

Microscope Manipulator

Joe Hippensteel – Team Leader
Evan Rogers – Communications
Chris Webster – BSAC
John Baran – BWIG

Client: Robert Jeraj, PhD.
Advisor: Willis Tompkins, PhD.

December 6th, 2005

Abstract: There has been an insurgence of research involving Zebrafish embryos within the last few years, primarily due to the ease with which an observer can view changes in their internal physiology caused by external and internal factors. It has been proposed that a digitally controlled micromanipulator be constructed to create a fast scanning system to image and irradiate a large sample of Zebrafish embryos efficiently. It is required that the stage not exceed 6 cm in height and have a minimum stepping resolution of 200 μm . In addition, the software necessary to operate the stage and recognize the individual fish must be developed.

§1. Problem Definition

Accurate scanning and re-positioning of samples under a dissecting microscope is inefficient with the equipment currently available to the client. The current stage is too large and the imaging and positioning hardware and software are outdated. The primary goal of this project is to develop a fused digitally interfaced stage and custom imaging technique that can systematically do the following: scan a sample of zebra fish, analyze the fused images, store the positions of each zebra fish and reposition the sample to the localized positions. An example of a micromanipulator system is shown in Fig. 1.



Fig. 1 – Micromanipulator set-up with intergrated analog monitoring equipment.

§2. Motivation

Zebrafish as Early Vertebrate Model:

Zebrafish embryos are becoming more popular in the scientific community as vertebrate models. Zebrafish embryos are transparent during their embryonic stage and develop *ex utero* (Xu). This transparency allows for observation of organ and skeletal development on the cellular level *in vivo*. This is preferable to the researchers, as they are able to pinpoint and monitor a specific area or multiple areas of interest. The Zebrafish genome project is nearing completion, and will afford the scientific community many opportunities for studying this vertebrate development.

The client specifically studies the inflammatory response of Zebrafish cells due to radiation exposure, and its relationship to cell apoptosis (programmed cell death). The Zebrafish is irradiated with approximately 50 keV of non-ionizing high energy photons. The mechanisms of inflammatory response of the Zebrafish may be elucidated as a result of this cutting edge technique, which would ultimately provide insight into the mechanisms of radiation poisoning in other vertebrates.

Cell Apoptosis

Apoptosis is the programmed destruction of cells by their own lysosomal enzymes (Campbell and Reece, 2002). The exact pathway for human apoptosis is not currently known and is an active area of research. The cell receives a signal to die from various signaling regions of the body (e.g. central nervous system, paracrine system and endocrine system) which initiate leakage of suicide proteins from the outer membrane of the mitochondria. Post mortem, the remaining pieces of the dead cell are then engulfed and digested by neighboring cells, allowing the embryo to stay free of the harmful proteins. An example of the failure of apoptosis in the morphogenesis of the human is webbed fingers or toes (Campbell and Reece, 2002). Apoptosis has become very prevalent in current cancer research, due to tumor cells' resistance to this mechanism.

§3. Client Requirements

The table must move freely in the XY plane and have a step precision of at least 200 μm . In order to clear the microscope lens, it cannot exceed 6 cm in height. It must be large enough to hold a 6 cm diameter Petri dish and have a range of motion that is wide enough

to scan the entire dish. Because each Zebrafish will be irradiated, the table must withstand 50 keV of ionizing radiation without demonstrating adverse effects or retaining any radioactivity. There should be no limit on the number of uses the micromanipulator can endure.

The camera used will be purchased and integrated with the microscope. It must have sufficient resolution to see the effects of the radiation on individual fish. It must also mount easily and securely on the eyepiece and should be directly connected to the operating computer for online analysis.

The imaging and positioning software must be integrated, automated, and PC compatible. Image processing software will be required to localize all embryos in a sample so that they can be irradiated one at a time. All components should be interfaced with the PC using USB or Firewire technology, but a serial port connection would suffice.

The image processing software should be designed to seek out the eyes of the fish and determine the orientation of their bodies. This should be accomplished using Matlab version 7.0. Once each eye is located, the positioning software should work with the stage controller to orient each individual fish under the radiation source so that it can receive radiation. Care must be taken when moving the stage so that fish are not shifted from their initial positions. The stage translation software should be programmed in Visual Basic.

Because the amount and area of radiation must remain constant, a device should be built to position the source above the sample. It must display an accurate measurement of the distance between the source and the sample of Zebrafish. The stand should be made out of a composite material and should not be attached to the microscope or stage.

§4. Design

Image Processing Software

One of the primary components of the automated Zebrafish irradiation system is the digital detection of the Zebrafish embryos (see Fig. 2). A program had been written by Dr. Robert Pyzalski to decipher the location and orientation of the individual Zebrafish, which could theoretically be used to irradiate the individual fish in a time optimized fashion. This was accomplished on a Linux based system using C code, interfaced with an analog camera set-up. Due to the necessity of developing a similar software suite for a PC environment with digital imaging techniques, it has been determined that Matlab version 7.0 will be the appropriate programming language because of accessibility and the team's knowledge of it (final program with graphical user interface syntax can be found in the appendix). The primary goal of the program is to output a file containing an array of pixel values that can be transferred into a Visual Basic program which will control the stage.

