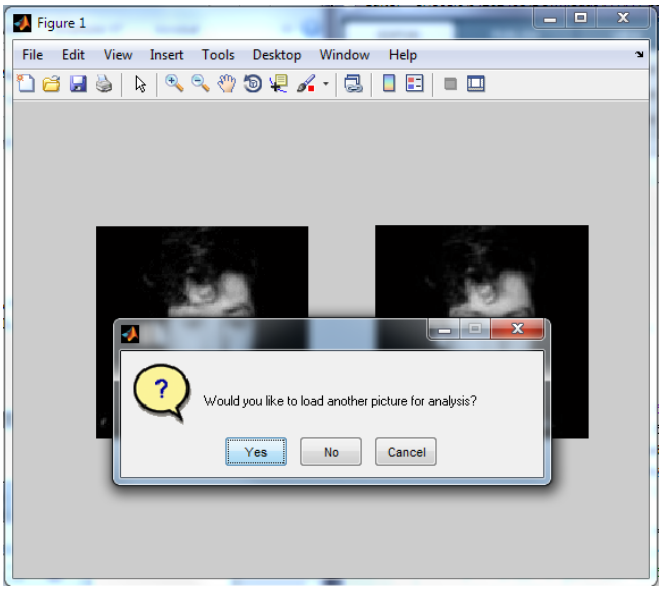


Figure 4: Asking if you'd like to compare another face

Script Output:

load the faces
...done
load the trainging faces
...done
Find difference images
...done
Covariance time and eigenfaces
...done
Reconstructing face #4
...done
Proceeding to match all previously loaded images to training faces
Person 1's picture #1 matches training face #1 with 0.000030 difference
Person 1's picture #2 matches training face #1 with 26.595762 difference
Person 1's picture #3 matches training face #1 with 54.788052 difference
Person 1's picture #4 matches training face #1 with 14.043936 difference
Person 1's picture #5 matches training face #1 with 40.094218 difference
Person 1's picture #6 matches training face #1 with 34.008767 difference
Person 1's picture #7 matches training face #1 with 23.797907 difference
.

```
Person 2's picture #1 matches training face #2 with 0.000382 difference
Person 2's picture #2 matches training face #2 with 7.421324 difference
Person 2's picture #3 matches training face #2 with 4.784634 difference
Person 2's picture #4 matches training face #2 with 9.293738 difference
.
Person 3's picture #1 matches training face #3 with 0.042323 difference
Person 3's picture #2 matches training face #3 with 10.416778 difference
Person 3's picture #3 matches training face #5 with 47.201309 difference
    The distance between this and the correct face was: 67.404429
Person 3's picture #4 matches training face #3 with 40.829404 difference
Person 3's picture #5 matches training face #3 with 22.004114 difference
Person 3's picture #6 matches training face #3 with 24.181621 difference
Person 3's picture #7 matches training face #3 with 45.134281 difference
Person 3's picture #8 matches training face #3 with 17.628958 difference
Person 3's picture #9 matches training face #3 with 40.031185 difference
Person 3's picture #10 matches training face #3 with 14.743086 difference
.
Person 4's picture #1 matches training face #4 with 0.046258 difference
Person 4's picture #2 matches training face #4 with 2.594970 difference
Person 4's picture #3 matches training face #4 with 16.902938 difference
Person 4's picture #4 matches training face #4 with 29.870594 difference
.
Person 5's picture #1 matches training face #5 with 0.060884 difference
Person 5's picture #2 matches training face #5 with 1.471047 difference
Person 5's picture #3 matches training face #5 with 17.007150 difference
Person 5's picture #4 matches training face #5 with 15.129118 difference
.
Person 6's picture #1 matches training face #6 with 0.054859 difference
Person 6's picture #2 matches training face #6 with 2.304627 difference
Person 6's picture #3 matches training face #6 with 29.045290 difference
Person 6's picture #4 matches training face #6 with 16.982313 difference
.
Percent correct: 96.969697
...done
Opening dialog for checking of specific files
Loaded picture
Your picture was matched to the other face in the figure with a distance of 2.304627
...finished face detection script
```

