

Watson Capstone Projects (WCP)
WCP28 Spotting Salamanders



Sponsor: Avangrid Foundation

Submitted by:
Christian Tejera, CoE
Jocelyn Ao, CS

Faculty Advisor: Prof. Scott Craver

May 3, 2019
Revision: -

Approved for public release; distribution is unlimited.
Submitted in partial fulfillment of WCP academic requirements.

Thomas J. Watson School of Engineering and Applied Science
Binghamton University
Binghamton, NY

Executive Summary

Dylan Horvath is conducting research on the migration pattern of salamanders within the Binghamton area. These salamanders' path cannot be interrupted. Therefore, majority of the pictures are taken in the dark (salamanders tend to move at night.) They are usually found on the road which has a similar color to the salamanders. This poses as a problem because currently no program exists that can identify and match animals unless they can be differentiated from their background. Dylan and the other members of his research would draw out the spot pattern by hand, and "go by eye" to match each individual salamander. There are currently over 70 different pictures of these salamanders.

We planned to create a program that will use each salamander's spot pattern to match them. Every salamander has a unique spot pattern which is never replicated by any other salamander. Prior to the project launch, this was done by selecting the spots manually on photoshop. The images were then grayscaled and placed on top of one another. Mean squared error was used to determine the similarity of the images.

For our design solution, we attempt to pre-process all the images which includes rescaling, converting and straightening the image. ImageMagick was used as a command line tool that could rescale and convert the image to a Portable Pixel Map file (PPM). With the use of Bezeir Curves and pixel manipulation, we aimed to straighten the salamander's curved spine. Alongside this, we developed a method to apply FisherFaces to a group of photos using C++ and OpenCV libraries. FisherFaces takes the set of pre-processed photos and predicts an image that is most like it. A threshold was applied to increase the precision of the match.

This is all done in the background of a web application. The web application sends out a web server request using AJAX to run a script that holds the FisherFace code. The communication consists of the web application requesting the matched image and salamander group. The server runs locally on our laptop using XAMPP.

At the highest level, the user sees a web application. The web application prompts the user to input an image of a salamander. The user will then be asked to click points along the spine of salamander. These points are used in the process of straightening the salamander. Once the server script computes the FisherFaces of the images, it returns an answer of whether the salamander is matched along with the salamander's name in the database.

Ultimately, we were successful in creating the framework for matching the salamander. The web application has an established connection to the server and can execute c++ and ImageMagick applications. However, we were unable to find a solution for pre-processing the images and therefore unable to fully test the FisherFace code. Instead we have a functioning FisherFace code with the ability to manipulate the tolerance of a match and the prototype to a straightening program. We remain optimistic that we will develop a solution soon.

Table of Contents

List of Figures	i
List of Tables	ii
1. Introduction	1
1.1. Initial Scope	1
1.2. Applicable Documents	1
2. Problem Definition.....	1
2.1. Project Scope.....	1
2.2. Technical Review	2
2.3. Design Requirements	2
3. Design Description	3
3.1. Overview	3
3.2. Detailed Description	5
3.2.1. Web Application	4
3.2.2. Server AJAX	Error! Bookmark not defined.
3.2.3. Initial Identification Design	7
3.2.4. FisherFaces Identification Design	7
3.3. Use.....	9
4. Implementation.....	9
5. Finances.....	10
6. Evaluation and Plans.....	10
6.1. Overview	10
6.2. Testing And Results	10
6.3. Assessment.....	12
6.4. Future Plans.....	12
References.....	13
Appendix A: Test Results for Each Requirement.....	15
Appendix B: Server Code	17
Appendix C: Web Application Code	37

List of Figures

Figure 1. Operational Context Diagram	4
Figure 2. System Diagram	5
Figure 3. Web application block diagram	6
Figure 4. PCA	8
Figure 5. WebPage	16
Figure 6. Output of Straightening	16

List of Tables

Table 1. Project Finances	9
---------------------------------	---

1. Introduction

1.1. Initial Scope

Dylan Horvath is tracking the migration pattern of salamanders in the nature preserve of Binghamton University. Due to the salamanders usually moving at night, it is hard to distinguish the salamander from the background of the captured images. Our goal is to create a program that can match salamanders based on their spot pattern.

1.2. Applicable Documents

Refer to the following documents for more information about the project.

1. WCP28 Spotting Salamanders Project Specification Document, published September 28, 2018.
2. WCP28 Spotting Salamanders Implementation and Test Plan, February 19, 2019
3. WCP28 Spotting Salamanders System Verification, published March 8, 2019.

2. Problem Definition

2.1. Project Scope

Many image processing programs rely on deviations between its subject and the background in order to identify the subject. In the images for this project, it is hard for a program to distinguish the salamanders from the background. Our goal is to devise a method to identify the salamanders simply due to their spot pattern. We planned to devise a method of pre-processing the images and applying FisherFaces to a set of images. This would hopefully output a correctly matched salamander. We, also, planned to create a user interface to guide the user. Our stretch goal was to develop the program into a mobile app, however, we feel that any mobile device with a web browser should be able to run the program.

2.2. Technical Review

The natural world is one that is everchanging which prompts questions about the animals within. As environments are transformed by the evolution of technology, researchers seek to understand how animals react to it. In the case of salamanders, there is little research on salamanders' migration pattern. The landscape of Binghamton University is constantly evolving; so, do these salamanders travel on the same path or are the changes to the environment causing their paths to change? In order to track this, there must be a method to identify salamanders. With these animals, it is done through their unique spot pattern.

There are many programs that are used by environmental researchers to identify animals. According to Dylan, these programs are not user friendly. Also, many of these programs do not account for a situation where the background of the image is the same color as the animal. For example, these programs will have trouble identifying a blue fish in a blue background. Since image processing is a growing commodity in many programs being built today, the importance of accurate image processing cannot be understated. There is currently countless research being put into the improvement of facial recognition which have sparked the creation of several open source libraries dedicated to functions for image processing.

2.3. Design Requirements

All of the requirements can be found in the WCP28 Spotting Salamanders Project specifications.

The program shall tell the user to only upload a picture that is from the top-down perspective. {WCP28-R-002}

- The salamanders have a spot pattern along the top of their body. It is very important that the entire spot pattern is captured in the picture. Also, there can be distortion in the spot pattern when the pictures are taken at an angle. The more consistent the orientation of the pictures are, the more accurate the program will be.

The program shall be able to separate the salamander and its spots from the rest of the picture. {WCP28-R-003}

- The salamanders have a unique spot pattern that can be used to identify them (like a fingerprint.) These spots, which are yellow, heavily contrast the salamanders' black bodies. The spots are the perfect artifacts to extract from the images.

The program shall be able to orient and transform the picture of the salamander to fit the reference picture as best as possible. {WCP28-R-004}

- To properly match the salamanders, they must be aligned the exact same way. In its most basic form, matching two image requires them to be placed on top of each other. The differences between every pixel is accounted for and if the spots line up, the salamanders are the same.

The program shall have a computation that demonstrates the similarity between salamander images. {WCP28-R-006}

- Most codes need a mathematical computation to produce an output. There are many computations that use the pixels that are the same in two images to state whether two pictures are of the same thing.

The application should have a reference database that compares the captured picture with pictures in the database. {WCP28-G-004}

- A database will improve the accuracy of the program. This is because the more images there are of the same salamander; the more chances there are to identify the salamander.

3.Design Description

3.1. Overview

Our program consists of a user interface that contacts to a server. The system begins by prompting the user to be upload an image to the web application. This web application is created using JavaScript and HTML, and it serves as a user interface. The user can click on multiple buttons prompting a specified action. There is also a feature to trace a line on the salamander's spine which will later be used for straightening the salamander.

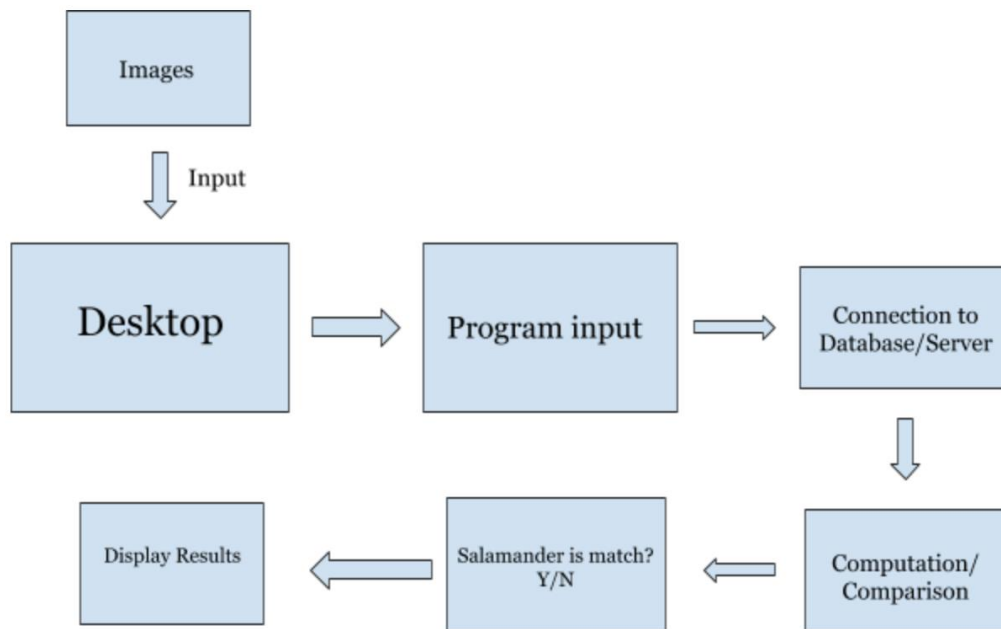


Figure 1. Operational Context Diagram

Once the images are uploaded and traced, they are sent as items to a server. The server uses this data as the inputs of the code which pre-processes the image and outputs a matched image. This result is returned to the web application.