Image Processing Methodology

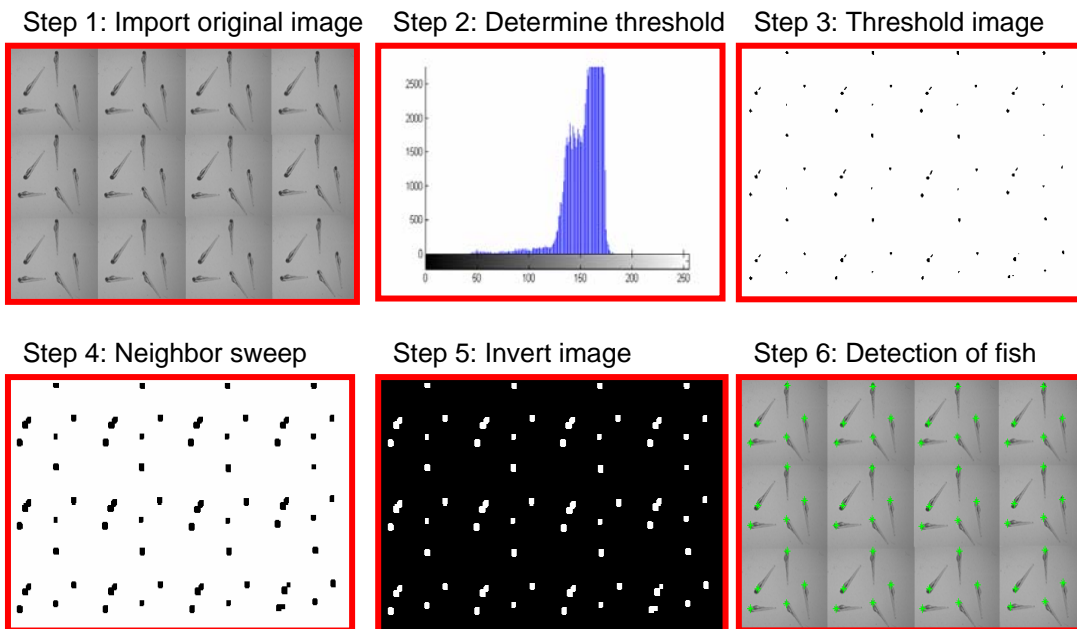


Fig. 2 – Image processing methodology realized using a custom Matlab 7.0 program named *fisherman*.

After reviewing Dr. Pyzalski's code, it was determined that the initial step in the image processing process was to minimize the noise levels inherent in the images captured. Using a camera supplied by Manuel Rodriguez (approximately 1 Megapixel resolution digital camera), photographs were taken of a representative sample of Zebrafish embryos to be used for stage translation speed optimization and off-line image processing software development. These pictures were imported into Matlab in an attempt to accomplish the required image processing (Note: both gray scale and RGB valued images are compatible with the software). A built-in filter (`medfilt2`) is used to sharpen the image and create

better edges. Several techniques for this portion were tested to determine optimal settings.

Thereafter, substantial research was done to determine image recognition possibilities. Masking techniques were researched based upon the Linux based C code. No tutorials were found for masking techniques in Matlab that were comprehensible by the group. The image processing toolbox tutorial was reviewed in detail to determine other available options. The final choice was a single command called *bwlabel* which allows for recognition of binary bodies, which in turn creates an array for each body including centroid, orientation, and area values, amongst others. This array can be accessed using the *regionprops* command. In order for the image to be compatible with this command, the “bodies” (Zebrafish embryos) of interest need to be transformed into binary black and white images, where the white portions are the areas of interest.

The initial step is to perform simple thresholding techniques on the loaded image using a variable defining the pixel values of interest (ranging from 0 to approximately 250, black to white respectively; Fig. 2 step 2 is the images pixel value histogram). Since the regions of interest are nearer the black end of the spectrum, the threshold settings are typically set at 95, turning all pixel values above this value to white, and those below to black. Thresholding leaves the user with small black blotches throughout the screen, as seen in Fig. 2 Step 3, which need to be connected in order to be recognized as individual fish.

The expansion of individual pixels was initially accomplished using a neighboring command, which was highly inefficient due to the long processing time of this command. An alternative was found called *colfilt*, which allows for this expansion using a sweeping process nearly instantaneously. This command requires a set of input values defining the extent of pixel expansion, which generally ranges from 3 to 10. This process connects the individual dots exhibited in Fig. 2 Step 3 and transforms the image to appear as is seen in Fig. 2 Step 4. As aforementioned, the bodies are required to be white (binary black/white form) in order to be recognized by the *bwlabel* function. This is done using a simple nested *for-if* loop inverting the picture and the command *im2bw*.

The aspects of the Zebrafish “bodies” that are processed are the area of each body and the location of its centroid. In the future, it will be possible to determine the orientation of the fish using the orientation command. Unfortunately, the simple thresholding technique does not allow for use of the orientation command due to the lack of distinction between minor and major axes. The localized “bodies” are analyzed individually to negate any which are too large or too small to be fish. As the data is reviewed by the code, the centroids of the bodies are stored in an array that defines the centroids of each body based upon the pixel values of the image. In order to verify that the fish are localized correctly, the centroid values are plotted over the image, shown in Fig. 2 Step 6, which can be reviewed by the user prior to irradiation.