Code:

```
% Script made by s4262468 as the solution to part 2 of the 4th assessment
% task of ELEC4630

close all, clear, clc

% Load all the images from all the folders
disp('load the faces')
imgType = '*.bmp'; % change based on image type
addpath('faces');
for j = 1:6,
    imgPath = ['Faces/' int2str(j) '/'];
    images  = dir([imgPath imgType]);
    for idx = 1:length(images)
        [img, map] = imread([imgPath images(idx).name], 'bmp');
        f{j}{idx}.img = im2double(rgb2gray(ind2rgb(img, map)));
    end
end
disp('...done')

disp('load the trainging faces')
```

```
% Make the mean image
imgType = '*.bmp'; % change based on image type
addpath('faces');
images  = dir(['Faces/eig/' imgType]);
for idx = 1:length(images)
    [i,m] = imread(['Faces/eig/' images(idx).name], 'bmp');
    tmp = im2double(rgb2gray( ind2rgb(i, m)));
    trn(:,:,idx) = tmp;%/max(max(tmp));
end
disp('...done')

disp('Find difference images')
[h,w,d] = size(trn); n = h*w;
trn = reshape(trn,[n d]);
m = mean(trn,2);
for i = 1:length(images), dif(:,i) = trn(:,i) - m; end
disp('...done')

disp('Covariance time and eigenfaces')
[U,S,V] = svd(dif);

s = max(S);
for i=1:length(images),
    tmp = (dif*V);
    u(:,i) = (1/s(1,i)).* tmp(:,i);
end

wght = u'*dif; % find the weights
faces={};
for i=1:length(images),
    faces{i} = reshape(u(:,i), [h,w]);
    tmp = faces{i};
    t_min = min(min(tmp));
    tmp = tmp-t_min;
    t_max = max(max(tmp));
    tmp = (tmp)./t_max;
    figure, imshow(tmp);
%     imwrite(tmp, ['eig' int2str(i) '.png'])
end
close all
disp('...done')

face = 4;
disp(['Reconstructing face #' int2str(face)])
re = m;
for i = 1:length(images),
    weighting = wght(i,face);
    re = re + weighting * u(:,i);
    figure, imshow(reshape(re, [h,w]))
end
re = reshape(re, [h,w]);

imshow(re)

disp('...done')
close all

clearvars -except f wght S s U u V m trn

disp('Proceeding to match all previously loaded images to training faces')
totalCount = 0;
correctCount = 0;
```

```
for person = 1:length(f),
    for face = 1:length(f{person}),
        totalCount = totalCount+1;

        img = f{person}{face}.img;
        [h,w,d] = size(img); n = h*w;
        i = reshape(img, [n d]);
        i = i-m; % get the difference vector

        wi = u'*i; % the weights
        f{person}{face}.w = wi;



        for tface = 1:size(wght,2),
            dif(tface) = sum((wght(:,tface) - wi).^2);
        end

        match = find(dif==min(dif));
        f{person}{face}.d = dif;

        str = ['Person ' int2str(person) '''s picture #' int2str(face) ...
            ' matches training face #' int2str(match) ' with ' sprintf('%f', min(dif))
' difference'];

        disp(str)

        if (match == person), correctCount = correctCount+1;
        else
            % What was the value we wanted the?
            disp(['     The distance between this and the correct face was: '
sprintf('%f', dif(person))])
        end

    end
    disp('.')
end
disp(['Percent correct: ' sprintf('%f', correctCount/totalCount*100)]);

disp('...done')

res = questdlg('Would you like to load a different picture?');
if (strcmp('Yes', res)),
while(1),
    close all
    disp('Opening dialog for checking of specific files')
    [filename, pathname, filterindex] = uigetfile('*.*', 'Load a picture to check
against');
    [picked, map] = imread([pathname filename]);
    i = im2double(rgb2gray(ind2rgb(picked, map)));
    disp('Loaded picture')

    [h,w,d] = size(i); n = h*w;
    i = reshape(i, [n d]);
    i = i-m; % get the difference vector
    wi = u'*i; % the weights

    for tface = 1:size(wght,2),
        dif(tface) = sum((wght(:,tface) - wi).^2);
    end
    match = find(dif==min(dif));
```

```
    disp(['Your picture was matched to the other face in the figure with a distance of
' sprintf('%f',min(dif))])
    figure, subplot(1,2,1), imshow(rgb2gray(ind2rgb(picked, map)));
            subplot(1,2,2), imshow(reshape(trn(:,match), [h,w]));

    res = questdlg('Would you like to load another picture for analysis?');
    if (strcmp('No', res)), break; end
    if (strcmp('Cancel', res)), break; end
end
end
disp('...finished face detection script')
```