3.2 Detailed Description

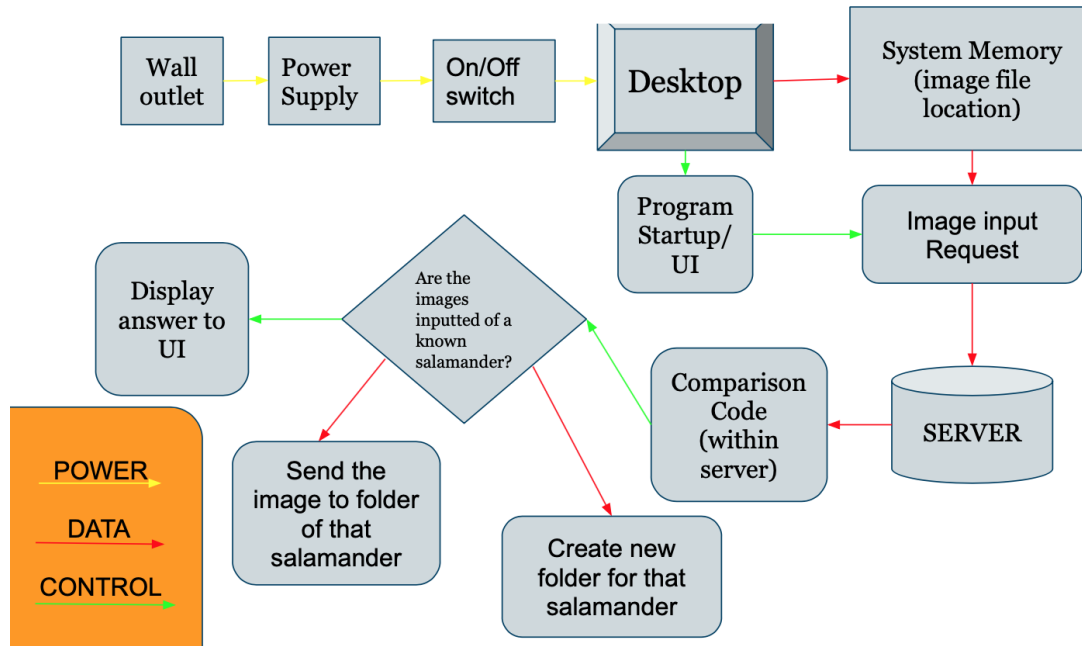


Figure 2. System Diagram

The system is split into three parts: the web application, server, and program code. The web application serves as a user interface for the program. The web application sends a request with the desired images and moused clicked coordinates to the server. This response is computed within the server which contains a code that will find a matched image.

3.1.1. Web Application

The web application provides a clear user interface so that a user can upload an image and mark out the spine of the salamander. The web application was created using HTML, CSS, and Javascript. Once a user clicks on the image, a mouse event will be triggered that causes the program to call on multiple Javascript functions that find the mouse's coordinate values and draw a line connecting those points. The web application also has a button that will trigger an AJAX call to open another program to modify the Salamander image using the coordinate data.

When the user clicks on the image, this notifies the program that a mouse event has occurred. This triggers the program to first find the image's position within the page which will be where the image's top left corner is. This is done by

calculating how far left and down the image is in pixels relative to the web application. Then the image's position is subtracted from the mouse's x and y coordinates so that the coordinate values will align with where the image starts in the entire web application. The values are then stored into arrays so that a line can be drawn that connects all the points.

To draw a line over the image, an HTML canvas object had to be created and placed over the picture. An event listener also had to be added to the canvas object so that whenever the user clicks on the image, the canvas will trigger a function that calculates the point coordinates. After the coordinate values have been generated, a drawing tool will be created from the canvas to draw a line connecting the last two points in the arrays.

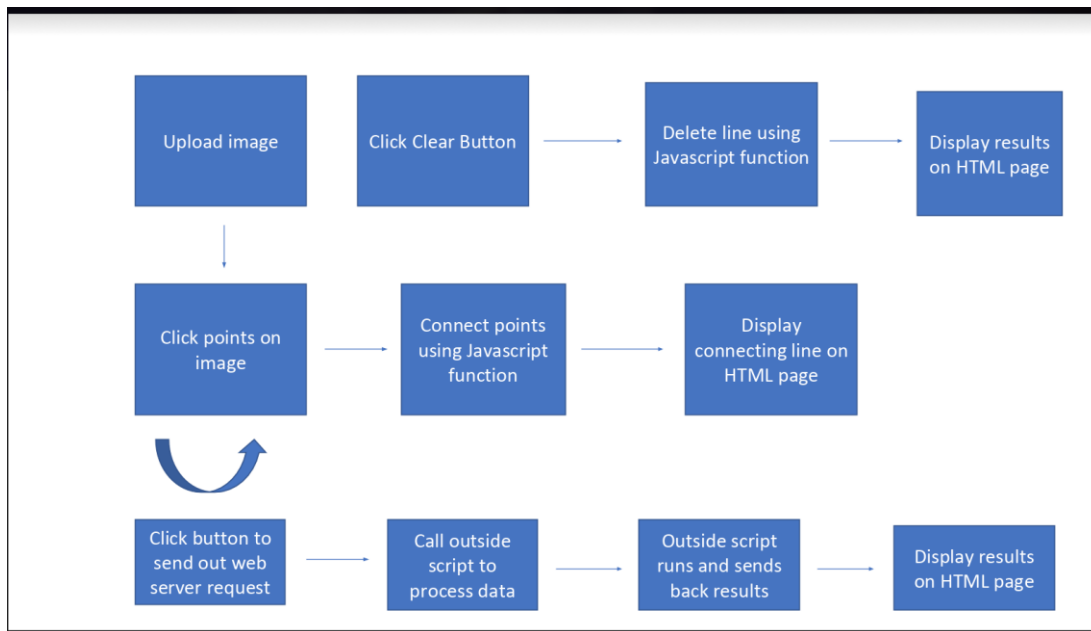


Figure 3: WebPage block diagram

3.1.2. Server AJAX

To alter the salamander image using an outside script, a local web server is used to host the web application so that the web application can send out web server requests using AJAX and XAMPP. XAMPP, is a program that establishes an environment for connecting the server and Web Application. With AJAX, an object can be created that requests data from another resource such as a script. The request web application sends out the coordinates of a salamander's spine and uses AJAX to call a script that will incorporate FisherFaces that finds the matching image.

3.1.3. Initial Identification Design

Our first design was to create a database of salamander images. These images would be of the same size (ex. 100x100) We would then allow the user to upload a single image and click the tip of the head of the salamander. This location will be stored as a coordinate. The coordinate would shift that head to the middle of the image (in ex. 50th pixel.) The other images will already have the head positioned in the middle of the image.

Mean squared error (MSE) is used to compare the images. MSE will take each pixel of two images and designate a value of 1 if the pixels differ or a zero if they are the same. The pixels that will be compared are based off of their coordinate location (pixels of the same location are compared.) Once all these values are gathered, they are summed and average. This average serves as the percentage of similarity. An average of zero would mean two images are the same.

The problem with this design is that it is extremely slow due to the time it takes to compute MSE (a lot of information will be compared.) Also, if a match does not exist in the database, the salamanders will be mismatched. This method also doesn't account for if the salamander has a curved spine. If the inputted image was a straight salamander, MSE will not find that salamander as a match to the curved one.

3.1.4. FisherFaces Identification Design

Our design uses fisherfaces as the basis for matching the images. Fisherfaces uses principal component analysis (PCA) and Linear Discriminant Analysis (LDA) to classify images. This begins by plotting all the data/images on a grid. These images have differences between them that will determine their location on the graph, also known as dimensions. This process of finding these differences is described by PCA.

PCA finds the dimensions that provides the most variance. For example, let's say a set of data has three variables: height, age and gender. If the set of data shows variance in the height and age, but little in gender, we can ignore the gender dimension and simplify the grid to a 2-D plane of height and age.

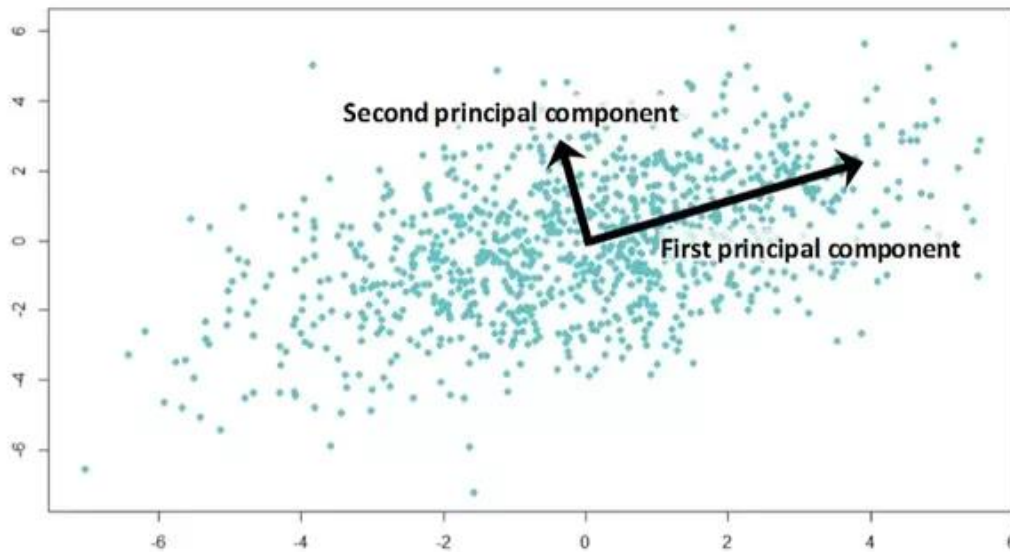
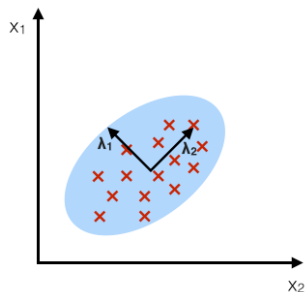


Figure 4: PCA

The line that says principle component represents the dimension with the most variance. The next principal component taken must be independent (or orthogonal) of the principal component.

PCA:

component axes that maximize the variance



LDA:

maximizing the component axes for class-separation

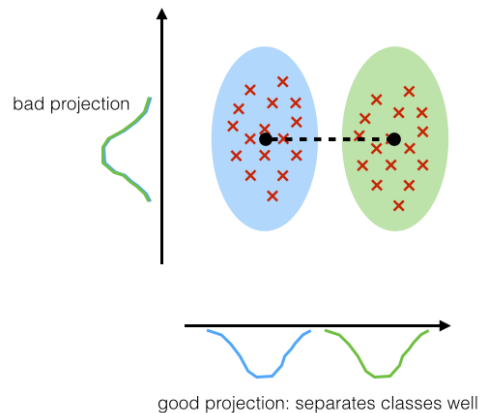


Figure 5: LDA

LDA takes this result further and attempts to separates the data set into classes/groups. These classes consist of multiple images of the same person or thing. The method for comparing is improved with the more images of the same entity; because the program will place them in the same class. [1]

We use FisherFaces to find the most important dimensions and then classify the images of the salamanders. Then we use Euclidean distance, which simply finds the shortest distance between data points in a dimensional space, to find a match.

Images that are close to each other have a high chance of being in the same class and therefore a match.

$$\begin{aligned}
 d(p, q) &= d(q, p) \\
 &= \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} \\
 &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}
 \end{aligned}$$

Figure 6: Euclidean Distance

OpenCV is required to use FisherFaces. OpenCV is an open sourced library that has real-time computer functions. This includes facial recognition functions and algorithms such as FisherFaces. The FisherFace code calls OpenCV functions to append all the photos to a matrix. Then PCA and LDA is applied to the matrix and outputs the predicted label. To properly use this function, the photos must be pre-processed.