In order to increase user-friendliness, a graphical interface was developed that facilitates user input to iteratively determine the location of the fish. The particular values that can be set are filename, threshold level, maximum and minimum pixel size, neighboring values (*colfilt* command), and displayed images. In addition, the user can define whether he would like an array to be output including the located centroid values or if he would prefer a simple trial run. The standard output array is named “array.txt” and is written to the directory where the program was located. This output filename and directory can be easily

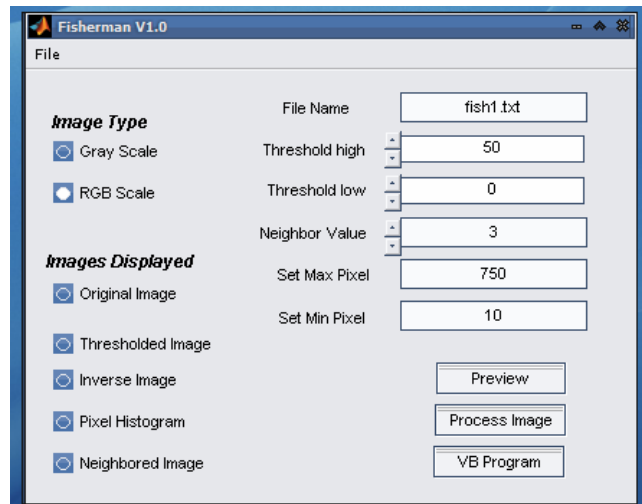


Fig. 3 – *Fisherman* program developed by microscope manipulator team for simple image recognition. This program was developed in Matlab using simple image processing techniques.

customized to suite the users needs. The program, aptly named *fisherman*, was compiled using standard compiling techniques and can be used on any computer with Matlab 7.0 and includes a simple GUI shown in Fig. 3.

Stage Translation Software Interface

The program the client was using was built using Visual Basic 6.0, and integrated a user interface with stepper motor motion. First, an image was saved from the camera, which was attached to the microscope setup, and loaded into the program. This provided the user with a way to manipulate the stage using the image of the Zebrafish. The next step in the setup was to calibrate the program. This is necessary because the stage needs

metric measurements to be inputted through the program. For this to be done, image values of measurement, or pixels, needed to be changed into centimeters, allowing the stage to move. Once the setup was complete, clicking on the loaded picture caused the stage to move to that specific point to a location directly over the radiation source.

The program had a couple of problems which needed to be addressed in order to deal with the requirements of the client. One was to make the process automated, eliminating the need to click on each point in order to irradiate the entire dish. The first problem considered was the ability to import specific points for irradiation from an outside source. It was also necessary to move to those specific points, but at the same time make sure sufficient time was available to cover the whole dish. It was decided to use the code which was provided as a base and add code as deemed fit. For the first problem, a text file was used as a means of transporting the information from one program to the other. After the files are loaded into the program, it is easy to integrate another piece of movement code to allow the text file radiation points to be used.

Once the specifications were received a problem arose because of the system incompatibility issues. The current program was built using Visual Basic 6.0, an earlier version of Visual Basic, and the only program available was Visual Basic .Net. The basic setups of the programming environments are similar, but the syntax which is used in the two environments is very different, making it difficult to translate from one language to another. Finally, it was decided to go ahead and create a piece of code to open a text file

in Visual Basic.Net and hope to be able to integrate the existing code into the new program.

The first step was to create a program which would allow the opening of a text file into a program and allow the information to be manipulated. The information was brought into an array by reading the text file one line at a time. An array is basically a list which can store different kinds of information. Once the text file could be read one line at a time, the next challenge was to be able to split the individual lines up and read the individual numbers in the line. This was accomplished using Visual Basic's built in functions. There then was an array of the individual points which needed to be radiated. All of the previously existing code needed to be converted into Visual Basic.Net in order to have a fully functional system. Unfortunately, the existing program was much more difficult to integrate than previously

thought. After more consulting with the clients, it was concluded that the best way to create this final working product was to convert all of the open code into Visual Basic 6.0, ultimately making it easier to combine it with the existing code. After some trial and error, the program finally opened text

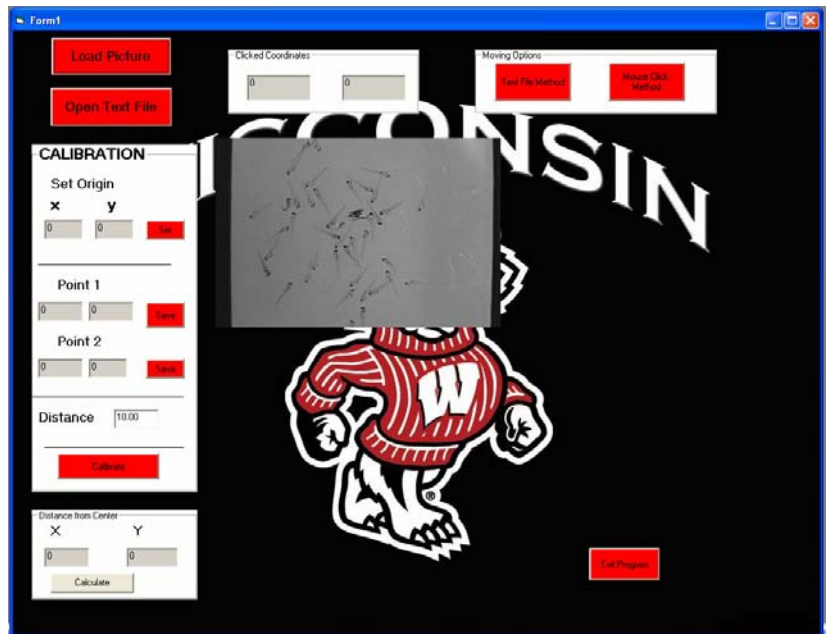


Fig. 4 – Software developed in Visual Basic for stage manipulation. This was adapted from Manuel Rodriguez's initial VB 6.0 software.

files in Visual Basic 6.0.