FisherFaces require that the image set to be of the same dimensions. ImageMagick is a command line tool that allows its user to manipulate photos. ImageMagick comes with a built-in function called “magick” which has a vast array of uses. We use the function to scale the image to 500x400 which is the size for all the images in our data set. There is also the problem of the salamander’s curved spine.

Since it displaces the spots, a salamander’s curved spine can distort the results. At the start of the web application, the user clicks along the spine of the salamander. We will then create a Bezier curve using these clicked points. A Bezier curve is a parametric curve that predicts what points would make up a curve that passes through a given set of points, also known as Cubic Interpolation.

$$\mathbf{Bezier(t)} = (1 - t)^3 \mathbf{P_0} + 3(1-t)t^2 \mathbf{P_1} + 3(1-t)t^2 \mathbf{P_2} + t^3 \mathbf{P_3}.$$

P₀ and P₃ are the points that will be passed through. P₁ and P₂ determine the shape of the curve. Furthermore, B(t), where t=[0,1], equals all the point in a curve between p₀ and p₃. This must be done for the x coordinates and y coordinates separately.

When the image is a ppm, we can use a ppm function to manipulate each pixel in the image. Our method for straightening involves placing all the pixels corresponding with the spine of the salamander in the center of the image. We then loop over points perpendicular to the curve; re-arranging the pictures to align the salamander straight across the center.

The FisherFace code requires all the images be of the same size and straightened. It also requires a unique labeling system. Each photo must have a tag attached to the end of its name corresponding with the class the image is expected to be in. The first salamander in our database will have a tag of 0. Ultimately, the

algorithm uses this to predict the salamander the inputted picture most likely matches with.

3.2. Use

This program is used through a web application that can be found on a web browser. The server will contain a database of all the image of the salamanders. The user will upload an image and the application will display a match or create a new salamander in our database.

4. Implementation

The goal of the prototype was to create the build for the website and matching programs. Since this is a software project our prototype was everchanging throughout the debugging process and will be reused in the final build.

The prototype consisted of two sets of programs that correspond with the web application and server. The web application prototype can load, display and copy pictures on the website. It also can find points on an image and display the coordinates and the point on the image. Using the program xampp, the website has a connection to a local server, our laptop.

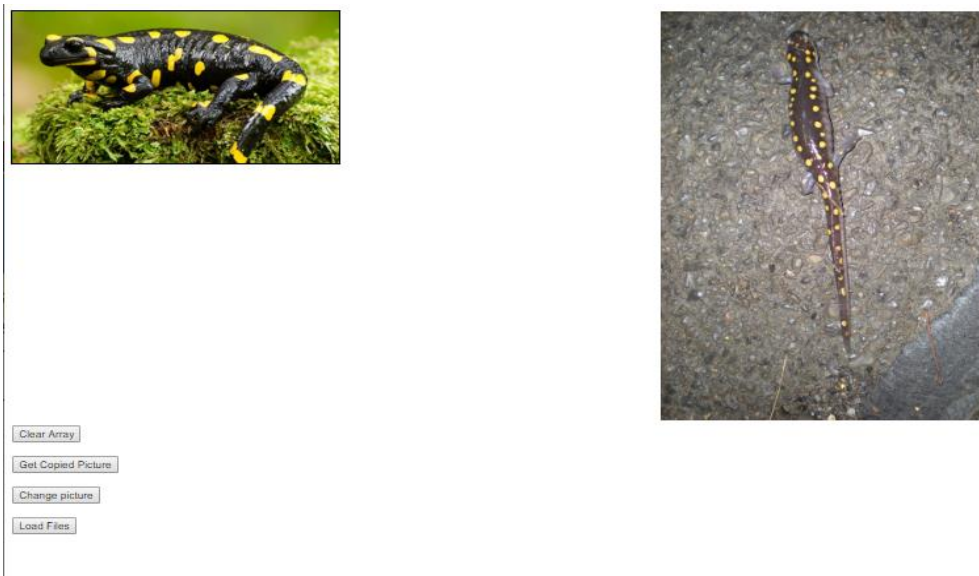


Figure 7: Web Page

The Prototype for our server is a c++ code that calculates a Bezeir of a given by the points clicked on the web application. After this calculation, the program then distorts the image based on straightening the Bezier which is done by re-arranging the location of each pixel in the image. The prototype also has a code that uses opencv to apply fisherfaces to a set of images. All photos must be organized in a specific naming scheme such that matched salamanders have the

same label. The program uses an inputted image (from the web application) as a test subject to compare to a database of photos. It will then either predict the label that matches with the salamander or create a new label for the inputted salamander.

We ran into plenty of errors when testing these parts. Due to many of them being syntax errors, we will only discuss the major ones. A major bug was found in our straightening program. We found that we couldn't take command line arguments because our code for partitioning pixels using ppm used stdin. This caused our outputted image to have a size of zero kb. Instead, we created a new program that wrote command line arguments to a text file and then our straightening code would read in these values from the text file. Also, there were times when negative or values larger than the image coordinate bound would be interpolated by the Bezier function. This also caused our outputted image to have a size of zero kb. We added a system to check if the values are out of the image bounds and set them to their closest bound value, i.e. -100 would be set to 0. However, the resulting image is still not desired and improvements are needed.

The connection between the web application and server did not have any major problems. We only ran into trouble finding a way to tell the web application which picture to display. This was remedied by having the fisherface program output the name of the image it matched with to a text file. The web application then read the image name from that text file.

Our biggest lesson was time management and debugging. We had a lot of frustrating weeks in which simple parts were stretched out due to a small unseen error. We also learned more about Java, C++ and Web application building.

5. Finances

Items	Original Estimate \$	Actuals to Date \$	Estimate to Completion \$	Estimate at Completion \$
JavaScript	0	0	0	0
OpenCV	0	0	0	0
Google Chrome	0	0	0	0
Total \$	0	0	0	0
Reserve	500			500
Funding Limit \$	500			500

Table 1. Project Finances

6. Evaluation and Plans

6.1 Overview

Most of the testing was accomplished by Test/Demonstration. Please reference section 2.3 Design Requirements or WCP28 Project Specification document for the detailed list of requirements. For how the test procedures are performed, please reference WCP28 Intergration and Test Plan document.

6.2 Testing and Results

For a detailed breakdown of each requirement and their result please reference Appendix A: Test Results for Each Requirement. The table below summarizes each requirement and their result.

WCP Requirement ID	Summarized Requirement	Results/What is done	Interpreted Results
WCP28-R-001	At least two pictures inputted	Included in user manual(web application buttons)	User manual
WCP28-R-002	Prompt the user to submit top-down	User manual	User manual
WCP28-R-003	Extract Spots	--	We found this to be unnecessary due to the program being able to match the salamanders without doing this.
WCP28-R-004	Orient and transform picture	The straightening program sets the size of the image and places the salamander in the middle of the image.	The resulting image is extremely skewed and requires further testing.
WCP28-R-005	Scale entire picture	See previous reasoning.	See previous reasoning.
WCP28-R-006	Calculate similarity	We used fisherfaces to	The program tries to predict which

		determine which set of salamanders the image is most similar too.	salamander folder /class it would most likely belong too.
WCP28-R-007	Threshold for similarity	We can set the max distance necessary to determine which class the input is closest too. However, it is not 100% accurate.	Due to the straightening algorithm not working, we were unable to confirm the efficiency of our FisherFaces code.
WCP28-G-001	Program shall be a mobile application	--	We were unable to begin working on a mobile application, but we feel it should work on any mobile device with a web browser.
WCP28-G-002	Allow other perspectives	--	We did not reach this test, however, it was a stretch goal.
WCP28-G-003	Allow immediately captured images	--	We did not reach this test; however, it was a stretch goal.
WCP28-G-004	Create a database for the pictures	The web application has a connection to a local server containing the database through AJAX.	--
WCP28-G-005	Group images of the same salamander in the same folder	When pre-processing an image, we assign a label ≥ 0 to the name of each set of images of a single salamander.	The label determines if the salamander is the same or not.

WCP28-G-006	Create new folder for unmatched salamander	If predicted label is -1, a new folder is created.	--
WCP28-S-001	Run on a desktop computer	--	User Manual.
WCP28-O-001	Java must be updated	--	User Manual.
WCP28-O-002	Computer must have a web browser	--	User Manual.
WCP28-B-001	Identify without solid color background	--	FisherFace algorithm doesn't consider the background.
WCP28-B-002	Provide a manual for program	This is part of the prompts given by the web application.	--

We were on track to have the program fully functional, but due to the straightening algorithm being inaccurate, we must continue testing it. The salamander does not properly get straightened and the resulting image is skewed. This will be remedied in future redesigns.

6.3 Assessment

The strength of our project is found in the framework for the system and our research. Plenty of research was put into creating the method for matching salamander. The use of FisherFaces vastly simplifies the matching process due to it applying the algorithm of the full set of photos. Also, our web application serves as a clear user interface that provides simple buttons that call these algorithms.

Our glaring weakness is within the accuracy of the FisherFace algorithm which is due to the inefficiency of our straightening algorithm. Our straightening program currently produces extremely skewed images. We found that for this project, we can easily hit a roadblock and be setback at any point in development. However, we felt this isn't solely on our design, instead it can be attributed to how delicate software can be as one bug can potentially corrupt your code. We tried to account for this, however, it proved to delay our project enough so that we couldn't get the system to fully function. We have steadily been moving towards a solution to the straightening algorithm and we are hopeful in finding a solution.

6.4 Future Plans

In the future, we would like to complete the debugging of straightening algorithm to eventually fine tune the FisherFaces algorithm. We feel we are extremely close and will not require another group to work on it. It should also be stated that we are willing to finish up during the summer if need be. However, if so, the next group will need to work on debugging the straightening program by figuring out why skewed images are being outputted. This may be due to the values being outputted by the Bezeir curve are outside to bound of the images. Then they should begin testing for a threshold of matching in the FisherFaces code. We are attached this project and hope to see its completion soon.

7. References

[1] Principal Component Analysis 4 Dummies: Eigenvectors, Eigenvalues and Dimension Reduction. Retrieved from <https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction>

https://sebastianraschka.com/Articles/2014_python_lda.html

Appendix A: Test Results for each Algorithm

1. WCP28-R-001: The program shall require at least 2 pictures of salamanders be inputted.
 - Result: This case was not directly tested but solved by design. The FisherFace algorithm requires a set of pre-processed images to function. Therefore, we provide the user with a set of images.
2. WCP28-R-002: The program shall tell the user to only upload a picture that is from the top-down perspective.
WCP28-B-002: The Business (nature preserve) shall be given a guide for program usage.
 - Result: The web application has on screen buttons that will serve as a guide.
3. WCP28-R-003: The program shall be able to separate the salamander and its spots from the rest of the picture.
 - Result: This proved unnecessary for completion of the project as we found that FisherFaces will find the principle components for us.
4. WCP28-R-004: The program shall be able to orient and transform the picture of the salamander to fit the reference picture as best as possible.
WCP28-R-005: The transformations shall scale the entire picture as to not scale each part individually.
 - Result: We used imagemagick to convert jpg images to ppm “\$ magick in.jpg out.ppm” and resized the image “\$ magick in.jpg -resize 500x400! Out.jpg.”
 - We researched cubic spline interpolation to create a Bezier curve that passes through a set of points. This was done by using the Bezier formula: $B(t) = (1 - t)^3P_0 + 3(1-t)t^2P_1 + 3(1-t)t^2P_2 + t^3P_3$.