Once all of the information was brought into the program, the goal was to use the information to move the stage. The best way was to use the existing code to move the stage and add extra code to allow the use of the new open code. The move button works by clicking on the “move using text file” button, which in turn moves the stage to the next coordinate in the sequence. The challenging aspect of writing this piece of code was that it had to be written on a computer other than the one connected to the microscope stage, and it was difficult knowing if it would work on both. The code was written as universally as possible, and eventually had to be extensively tested on the functioning machine. After a few minor problems were worked out, the code was fully functional (See Fig. 4 for finished program).

Microirradiator Stabilizer

The microirradiator needs to be held at an exact distance from the sample of Zebrafish. The dosage given to a sample is highly dependent on the distance the x-rays travel. The equation is governed by the inverse logarithmic square of the distance. The holder piece only needs to have fine adjustment in a range of approximately 2 cm. A coarse adjustment is sufficient to set the irradiator in the right neighborhood of the sample.

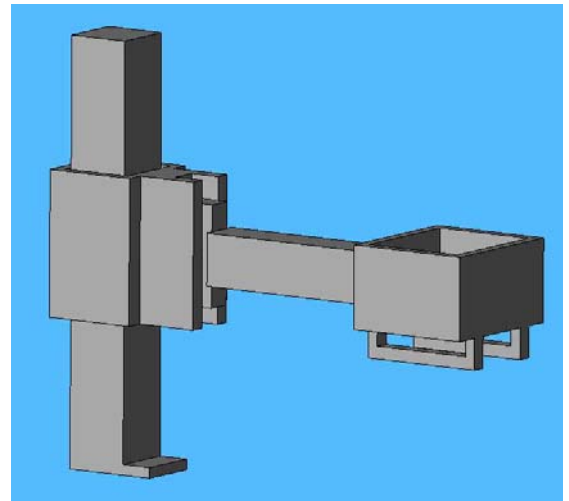


Fig. 5 – Proposed assembly for Microirradiator holder. The assembly consists of three parts, a base, bracket and a basket piece.

The proposed holder consists of a three part assembly. The base of the holder is shown in Fig. 5. The base consists of a 20 cm tall post and a small 3 cm lip behind it. The lip allows the post to be screwed into the current apparatus.

The bracket piece in Fig. 5 allows for the coarse adjustment of the irradiator location. The bracket slides onto the post and screws into the side wherever the user sees fit. The track on the side of the bracket piece will allow for fine adjustment of the distance.

The basket piece of the assembly holds the irradiator and provides precise distance measurements. The basket consists of a rectangular box with two braces on the bottom to prevent the irradiator from slipping through. Before building the actual product, it must be confirmed that the braces will not be in the path of the collimator. If this is the case, the braces will deflect the rays and the basket will be doing more harm than good. The track on the smaller post of this piece will allow the user to fine tune the distance of the holder using a technique similar to a caliper. Gear ratios will be set such that every turn of a knob is equivalent to a tenth of a millimeter.

Recommended System

After the experiments trial phase has begun to show results, a more technologically advanced set of equipment would be helpful to gather improved data and to aid in overall progress. The most important components of this system are the microscope, the camera, the stage and the controller (Fig. 6 shows these components).

The Motic K400 Stereomicroscope is recommended for its versatility, quality, and reasonable cost. It has a 50x maximum focus and a tall, adjustable column. It is relatively easy to come by and is being used in many labs around the US and in Europe.



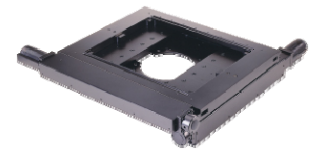
Motic™ K400
List: \$1,075

The Scion CFW-1308C Camera comes very highly recommended by reviewers and current owners. It is well priced compared to other cameras of similar quality. It is 1.3 megapixels and connects via IEEE-1394 Firewire. Higher resolution cameras exist, but 1.3 megapixels far exceeds the current system and should be more than sufficient to continue experimentation. A C-mount adaptor is required to mount the camera on to the microscope.



Scion™ CFW-1308C
List: \$1,695

Sutter™ is one of the leading producers of biomedical and research equipment. The MT-1000 Translator System and MPC-200-ROE Controller are their midline translator and controller bundle. This combination will achieve the efficiency and precision that is required for this experiment, but will remain low in cost compared to other, higher end utilities.



Sutter™ MT-1000
List: \$5,900

Fig. 6 – Commercial microscope, camera, and stage suggested.

List Prices	
Motic™ K400	\$1,075
Scion™ CFW-1308C	\$1,695
C-Mount Lens Adapter	\$24.00
Sutter™ MT-1000	\$5,900
Sutter™ MPC-200-ROE	\$3,500
Total	\$12,194

§5. Testing

Speed Optimization

The stage has a spectrum of operating speed ranging from A1 to A127, lowest to highest respectively. In order to determine what speed was optimal with a minimum observed movement, three speeds were tested: A1, A50, and A127. The stage speed was set to these values in turn using the Visual Basic program developed by Manuel Rodriguez. A picture was taken and then the stage was moved randomly. The stage was returned to its original position. A picture was taken after its return. To determine any movement using these operating modalities, the