Figure 8: Bezeir Curve.

- Using ppm.c we were able to manipulate the individual pixel of the image. We were unable to output a straightened salamander. We are unsure of the reason for this result, but we are moving towards a solution. Sometimes the Bezier curve produces points outside the bounds of the image. We are currently looking into this.



Figure 9: output of straightening.

5. WCP28-R-006: The program shall have a computation that demonstrates the similarity between salamander images.
WCP28-R-007: The program shall use this computation to determine whether the salamanders are the same or not.
WCP28-B-001: The program shall be able to identify the salamander without a solid color background.
 - Result: We were unable to find a threshold that would exactly match a salamander due to the straightening algorithm not working. As of now our fisherface code finds the image that is closest in similarity to the inputted image. For more, refer to appendix B.
6. WCP28-G-001: The program should be translated into a mobile application.
WCP28-G-003: The application should allow the user to use their camera to capture a picture of a Salamander.

- Result: The use of a web application means that any device with a web browser should be able to use the application.
7. WCP28-G-002: The application should allow the user to upload different perspectives of the salamander (not just top-down.) { }
- Result: We did not reach this stretch goal. However, we may be able to solve this issue by creating a second set of images from a side perspective. We are unsure if one side of a salamander can be used to match it since the spots cannot all be seen.
8. WCP28-G-004: The application should have a reference database that compares the captured picture with pictures in the database.
- WCP28-G-005: The database should group pictures of the same salamander and have a folder for each salamander.
- WCP28-G-006: The database should create a new folder when a picture could not be matched with any other salamander.
- WCP28-O-001: The system shall require the computer have java updated.
- WCP28-O-002: The system shall require the computer have a web browser.
- Result: We successfully created a web application using javascript. This web application communicates to a local webserver using an ajax request. It also calls the c++ functions necessary for straightening and matching the salamander.

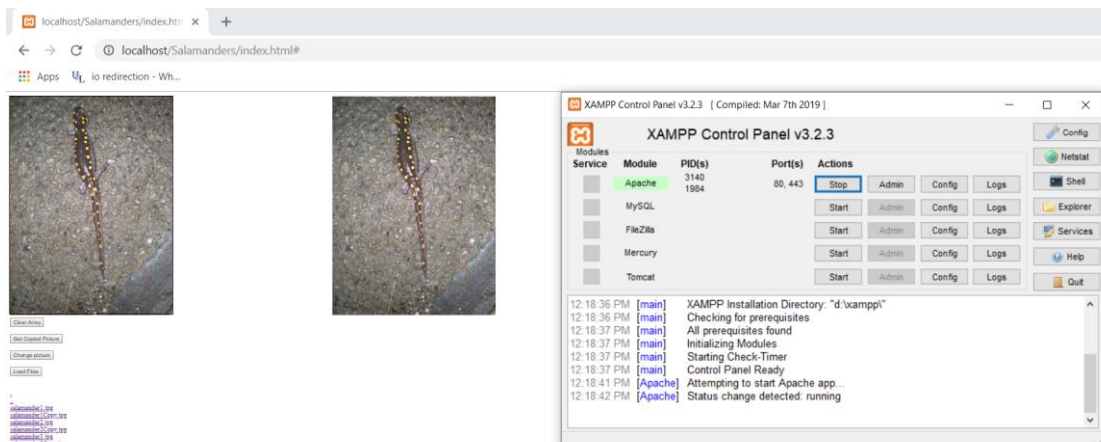
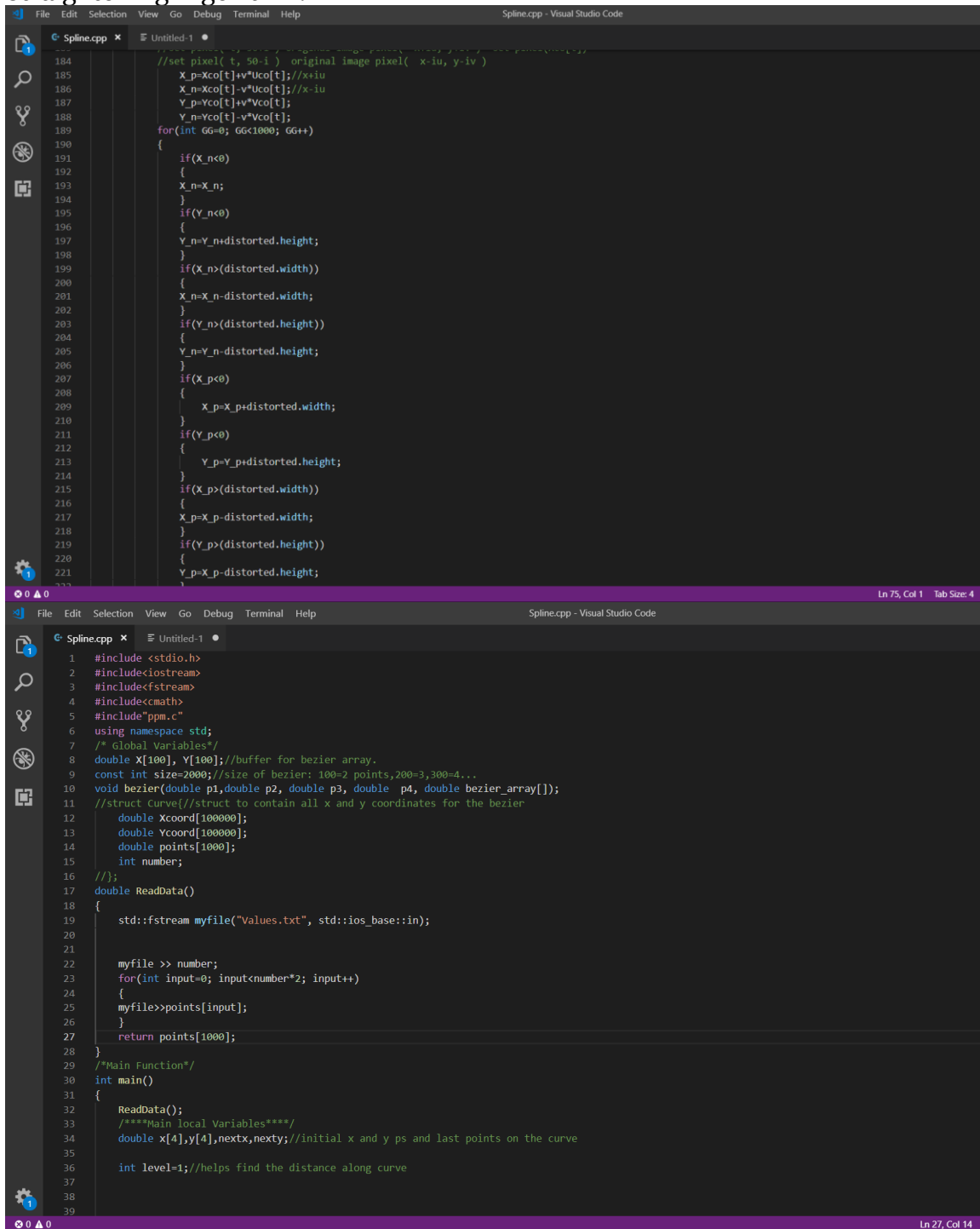


Figure 10: Server

Appendix B: Server Code

Straightening Algorithm:



```
184 //set pixel( t, 50-i ) original image pixel( x-iu, y-iv )
185 X_p=Xco[t]+v*Uco[t]; //x+iu
186 X_n=Xco[t]-v*Uco[t]; //x-iu
187 Y_p=Yco[t]+v*Vco[t];
188 Y_n=Yco[t]-v*Vco[t];
189 for(int GG=0; GG<1000; GG++)
190 {
191     if(X_n<0)
192     {
193         X_n=X_n;
194     }
195     if(Y_n<0)
196     {
197         Y_n=Y_n-distorted.height;
198     }
199     if(X_n>(distorted.width))
200     {
201         X_n=X_n-distorted.width;
202     }
203     if(Y_n>(distorted.height))
204     {
205         Y_n=Y_n-distorted.height;
206     }
207     if(X_p<0)
208     {
209         X_p=X_p-distorted.width;
210     }
211     if(Y_p<0)
212     {
213         Y_p=Y_p-distorted.height;
214     }
215     if(X_p>(distorted.width))
216     {
217         X_p=X_p-distorted.width;
218     }
219     if(Y_p>(distorted.height))
220     {
221         Y_p=Y_p-distorted.height;
222     }
223 }
```

```
1 #include <stdio.h>
2 #include<iostream>
3 #include<fstream>
4 #include<cmath>
5 #include"ppm.c"
6 using namespace std;
7 /* Global Variables*/
8 double X[100], Y[100]; //buffer for bezier array.
9 const int size=2000; //size of bezier: 100=2 points, 200=3, 300=4...
10 void bezier(double p1,double p2, double p3, double p4, double bezier_array[]);
11 //struct curve{//struct to contain all x and y coordinates for the bezier
12     double Xcoord[100000];
13     double Ycoord[100000];
14     double points[1000];
15     int number;
16 };
17 double ReadData()
18 {
19     std::fstream myfile("Values.txt", std::ios_base::in);
20
21     myfile >> number;
22     for(int input=0; input<number*2; input++)
23     {
24         myfile>>points[input];
25     }
26     return points[1000];
27 }
28
29 /*Main Function*/
30 int main()
31 {
32     ReadData();
33     /*****Main local Variables*****/
34     double x[4],y[4],nextx,nexty; //initial x and y ps and last points on the curve
35
36     int level=1; //helps find the distance along curve
37
38
39 }
```

```

39
40 //initial coordinates
41 x[0]=points[0];
42 y[0]=points[1];
43
44 x[1]=50;
45 y[1]=100;
46
47 x[2]=60;
48 y[2]=200;
49
50 x[3]=points[2];
51 y[3]=points[3];
52
53 bezier(x[0], x[1], x[2], x[3],X);
54 bezier(-y[0],-y[1],-y[2],-y[3],Y);
55
56
57 /*assigning values of first bezier to full bezier*/
58 for(curve_count=0;curve_count<100;curve_count++)
59 {
60     Xcoord[curve_count]=X[curve_count];
61     Ycoord[curve_count]=Y[curve_count];
62 }
63
64 double x0,x1,x2,x3;
65 double y0,y1,y2,y3;
66 level=1;
67 for(int Loop=0; Loop<number-2;Loop++)
68 {
69     /*****Continuity*****/
70     /***x***/
71     x1=x[1]; x2=x[2]; x3=x[3];
72     x[0]=x3; /*C0 continuity pn=q0
73             | A(1)=B(0)*/
74     x[1]=2*x3-x2; /*C1 continuity Q1=2*Pn-Pn-1
75             | A'(1)=B'(0)*/
76     x[2]=4*x3-4*x2+x1; /*C2 cont Q2=2*Q1-2*Pn-1-Pn-2 */
77
78     /***y***/
79     y0=y[0]; y1=y[1]; y2=y[2]; y3=y[3];
80     y[0]=y3;
81     y[1]=2*y3-y2;
82     y[2]=4*y3-4*y2+y1;
83
84     x[3]=points[4+2*Loop];
85     y[3]=points[5+2*Loop];
86     //printf("Bezier Curve %d\n",c_number);
87     bezier(x[0],x[1],x[2],x[3],X);
88     bezier(-y[0],-y[1],-y[2],-y[3],Y);
89
90
91     /*****assigning next value for Full b*****/
92     for(curve_count=0;curve_count<100;curve_count++)
93     {
94         Xcoord[100*level+curve_count]=X[curve_count];
95         Ycoord[100*level+curve_count]=Y[curve_count];
96     }
97
98     level++;
99     Xcoord[((100*level)-1)]=x[3];
100     Ycoord[((100*level)-1)]=-y[3];
101 }
102 /*Last Point*/
103
104
105
106
107
108
109 /*Image Part*/
110
111 int A,B,C=0;