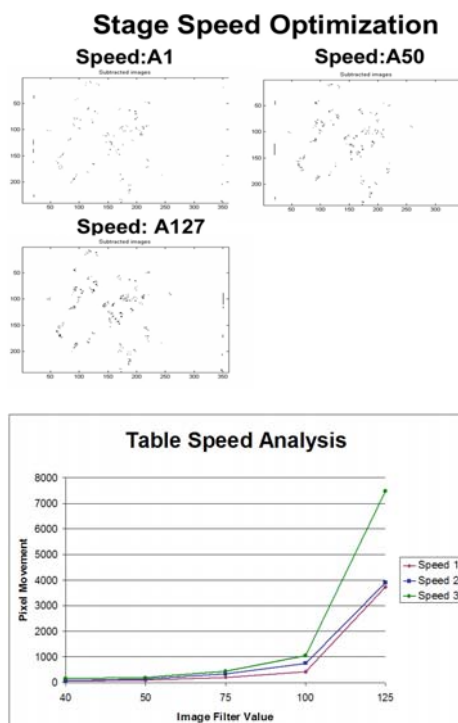


Fig. 7 – The top picture is the amount of displaced pixels at each speed. A graph of displaced pixels is shown below.

images were imported into Matlab. Both initial and post-movement images were thresholded at a constant level as outlined in the *Image Processing Software* section. The two images were subtracted from one another and the number of pixel values that were different was quantified. It was demonstrated (Fig. 7) that A1 exhibited some movement and that the A50 and A127 settings demonstrated higher pixel value displacement values qualitatively and quantitatively. Based upon the time constraint of one hour set by the client, it was decided that A1 would be sufficient for the irradiation process in order to minimize fish movement.

Visual Basic Program Testing

The current Visual Basic program has been tested with the client's microscope stage. The program accurately moves the stage to the specific points which need to be radiated.

§7. Pros and Cons

Due to the complexity of this project and the disparity of the individual required products, a linear pursuance was implemented for each portion of the design. Therefore, it is difficult to delineate all of the pros and cons of the current system based upon the limitations of the gathered information on image processing, stage operation, and irradiator support. Throughout, recognized deficits were improved in real time as the project progressed. The following section describes some of the major pros and cons of the individual products.

Image Processing Software

Pros

The primary advantage of the *fisherman* system is its ability to reproducibly locate the fish eyes. In addition, it includes a GUI that allows for user input to customize the processing to their particular needs. Another built-in feature that improves user-friendliness is the ability to choose whether an image is on a gray or RGB (red,green,blue) scale. This was a problem that had to be troubleshot offline using the gray scaling abilities of imaging programs such as Adobe Photoshop. A built-in Matlab option minimizes the amount of off-line photo editing to virtually nothing.

In addition, a preview button was inserted so that the user could view the processed image prior to sending the values to the stage translation software. This increased functionality and overall program flexibility.

Cons

The image processing software uses an iterative process to determine the locations of all of the fish. Due to the necessity of interaction with the process, it increases handling time and does not fully eliminate human error. The design cannot determine orientation as it currently stands, but will be a feature that would be of particular interest in the future. Other image processing techniques may be utilized to aid in this endeavor.

The speed of the image processing may be an issue to the client, although certain alternatives were reviewed but not pursued due to an observed decrease in efficiency.

More research is necessary to determine the most efficient methodology for this portion that minimizes processing time. Part of the time inefficiency can be attributed to poor programming style and could be improved with minimal changes.

Current System

Pros

The system being used currently is very inexpensive, as it was made from components that were already owned by the client. The client is already very familiar with its operation. The software has been thoroughly tested and is known to work. If need be, it is possible to make any changes that might be necessary to the coding or structure of the system.

Cons

It is impossible to perform a viable automated study using the system that is currently available to the client. With the image processing and the controlling software on separate machines, the fish can not be radiated and observed in real time. The camera being used to observe and take pictures of the fish is not high enough quality to show the amount of detail needed to witness any changes that the fish endure due to movement. In the current system, the source is positioned in such a way that radiation will go through the fish and into the lens of the camera, potentially causing damage to the lens.

§8. Future Work

Micro-Irradiator Holder

A prototype of the holder will be made to allow for testing. The fine adjusting mechanism has not been finalized. After that is completed, the system can be tested. Until that time, there is no way of knowing this is the correct way. The most important feature of the micro-irradiator holder is the adjustable track that allows the distance between the irradiator and the sample to be exact. A dial system similar to a caliper can be used to measure the exact distance. This will give the clients confidence that the distance and thus the measured dosage is in fact correct.

Software

The current imaging software locates the eyes of the fish and reports back the location. The orientation of each fish in the picture is not known. Overlapping fish have indistinct eye placement. Ultimately, the software will be completely automated. In the current state, the user must open the MATLAB program and run *Fisherman*. This program outputs a text file which then must be opened in Visual Basic 6.0. A proposed solution is an intermediary program that takes the data from one program to the other.

The image processing software uses an iterative process to determine the locations of all of the fish. Due to the necessity of interaction with the process, it increases handling time and does not fully eliminate human error. When fish locations are set, there are problems with the maximum and minimum values which can be selected. This will cause the program to send movement signals to the stage sending it to a location which it can not

actually move to. This causes the stage to “jump” when it reaches its boundary. A type of recognition needs to be added to the program so one can tell when points have been visited. The easiest way to do this is by drawing or placing a picture on the image at the specific location, so one knows the points which have been radiated. The design cannot determine orientation as it currently stands, but will be a feature that would be of particular interest in the future. Other image processing techniques may be utilized to aid in this endeavor.

The speed of the image processing may be an issue to the client, although certain alternatives were reviewed but not pursued due to an observed decrease in efficiency. More research is necessary to determine the most efficient methodology for this portion that minimizes processing time. Part of the time inefficiency can be attributed to poor programming style and could be improved with minimal changes.

References:

Campbell, N. and Reece, J., Biology: Sixth Edition, Benjamin Cummings, San Francisco, 2002.

Xu, X., "Xu Lab: Zebrafish Genetics Laboratory," Retrieved October 18th, 2005 from <http://mayoresearch.mayo.edu/mayo/research/Zebrafish>.

Appendix

Fishfinder code - Original M file for Image recognition

```
clc
clear variables
%%%%Loading picture and Displaying%%%%%%%%

% img=imread(filename); %image array named img
img=imread(input('What is the file name?','s')); %image array named
img
y=size(img);

img=rgb2gray(img); %NECESSARY FOR NON GRAY SCALE PICS

% subplot(1,2,1), imshow(img)
% subplot(1,2,2), imshow(img1) %plots on same figure

% figure, imhist(img) %histogram of pixel values

% img=edge(img, 'sobel'); %Edge filtering other is 'canny'
% figure, imshow(img) %Canny not good

% imgbodies=im2bw(img);
% imshow(imgbodies)

% K = rangefilt(img); %range filtering highlights the edges
% figure, imshow(K)

img=medfilt2(img, [3 3]); %Filtered image to use
% figure, imhist(img)

for k=1:length(img) %Thresholding
    for z=1:y(1,1)
        if img(z,k)>130
            img_a(z,k)=250;
        else
            img_a(z,k)=0;
        end
    end
end

% Neighboring
f = inline('sqrt(min(x(:)))');
% img_a2 = nlfilt2(img_a,[3 3],f);
img_a2 = colfilt(img_a,[6 6], 'sliding', @min);
figure, imshow(img_a2)
figure
```

```

for k=1:length(img) %Inversion
    for z=1:y(1,1)
        if img_a2(z,k)>75
            img_a3(z,k)=0;
        else
            img_a3(z,k)=250;
        end
    end
end

%Body count
level = graythresh(img_a3);
bw = im2bw(img_a3,level);

[labeled,numObjects] = bwlabel(bw,4);

imagedata=regionprops(labeled,'all');

%%%%%OPTIONS%%%%%
%Centroid
%Area
%FilledArea
%FilledImage
%Orientation

totalnum=numObjects;
count=1;

for k=1:length(imagedata)
    pixeltemp=imagedata(k).Area;
    if pixeltemp<1 | pixeltemp > 3000
        totalnum=totalnum-1;
    else
        center(count,1:2)=imagedata(k).Centroid;
        count=count+1;
    end
end
disp([num2str(totalnum),' is total # of bodies'])

%Fixing imagedata.centroid

l=1;

if totalnum~=0

% Loading centroid data into single column array
for k=1:length(center)
    a(l,1)=center(k,1);
    a(l+1,1)=center(k,2);
    l=l+2;
end

```

```

% open file

fid = fopen('array.txt','wt');% 'wt' means "write text"

if (fid < 0)

    error('could not open file "array.txt"');

end;

% write centroid info to file array

fprintf(fid,'%1.0f %1.0f\n',a); %%%Cuts off decimal...no rounding
                                %%%Appropriate format for VB

% close the file

fclose(fid);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % % % % figures chronologically%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% iptsetpref('ImshowBorder','tight'); %Filtered image
%
imshow(img)
hold on
scatter(center(:,1),center(:,2),'*', 'g');

end
% figure, imhist(img) %histogram

% figure, imshow(img_a) %Thresholded Image

% figure, imshow(img_a2) %Neighbored image

% figure, imshow(bw) %binary image

% figure, imshow(img_a3) %Inverted image

```

```

function varargout = fisherman(varargin)
% FISHERMAN M-file for fisherman.fig
%     FISHERMAN, by itself, creates a new FISHERMAN or raises the
existing
%     singleton*.
%
%     H = FISHERMAN returns the handle to a new FISHERMAN or the
handle to
%     the existing singleton*.
%
%     FISHERMAN('CALLBACK',hObject,eventData,handles,...) calls the
local
%     function named CALLBACK in FISHERMAN.M with the given input
arguments.
%
%     FISHERMAN('Property','Value',...) creates a new FISHERMAN or
raises the
%     existing singleton*. Starting from the left, property value
pairs are
%     applied to the GUI before fisherman_OpeningFunction gets called.
An
%     unrecognized property name or invalid value makes property
application
%     stop. All inputs are passed to fisherman_OpeningFcn via
varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help fisherman

% Last Modified by GUIDE v2.5 30-Nov-2005 22:51:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @fisherman_OpeningFcn, ...
                  'gui_OutputFcn',  @fisherman_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% pushbutton8 initialization code - DO NOT EDIT

```

```

% --- Executes just before fisherman is made visible.
function fisherman_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to fisherman (see VARARGIN)

% Choose default command line output for fisherman
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes fisherman wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = fisherman_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in process_button.
function process_button_Callback(hObject, eventdata, handles)

try
%%%%Loading picture and Displaying%%%%%%%%
file=get(handles.filename, 'String');
img=imread(file); %image array named img
y=size(img);
catch
    disp('Filename invalid!')
end

boolean=get(handles.RGBvalue, 'Value');
if boolean==1;
    img=rgb2gray(img); %NECESSARY FOR NON GRAY SCALE PICS
end

img1=medfilt2(img, [3 3]); %Filtered image to use