```



```
File Edit Selection View Go Debug Terminal Help Spline.cpp - Visual Studio Code
Spline.cpp x Untitled-1
110 int A,B,C=0;
111 image original, distorted;
112 pixel P,Px,Py,P,P_n; //positive pixel and negative pixel
113 double Xco[100],Yco[100],Uco[100],Vco[100],X_p,X_n,Y_p,Y_n;
114 if( get_ppm( &original ) && copy_ppm( original, &distorted ) ) {
115     //copy original image to distorted image
116     //for t=0 to 100 (from 0% along the spline to 100% along the spline, head to tail)
117     for(int t=0;t<100;t++) //0100 71-99
118     {
119         //array of x and why coordinates/ this is not how finding the percentage works, so its a placeholder
120         //Xco[0]=Xcoord[0];
121         //Yco[0]=Ycoord[0];
122         //set <x,y> = spline at position t Question: point at pos t or spline formula at pos t or does that mean the same thing
123         Xco[t]=Xcoord[t*(level)];
124         Yco[t]=-Ycoord[t*(level)];
125         printf("%d %d %f\n",t,Xco[t],-Yco[t]);
126         for(int pls=0; pls<100;pls++)
127         {
128             if((Xco[t]>distorted.width) || (Xco[t]<0))
129             {
130                 if(t*level<(level*100)-100)
131                 {
132                     Xco[t]=Xcoord[t*(level)+pls];
133                 }
134                 else
135                 {
136                     Xco[t]=Xcoord[100*level-1];
137                 }
138             }
139             if((Yco[t]>distorted.height)|| (Yco[t]<0))
140             {
141                 if(t*level<(level*100)-100)
142                 {
143                     Yco[t]=-Ycoord[t*(level)+pls];
144                 }
145                 else
146                 {
147                     Yco[t]=-Ycoord[100*level-1];
148                 }
149             }
150         }
151         /*if(Xco[t]<500 || Yco[t]<400)
152         {
153             if(Xco[t]<0)
154             {
155                 Xco[t]=(Xco[t-1]+5)+Xco[t];
156             }
157             if(Yco[t]<0)
158             {
159                 Yco[t]=(5+Yco[t-1])+Yco[t];
160             }
161         }
162         */
163         Xco[99]=Xco[100*level-1];
164         Yco[99]=-Yco[100*level-1];
165         //set <u,v> = vector perpendicular to spline/u,v= (-dy, dx) dy=y2-y1
166         if(t==0)
167         {
168             Uco[t]=-Yco[t];
169             Vco[t]=-Xco[t];
170         }
171         else
172         {
173             Uco[t]=-Yco[t]+Yco[t-1];
174             Vco[t]=Xco[t]-Xco[t-1];
175         }
176         // printf("%d %d %f\n",t,Xco[t],Yco[t]);
177         //set pixel( t, 50 ) = original image pixel( x, y ) Question I dont know what to do here
178         P=get_pixel(distorted,Xco[t],(distorted.height)/2);
179         put_pixel( distorted, Xco[t], Yco[t], P);
180
181         for(double v=0; v<(distorted.height)/2;v++)
182         { //for i=0..50:
183             //set pixel( t, 50+i ) original image pixel( x+iu, y+iv ) set pixel(Xco[t])
184             //set pixel( t, 50-i ) original image pixel( x-iu, y-iv )
185         }
186     }
187 }
```

```

226         X_p=distorted.height;
227     }
228     if(X_n>distorted.height)
229     {
230         X_p=distorted.height;
231     }
232     if(Y_p>distorted.width)
233     {
234         Y_p=distorted.width;
235     }
236     if(Y_p>distorted.width)
237     {
238         Y_p=distorted.width;
239     }
240     if(X_p<0)
241     {
242         X_p=0;
243     }
244     if(X_n<0)
245     {
246         X_p=0;
247     }
248     if(Y_p<0)
249     {
250         Y_p=0;
251     }
252     if(Y_p<0)
253     {
254         Y_p=0;
255     }
256     P_p=get_pixel(distorted, Xco[t], ((distorted.height)/2)+v);
257     P_n=get_pixel(distorted, Xco[t],((distorted.height)/2)-v);
258     put_pixel(distorted, X_p,Y_p,P_p);
259     put_pixel(distorted,X_n,Y_n,P_n);
260 }
261
262
263
264 }

```

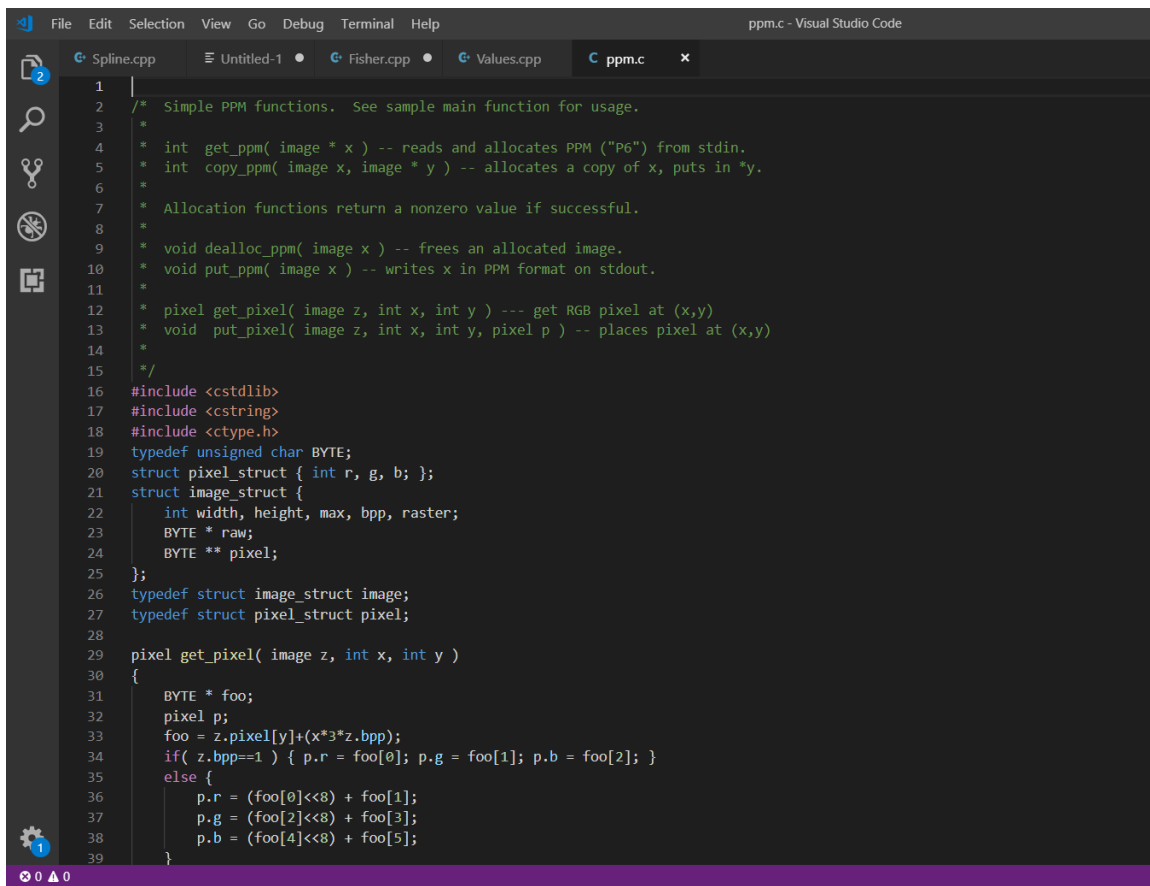
```

256     P_p=get_pixel(distorted, Xco[t], ((distorted.height)/2)+v);
257     P_n=get_pixel(distorted, Xco[t],((distorted.height)/2)-v);
258     put_pixel(distorted, X_p,Y_p,P_p);
259     put_pixel(distorted,X_n,Y_n,P_n);
260 }
261
262
263 }
264
265
266
267     put_ppm( distorted);
268     dealloc_ppm( original);
269     dealloc_ppm( distorted );
270 }
271
272
273 return 0;
274
275
276 }
277
278
279 void bezier(double p1,double p2, double p3, double p4, double bezier_array[])
280 {
281     double t=0;
282     for (t = 0.0; t < 100; t++)
283     { //Bezier Caluculation
284         //B(t) = (1 - t)3P0 + 3(1-t)2tP1 + 3(1-t)t2P2 + t3P3
285         double Ct = pow(1 - (t/100), 3)*p1 + 3 * (t / 100)*pow(1 - (t / 100), 2)*p2 + 3 * pow((t / 100), 2)*(1 - (t / 100))*p3 + pow((t / 100), 3)*p4;
286         bezier_array[(int)t] = Ct; //used for outputting point
287     }
288 }

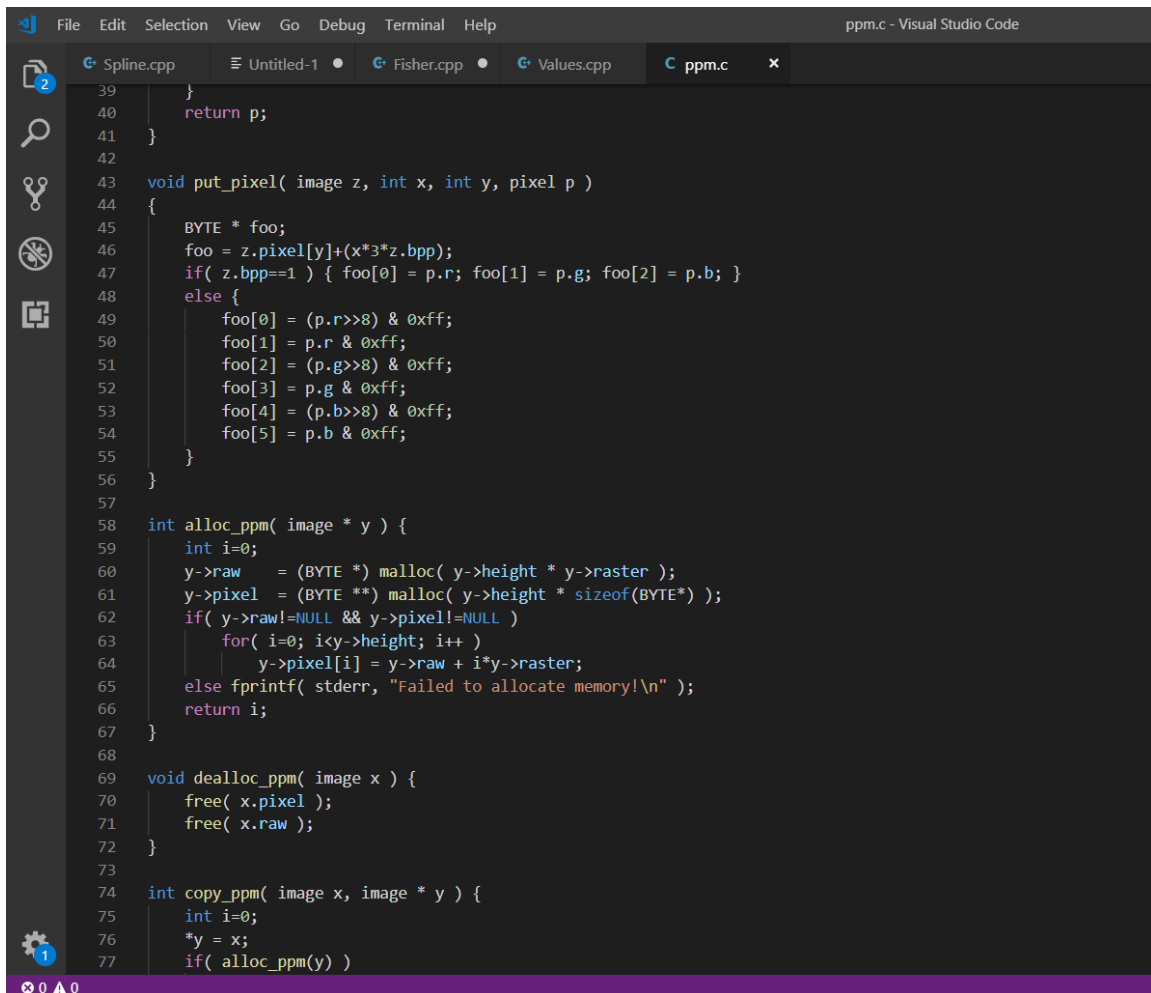
```