```

```

for k=1:length(img1) %Thresholding ---possibly add max value (rid of
dark noise
    for z=1:y(1,1)
        if img1(z,k)>str2num(get(handles.thresholdvalue,'string'))...
            | img1(z,k)<str2num(get(handles.thrlowbox,'string'))
            img_a(z,k)=250;
        else
            img_a(z,k)=0;
        end
    end
end

if str2num(get(handles.neighborvalue,'string'))>1
% Neighboring
nn=str2num(get(handles.neighborvalue,'string'));
f = inline('sqrt(min(x(:)))');
img_a2 = colfilt(img_a,[nn nn],'sliding',@min);
else
    img_a2=img_a;
end

h=size(img_a2);
for z=1:3
for k=1:h(1,2)

        img_a2(z,k)=250;
        img_a2(h(1,1)-z+1,k)=250;
    end
end

    for z=1:3
for k=1:h(1,1)

        img_a2(k,z)=250;
        img_a2(k,h(1,2)-z+1)=250;
    end
end

for k=1:length(img) %Inversion
    for z=1:y(1,1)
        if img_a2(z,k)>95
            img_a3(z,k)=0;
        else
            img_a3(z,k)=250;
        end
    end
end

%Body count
level = graythresh(img_a3);
bw = im2bw(img_a3,level);

[labeled,numObjects] = bwlabel(bw,4);

```

```

imagedata=regionprops(labeled, 'all');

%%%%%OPTIONS%%%%%
%Centroid
%Area
%FilledArea
%FilledImage
%Orientation

totalnum=numObjects;
count=1;
for k=1:length(imagedata)
    pixeltemp=imagedata(k).Area;
    if pixeltemp<str2num(get(handles.minpixel, 'string')) ...
        | pixeltemp > str2num(get(handles.maxpixel, 'string'))

        totalnum=totalnum-1;

    else
        center(count,1:2)=imagedata(k).Centroid;
        count=count+1;
    end
end
disp([num2str(totalnum), ' is total # of bodies'])

%Fixing imagedata.centroid

l=1;

>Loading centroid data into single column array
for k=1:length(center)
    a(l,1)=center(k,1);
    a(l+1,1)=center(k,2);
    l=l+2;
end

% open file

fid = fopen('array.txt','wt');% 'wt' means "write text"

if (fid < 0)

    error('could not open file "array.txt"');

end;

% write centroid info to file array

fprintf(fid, '%1.0f %1.0f\n', a); %%%Cuts off decimal...no rounding
                                %%%Appropriate format for VB

```

```

% close the file

fclose(fid);

if get(handles.orgimg, 'value')==1
figure('name', 'original image')
iptsetpref('ImshowBorder', 'tight'); %Filtered image
imshow(img);
hold on
scatter(center(:,1),center(:,2), '*', 'g');
hold off
end
if get(handles.pxlhst, 'value')==1
figure('name', 'Pixel Intensity Histogram'), imhist(img) %histogram,
end
if get(handles.thrimg, 'value')==1
figure('name', 'Thresholded Image') ,imshow(img_a) %Thresholded Image
end
if get(handles.ngbimg, 'value')==1
figure('name', 'Neighbored Image') ,imshow(img_a2) %Neighbored image
end
if get(handles.invimg, 'value')==1
figure('name', 'Inverse Image'), imshow(img_a3) %Inverted image
end

% hObject handle to process_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in preview_button.
function preview_button_Callback(hObject, eventdata, handles)

try
%%%%%Loading picture and Displaying%%%%%%%%
file=get(handles.filename, 'String');
img=imread(file); %image array named img
y=size(img);
catch
disp('Filename invalid!')
end

boolean=get(handles.RGBvalue, 'Value');
if boolean==1;
img=rgb2gray(img); %NECESSARY FOR NON GRAY SCALE PICS
end

img1=medfilt2(img, [3 3]); %Filtered image to use

```

```

for k=1:length(img1) %Thresholding ---possibly add max value (rid of
dark noise
    for z=1:y(1,1)
        if img1(z,k)>str2num(get(handles.thresholdvalue,'string'))...
            | img1(z,k)<str2num(get(handles.thrlowbox,'string'))
            img_a(z,k)=250;
        else
            img_a(z,k)=0;
        end
    end
end

if str2num(get(handles.neighborvalue,'string'))>1
% Neighboring
nn=str2num(get(handles.neighborvalue,'string'));
f = inline('sqrt(min(x(:)))');
img_a2 = colfilt(img_a,[nn nn],'sliding',@min);
else
    img_a2=img_a;
end

h=size(img_a2);
for z=1:3
for k=1:h(1,2)

        img_a2(z,k)=250;
        img_a2(h(1,1)-z+1,k)=250;
    end
end

    for z=1:3
for k=1:h(1,1)

        img_a2(k,z)=250;
        img_a2(k,h(1,2)-z+1)=250;
    end
end

for k=1:length(img) %Inversion
    for z=1:y(1,1)
        if img_a2(z,k)>95
            img_a3(z,k)=0;
        else
            img_a3(z,k)=250;
        end
    end
end

%Body count
level = graythresh(img_a3);
bw = im2bw(img_a3,level);

[labeled,numObjects] = bwlabel(bw,4);

```

```

imagedata=regionprops(labeled, 'all');

%%%%%OPTIONS%%%%%
%Centroid
%Area
%FilledArea
%FilledImage
%Orientation

totalnum=numObjects;
count=1;
for k=1:length(imagedata)
    pixeltemp=imagedata(k).Area;
    if pixeltemp<str2num(get(handles.minpixel, 'string')) ...
        | pixeltemp > str2num(get(handles.maxpixel, 'string'))

        totalnum=totalnum-1;

    else
        center(count, 1:2)=imagedata(k).Centroid;
        count=count+1;
    end
end
disp([num2str(totalnum), ' is total # of bodies'])

%Fixing imagedata.centroid

l=1;

>Loading centroid data into single column array
for k=1:length(center)
    a(l,1)=center(k,1);
    a(l+1,1)=center(k,2);
    l=l+2;
end

%%%%Figures Chronologic%%%%%%%%%
if get(handles.orgimg, 'value')==1
figure('name', 'Original Image')
iptsetpref('ImshowBorder', 'tight'); %Filtered image
imshow(img);
hold on
scatter(center(:,1),center(:,2), '*', 'g');
hold off
end
if get(handles.pxlhst, 'value')==1
figure('name', 'Pixel Intensity Histogram'), imhist(img) %histogram,
end
if get(handles.thrimg, 'value')==1
figure('name', 'Thresholded Image') ,imshow(img_a) %Thresholded Image

```

```

end
if get(handles.ngbimg, 'value')==1
figure('name','Neighbored Image') ,imshow(img_a2) %Neighbored image
end
if get(handles.inving, 'value')==1
figure('name','Inverse Image'), imshow(img_a3) %Inverted image
end

% hObject    handle to preview_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in vbrun.
function vbrun_Callback(hObject, eventdata, handles)
% hObject    handle to vbrun (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11
as a double

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text

```

```
%      str2double(get(hObject,'String')) returns contents of edit12
as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit13_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit13 as text
%      str2double(get(hObject,'String')) returns contents of edit13
as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit14_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit14 as text
%      str2double(get(hObject,'String')) returns contents of edit14
as a double
```

```

% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%         str2double(get(hObject,'String')) returns contents of edit15
as a double

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
close
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% --- Executes on button press in orgimage.
function orgimg_Callback(hObject, eventdata, handles)

% hObject    handle to orgimage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of orgimage

% --- Executes on button press in invimg.
function invimg_Callback(hObject, eventdata, handles)
% hObject    handle to invimg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of invimg

% --- Executes on button press in pxlhst.
function pxlhst_Callback(hObject, eventdata, handles)
% hObject    handle to pxlhst (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pxlhst

% --- Executes on button press in ngbimg.
function ngbimg_Callback(hObject, eventdata, handles)
% hObject    handle to ngbimg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of ngbimg

% --- Executes on button press in thrimg.
function thrimg_Callback(hObject, eventdata, handles)
% hObject    handle to thrimg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of thrimg

function thresholdvalue_Callback(hObject, eventdata, handles)
% hObject    handle to thresholdvalue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of thresholdvalue as
text
%         str2double(get(hObject,'String')) returns contents of
thresholdvalue as a double

% --- Executes on button press in RGBformat.

% --- Executes during object creation, after setting all properties.
function thresholdvalue_CreateFcn(hObject, eventdata, handles)
% hObject    handle to thresholdvalue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in RGBvalue.
function RGBvalue_Callback(hObject, eventdata, handles)
set(handles.gryscale,'Value', 0);

% hObject    handle to RGBvalue (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of RGBvalue

% --- Executes on button press in gryscale.
function gryscale_Callback(hObject, eventdata, handles)
set(handles.RGBvalue,'Value', 0)

% hObject    handle to gryscale (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of gryscale

% --- Executes on slider movement.
function thrlvl_Callback(hObject, eventdata, handles)

```

```

slider_value = get(hObject,'Value');
set(handles.thresholdvalue, 'string',num2str(slider_value))
% hObject    handle to thrlvl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% --- Executes during object creation, after setting all properties.
function thrlvl_CreateFcn(hObject, eventdata, handles)

% hObject    handle to thrlvl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function ngblvl_Callback(hObject, eventdata, handles)
slider_value = get(hObject,'Value');
set(handles.neighborvalue, 'string',num2str(slider_value))
% hObject    handle to ngblvl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% --- Executes during object creation, after setting all properties.
function ngblvl_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ngblvl (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

```

```

% -----
function File_Callback(hObject, eventdata, handles)
% hObject    handle to File (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function exit_Callback(hObject, eventdata, handles)
close
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in button_new.
function button_new_Callback(hObject, eventdata, handles)
% hObject    handle to button_new (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function thrlowbox_Callback(hObject, eventdata, handles)
% hObject    handle to thrlowbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of thrlowbox as text
%        str2double(get(hObject,'String')) returns contents of
thrlowbox as a double

% --- Executes during object creation, after setting all properties.
function thrlowbox_CreateFcn(hObject, eventdata, handles)
% hObject    handle to thrlowbox (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

```

```

set(hObject, 'BackgroundColor', get(0, 'defaultUicontrolBackgroundColor'))
;
end

% --- Executes on slider movement.
function throwslide_Callback(hObject, eventdata, handles)
slider_value = get(hObject, 'Value');
set(handles.throwbox, 'string', num2str(slider_value))
% hObject    handle to throwslide (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'Value') returns position of slider
%        get(hObject, 'Min') and get(hObject, 'Max') to determine range
of slider

% --- Executes during object creation, after setting all properties.
function throwslide_CreateFcn(hObject, eventdata, handles)
% hObject    handle to throwslide (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
    set(hObject, 'BackgroundColor', [.9 .9 .9]);
else

set(hObject, 'BackgroundColor', get(0, 'defaultUicontrolBackgroundColor'))
;
end

```