Figure 8: Straightening code

PPM:

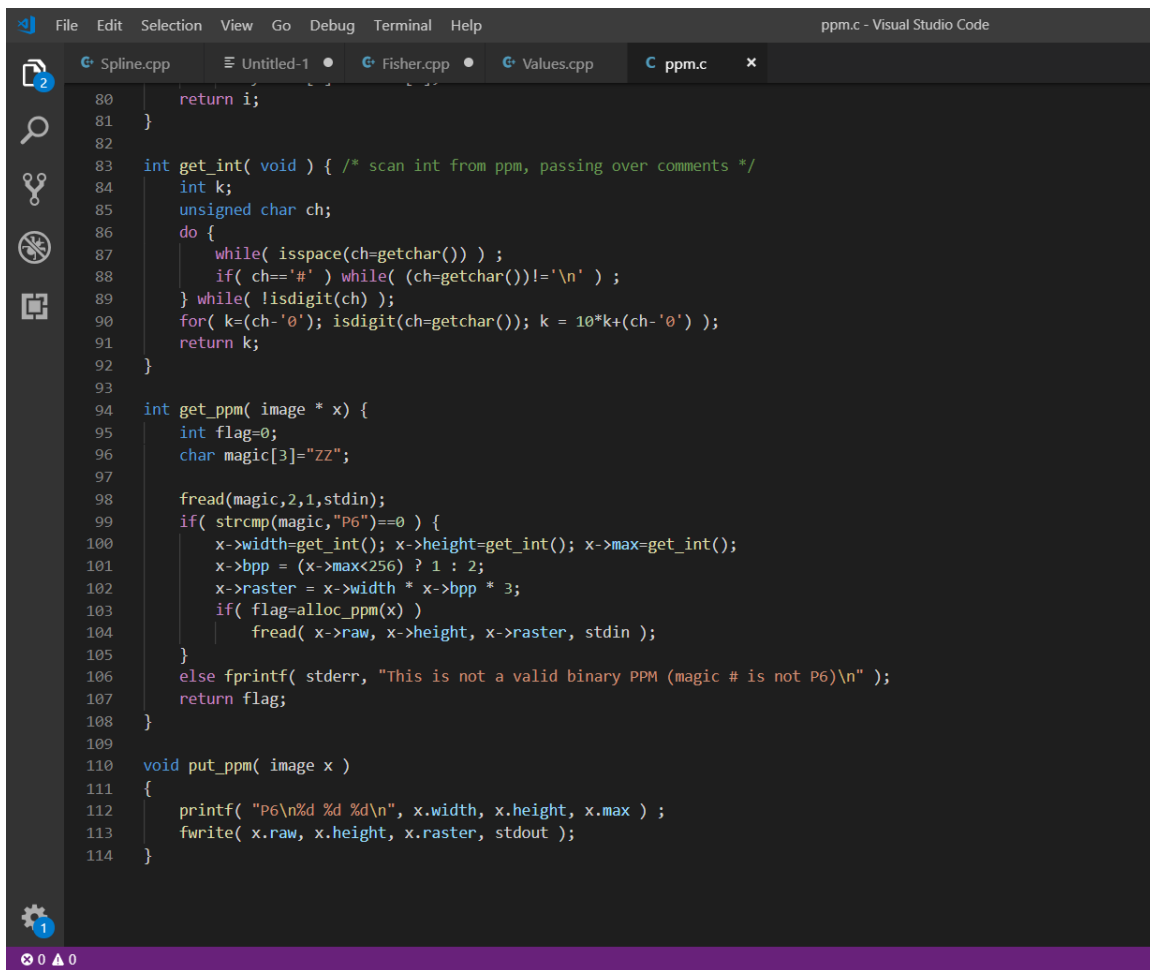


```
1
2 /* Simple PPM functions. See sample main function for usage.
3
4 * int get_ppm( image * x ) -- reads and allocates PPM ("P6") from stdin.
5 * int copy_ppm( image x, image * y ) -- allocates a copy of x, puts in *y.
6
7 * Allocation functions return a nonzero value if successful.
8
9 * void dealloc_ppm( image x ) -- frees an allocated image.
10 * void put_ppm( image x ) -- writes x in PPM format on stdout.
11
12 * pixel get_pixel( image z, int x, int y ) --- get RGB pixel at (x,y)
13 * void put_pixel( image z, int x, int y, pixel p ) -- places pixel at (x,y)
14
15 */
16 #include <stdlib>
17 #include <string>
18 #include <ctype>
19 typedef unsigned char BYTE;
20 struct pixel_struct { int r, g, b; };
21 struct image_struct {
22     int width, height, max, bpp, raster;
23     BYTE * raw;
24     BYTE ** pixel;
25 };
26 typedef struct image_struct image;
27 typedef struct pixel_struct pixel;
28
29 pixel get_pixel( image z, int x, int y )
30 {
31     BYTE * foo;
32     pixel p;
33     foo = z.pixel[y] + (x * 3 * z.bpp);
34     if( z.bpp == 1 ) { p.r = foo[0]; p.g = foo[1]; p.b = foo[2]; }
35     else {
36         p.r = (foo[0] << 8) + foo[1];
37         p.g = (foo[2] << 8) + foo[3];
38         p.b = (foo[4] << 8) + foo[5];
39     }
```



The screenshot shows the Visual Studio Code editor with the file `ppm.c` open. The editor displays the following C code:

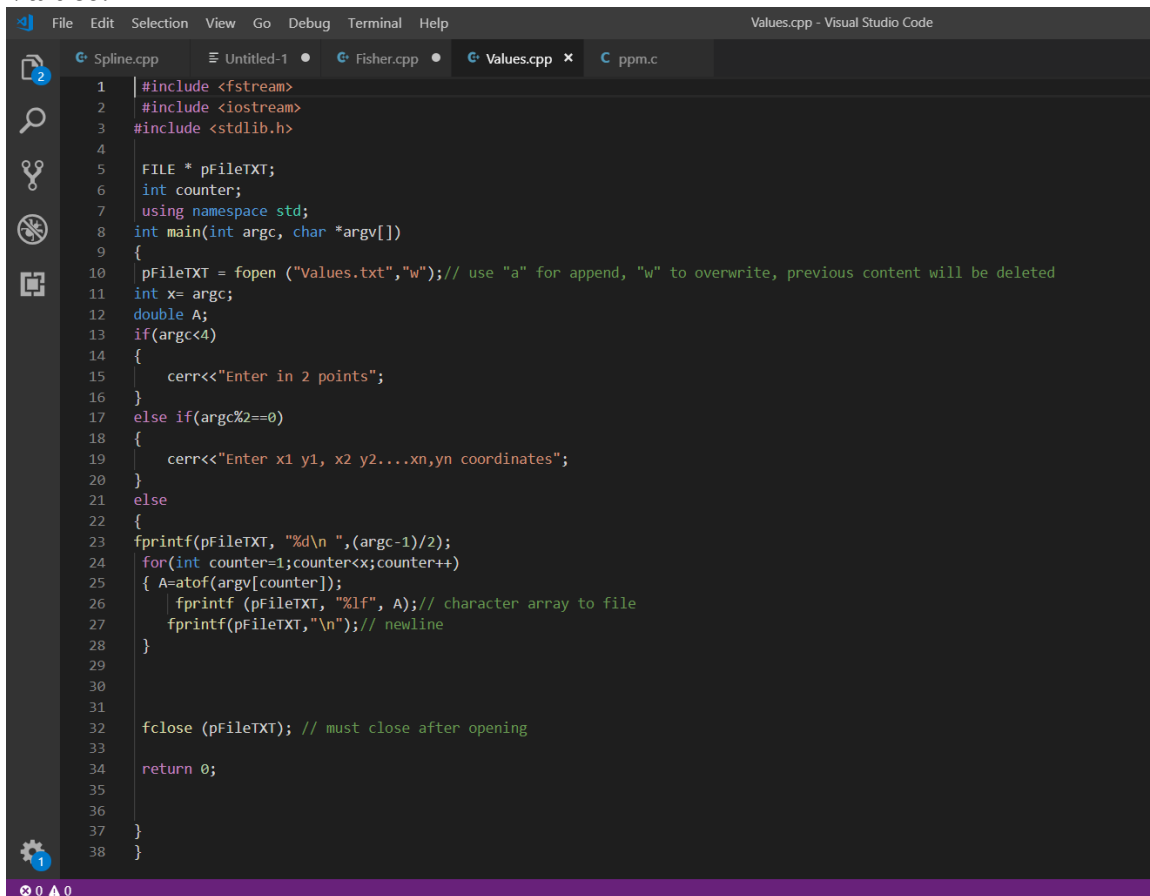
```
39     }
40     return p;
41 }
42
43 void put_pixel( image z, int x, int y, pixel p )
44 {
45     BYTE * foo;
46     foo = z.pixel[y] + (x * 3 * z.bpp);
47     if( z.bpp == 1 ) { foo[0] = p.r; foo[1] = p.g; foo[2] = p.b; }
48     else {
49         foo[0] = (p.r >> 8) & 0xff;
50         foo[1] = p.r & 0xff;
51         foo[2] = (p.g >> 8) & 0xff;
52         foo[3] = p.g & 0xff;
53         foo[4] = (p.b >> 8) & 0xff;
54         foo[5] = p.b & 0xff;
55     }
56 }
57
58 int alloc_ppm( image * y ) {
59     int i=0;
60     y->raw = (BYTE *) malloc( y->height * y->raster );
61     y->pixel = (BYTE **) malloc( y->height * sizeof(BYTE*) );
62     if( y->raw != NULL && y->pixel != NULL )
63         for( i=0; i < y->height; i++ )
64             y->pixel[i] = y->raw + i * y->raster;
65     else fprintf( stderr, "Failed to allocate memory!\n" );
66     return i;
67 }
68
69 void dealloc_ppm( image x ) {
70     free( x.pixel );
71     free( x.raw );
72 }
73
74 int copy_ppm( image x, image * y ) {
75     int i=0;
76     *y = x;
77     if( alloc_ppm(y) )
```



```
80     return i;
81 }
82
83 int get_int( void ) { /* scan int from ppm, passing over comments */
84     int k;
85     unsigned char ch;
86     do {
87         while( isspace(ch=getchar()) );
88         if( ch=='#' ) while( (ch=getchar())!='\n' );
89     } while( !isdigit(ch) );
90     for( k=(ch-'0'); isdigit(ch=getchar()); k = 10*k+(ch-'0') );
91     return k;
92 }
93
94 int get_ppm( image * x ) {
95     int flag=0;
96     char magic[3]="ZZ";
97
98     fread(magic,2,1,stdin);
99     if( strcmp(magic,"P6")==0 ) {
100         x->width=get_int(); x->height=get_int(); x->max=get_int();
101         x->bpp = (x->max<256) ? 1 : 2;
102         x->raster = x->width * x->bpp * 3;
103         if( flag=alloc_ppm(x) )
104             fread( x->raw, x->height, x->raster, stdin );
105     }
106     else fprintf( stderr, "This is not a valid binary PPM (magic # is not P6)\n" );
107     return flag;
108 }
109
110 void put_ppm( image x )
111 {
112     printf( "P6\n%d %d %d\n", x.width, x.height, x.max );
113     fwrite( x.raw, x.height, x.raster, stdout );
114 }
```

Figure 9: PPM code

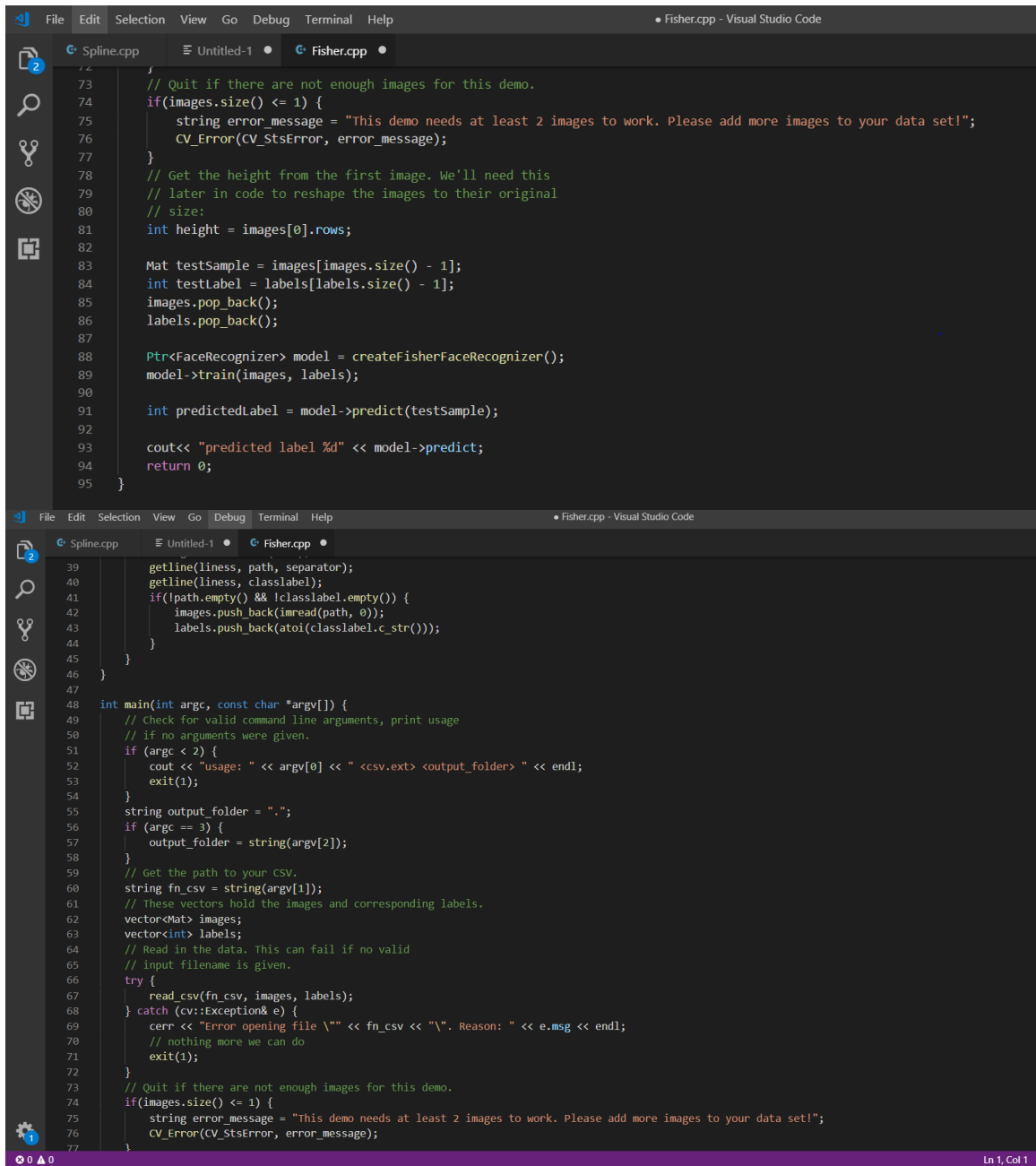
Values:



```
1 | #include <fstream>
2 | #include <iostream>
3 | #include <stdlib.h>
4 |
5 | FILE * pFileTXT;
6 | int counter;
7 | using namespace std;
8 | int main(int argc, char *argv[])
9 | {
10 |     pFileTXT = fopen ("Values.txt","w");// use "a" for append, "w" to overwrite, previous content will be deleted
11 |     int x= argc;
12 |     double A;
13 |     if(argc<4)
14 |     {
15 |         cerr<<"Enter in 2 points";
16 |     }
17 |     else if(argc%2==0)
18 |     {
19 |         cerr<<"Enter x1 y1, x2 y2....xn,yn coordinates";
20 |     }
21 |     else
22 |     {
23 |         fprintf(pFileTXT, "%d\n ",(argc-1)/2);
24 |         for(int counter=1;counter<x;counter++)
25 |         { A=atof(argv[counter]);
26 |             fprintf (pFileTXT, "%lf", A);// character array to file
27 |             fprintf(pFileTXT,"\n");// newline
28 |         }
29 |
30 |
31 |
32 |         fclose (pFileTXT); // must close after opening
33 |
34 |         return 0;
35 |
36 |     }
37 | }
38 | }
```

Figure 10: Values Code

FisherFace Code:



```
73 // Quit if there are not enough images for this demo.
74 if(images.size() <= 1) {
75     string error_message = "This demo needs at least 2 images to work. Please add more images to your data set!";
76     CV_Error(CV_StsError, error_message);
77 }
78 // Get the height from the first image. We'll need this
79 // later in code to reshape the images to their original
80 // size:
81 int height = images[0].rows;
82
83 Mat testSample = images[images.size() - 1];
84 int testLabel = labels[labels.size() - 1];
85 images.pop_back();
86 labels.pop_back();
87
88 Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
89 model->train(images, labels);
90
91 int predictedLabel = model->predict(testSample);
92
93 cout<< "predicted label %d" << model->predict;
94 return 0;
95 }
```

```
39 getline(liness, path, separator);
40 getline(liness, classlabel);
41 if(!path.empty() && !classlabel.empty()) {
42     images.push_back(imread(path, 0));
43     labels.push_back(atoi(classlabel.c_str()));
44 }
45 }
46 }
47
48 int main(int argc, const char *argv[]) {
49     // Check for valid command line arguments, print usage
50     // if no arguments were given.
51     if (argc < 2) {
52         cout << "usage: " << argv[0] << " <csv.ext> <output_folder> " << endl;
53         exit(1);
54     }
55     string output_folder = ".";
56     if (argc == 3) {
57         output_folder = string(argv[2]);
58     }
59     // Get the path to your csv.
60     string fn_csv = string(argv[1]);
61     // These vectors hold the images and corresponding labels.
62     vector<Mat> images;
63     vector<int> labels;
64     // Read in the data. This can fail if no valid
65     // input filename is given.
66     try {
67         read_csv(fn_csv, images, labels);
68     } catch (cv::Exception& e) {
69         cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg << endl;
70         // nothing more we can do
71         exit(1);
72     }
73     // Quit if there are not enough images for this demo.
74     if(images.size() <= 1) {
75         string error_message = "This demo needs at least 2 images to work. Please add more images to your data set!";
76         CV_Error(CV_StsError, error_message);
77 }
```

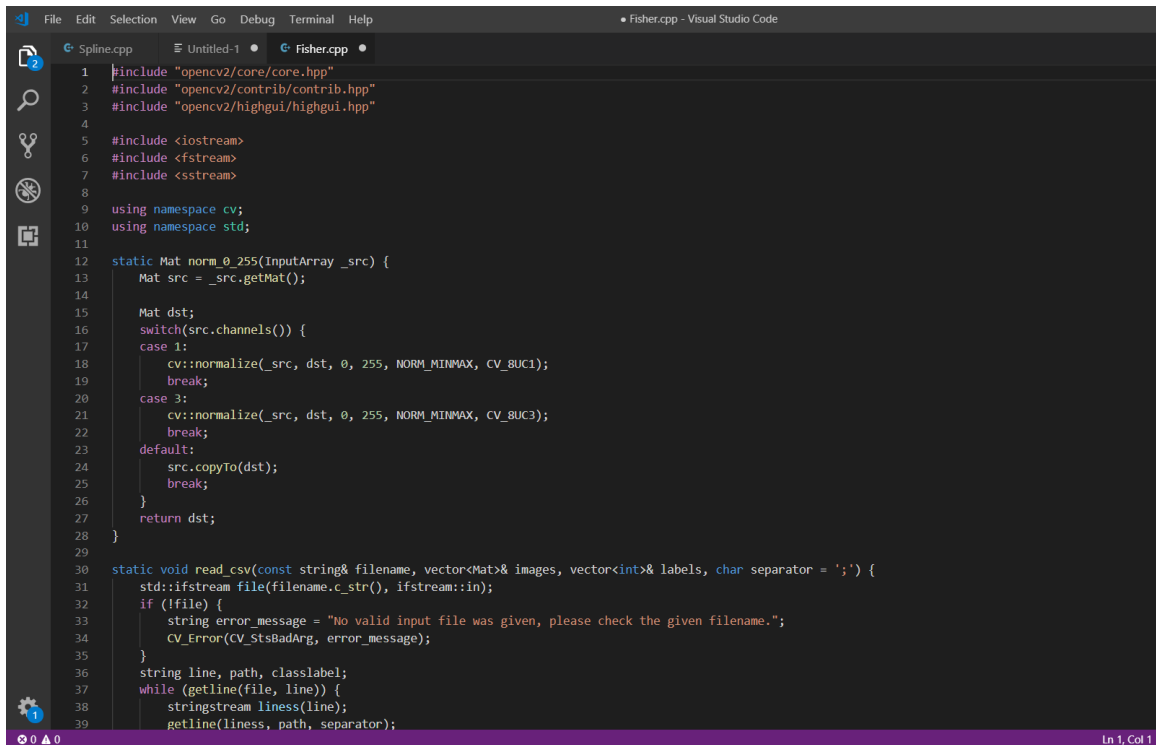


Figure 11: FisherFaces code

Appendix C: Web application code

Picture Coordinates:

```

// https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server
// https://openclassrooms.com/en/courses/3523261-use-javascript-in-your-web-projects/3759261-make-your-first-ajax-request
//https://stackoverflow.com/questions/35613799/change-image-object-dimensions-width-and-height
//py -m http.server and then go to localhost:8000
//c:\inetpub\wwwroot

```

```

var xCoords = [];
var yCoords = [];

```

```

var myImg = document.getElementById("sal1");
var canvas = document.createElement("canvas");
var imgName;

```



```

function hi(){
    console.log("HI");
}

function setCanvas(e){

    var height = myImg.height;
    var width = myImg.width;

    canvas.id = "myCanvas";
    canvas.width = width;
    canvas.height = height;
    canvas.style.position = "absolute";
    // canvas.style.right = 10;
    // canvas.style.top = 10;
    canvas.style.border = "1px solid";

    var context = canvas.getContext("2d");
    // context.drawImage(myImg, 0, 0, 400, 500);

    canvas.addEventListener("click", getCoordinates);

    document.body.appendChild(canvas);
    // var ctx = canvas.getContext("2d");
    // ctx.moveTo(0, 0);
    // ctx.lineTo(200, 100);
    // ctx.strokeStyle = "rgb(255, 0, 0)";
    // ctx.stroke();

}

function ajaxRequest(){
    var request = new XMLHttpRequest();
    request.onreadystatechange = function(){
        if (this.readyState == 4 && this.status == 200){
            // var changeText = document.getElementsByClassName("ajaxText");
            // changeText[0].innerHTML = this.responseText;
            sal2.src = this.responseText;
            // console.log(this.responseText);
        }
    };
    var imageName = myImg.src;
    request.open("GET", "http://localhost/Salamanders/test.php?x=" +
imageName, true);
    // request.open("GET",
"http://localhost/Salamanders/test.php?x=hi&y=hello", true);
    //request.open("GET", "text.html", true);

```

```

    request.send();
}

function loadFiles(){
    var request = new XMLHttpRequest();
    request.onreadystatechange = function(){
        if (this.readyState == 4 && this.status == 200){
            //var changeText = document.getElementsByClassName("ajaxText2");
            // changeText[0].innerHTML = this.responseText;
            var dataArray = JSON.parse((this.responseText));

            console.log(dataArray);
            // console.log(dataArray.length);
            //console.log(dataArray[0]);

            postFiles(dataArray);
            /*var changeText = document.getElementsByClassName("ajaxText2");
            changeText[0].innerHTML = picArray;*/
        }
    };
    request.open("GET", "getPics.php", true);
    request.send();
}

```

```

function postFiles(dataArray){

    var filesList = document.getElementById("list");
    while(filesList.firstChild){
        filesList.removeChild(filesList.firstChild);
        // filesList.removeChild(lis[0]);
    }

    for (var i = 0; i < dataArray.length; i++){
        //console.log(i);
        var a = document.createElement("a");
        a.setAttribute("href", "#");

        var imageName = dataArray[i];
        a.addEventListener("click", function(e){
            // changeImageName('Images/' + imageName);
            //console.log("IH: " + imageName);
            //changePicture("Images/" + imageName);
            //changePicture();

            if (e.target && e.target.nodeName == "LI"){
                // console.log(e.target.id);
                changePicture("Images/" + String(e.target.id));
            }
        });
    }
}

```

```

        }
    }
);
var node = document.createElement("LI");
node.setAttribute("id", String(dataArray[i]));
var textNode = document.createTextNode(String(dataArray[i]));
node.appendChild(textNode);
a.appendChild(node);
filesList.appendChild(a);
}

/* var filesList2 = document.getElementById("list2");
var a2 = document.createElement("a");
a2.setAttribute("href", "#");

var imageName2 = dataArray[dataArray.length - 1];
console.log(imageName2);
a2.addEventListener("click", function(){
    // changeImageName('Images/' + imageName);
    console.log("IH: " + imageName2);
    // changePicture("Images/" + imageName);
    //changePicture();
})
);
var node2 = document.createElement("LI");
var textNode2 = document.createTextNode(String(dataArray[i]));
node2.appendChild(textNode2);
a2.appendChild(node2);
filesList2.appendChild(a2);*/
}

//CHANGE PICTURE BUTTON WON'T WORK ANYMORE because
//name of the image source is not hardcoded anymore
function changePicture(imageName){

    // console.log("height: " + myImg.height);
    // console.log("width: " + myImg.width);

    // console.log("WHY:" + imageName);
    var image = document.getElementById("sal1");
    // console.log(image.height);

    var img = new Image();
    var newHeight = 0;
    var newWidth = 0;
    img.onload = getSize;
    img.src = imageName;

```

```

function getSize(){w = img.width;
    h = img.height;
    var percentChange = 400/w;
    var newWidth = percentChange * h;
    myImg.style.width = "400px"
    myImg.style.height = newWidth.toString() + "px";
    clearArray();
    setCanvas();
}

image.src = imageName;

}

function clearArray(){

    var textWords = document.getElementsByClassName("text");
    textWords[0].innerHTML = "";

    xCoords = [];
    yCoords = [];

    var context = canvas.getContext("2d");
    context.clearRect(0, 0, canvas.width, canvas.height);

    // myImg.src = "Images/salamander2.jpg";

}

function drawLine(e){

    var length = xCoords.length;

    if (length > 1){
        var x = xCoords[length - 2];
        var y = yCoords[length - 2];
        //console.log("first x: " + x);
        //console.log("first y: " + y);

        var ctx = canvas.getContext("2d");

        ctx.beginPath();
        ctx.moveTo(x, y);

        x = xCoords[length - 1];
        y = yCoords[length - 1];
    }
}

```

```

/* console.log("second x: " + x);
   console.log("second y: " + y);*/

ctx.lineTo(x, y);
ctx.strokeStyle = "rgb(255, 255, 255)";

ctx.closePath();
ctx.stroke();

}

}

function findPosition(oElement){
    if(typeof( oElement.offsetParent ) != "undefined"){
        for(var posX = 0, posY = 0; oElement; oElement = oElement.offsetParent){
            posX += oElement.offsetLeft;
            posY += oElement.offsetTop;
        }
        return [ posX, posY ];
    }
    else{
        return [ oElement.x, oElement.y ];
    }
}

function getCoordinates(e){

    // console.log("HI");

    var PosX = 0;
    var PosY = 0;
    var ImgPos;
    ImgPos = findPosition(myImg);
    if (!e) var e = window.event;
    if (e.pageX || e.pageY){
        PosX = e.pageX;
        PosY = e.pageY;
    }
    else if (e.clientX || e.clientY){
        PosX = e.clientX + document.body.scrollLeft
            + document.documentElement.scrollLeft;
        PosY = e.clientY + document.body.scrollTop
            + document.documentElement.scrollTop;
    }
    PosX = PosX - ImgPos[0];
    PosY = PosY - ImgPos[1];
}

```

```

xCoords.push(PosX);
yCoords.push(PosY);

var h = document.createElement("H1");
/* var newText = document.createTextNode("\nX: " + PosX +
    " Y: " + PosY);
h.appendChild(newText);*/

var textWords = document.getElementsByClassName("text");
textWords[0].appendChild(h);

drawLine();

}

// var h = document.createElement("H1");
// var t = document.createTextNode("X: " + PosX + " Y: " + PosY);
// h.style.fontSize = "20px";
// h.appendChild(t);
// document.body.appendChild(h);

```

Index:

```

<?php
    if (!empty($_SERVER['HTTPS']) && ('on' == $_SERVER['HTTPS'])) {
        $uri = 'https://';
    } else {
        $uri = 'http://';
    }
    $uri .= $_SERVER['HTTP_HOST'];
    header('Location: '.$uri.'/dashboard/');
    exit;
?>

```

Get Pictures:

```

<?php
$images = (scandir('Images'));
echo (json_encode($images));
?>

```