Studying cell migration behaviors in microtissues using a LabChip

Advisor: Dr. Long Que

Client: Dr. Long Que

The team: ***May1634***

Role	***Team Member***
Leader	Jonathan Yatckoske
Webmasters	Yaxiong Zhang
	Chun-Hao Lo
Communication Leader	Yuqian Hu
Key Concept Holder	Kaiyu Xu

Table of Contents:

Project Statement:

This project is to design a Matlab program to assist with cell behavior studies on our client's LabChip device. The program must track migration of one or multiple cells in arrayed microtissues formed by a polymer microfluidic chip developed in the lab (LabChip). The program must have a graphical user interface and track the cell migration using images from an optical microscope as input data.

Design Specification:

Each chamber of the microfluidic device contains one droplet with one or a few cells in microtissue. For this reason, we came up with four different ideas in order to make the cells stand out.

- Dye the cells, and track the movement of the color
- Collect the data with the Fluorescence Microscope
- Change the contrast of image data so we separate cell and background in different color
- Edge detection

Three out of the four ideas were not reasonable. This is covered in greater detail in Appendix II.

Matlab has several built-in image processing functions that provided a good starting point for our program. The primary detection method on the cells is edge detection. The basic edge detection function is followed by several functions for cleaning up the noise and isolating the cells from the rest of the source image. The exact methods and parameters for the process were set for the specific input data our client provided.

The edge detection is performed on each frame of an image stack to add the position data to vectors containing the x-direction trajectory and the y-direction trajectory. The trajectory vectors are converted from pixel position to µm position and plotted, with different colors representing each detected object.

The program also detects well-defined circles to determine how many full droplets are in the input data's source image, and processes through all frames for every detected droplet, producing trajectory plots for every non-empty droplet.

The program in general works reasonably well, but some accuracy was sacrificed in favor of the script running faster. The detection function we've written isn't reliable for every frame, so the program drops the frames that produce unreasonable data. The number of dropped frames are tracked to provide a quantifiable indication of error.

System Requirements:

Experiments on cell migration in microtissues are our client's goal. Using the optical microscope to get videos of the cell migration is expected.

A Matlab program is needed to identify and track the location of one or multiple cells in the droplets stored in the device chambers. The program must identify the number of cells in the chamber, collect location data and produce plots of the migration trajectory of the cells, as well as calculate velocity, distance traveled, and displacement of the cells.

Functional Decomposition:

1. Image the migration of cells in microtissues using optical microscope
2. Track these cells and take pictures of them every two minutes
3. Load photos into Matlab
4. Crop to the chambers and find locations of cells using our Matlab detection
5. Save the object location data
6. Comparing the current data point with last data point
7. Plot the migration trajectory of the cells using a Matlab program
8. Output the trajectory data and results of analysis calculations to the Matlab workspace and external files as necessary

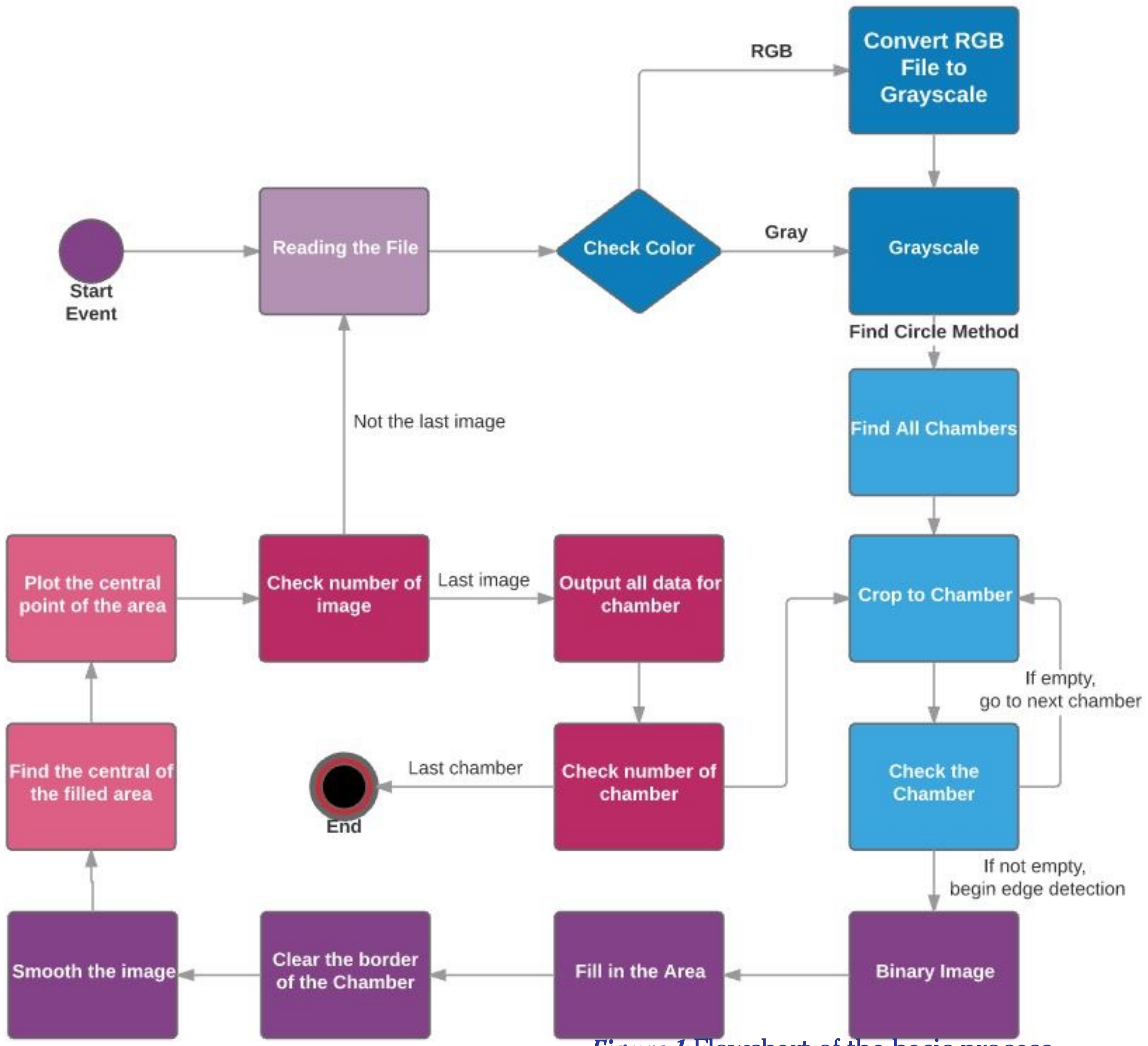


Figure 1: Flowchart of the basic process

Detailed Description:

Droplets & Chips:

The input of our collected data will be the cells within the droplets that will be collected in the storage chambers. Figure 2 shows the LabChip structure which forms and stores the droplets which are imaged for the input data to our program.

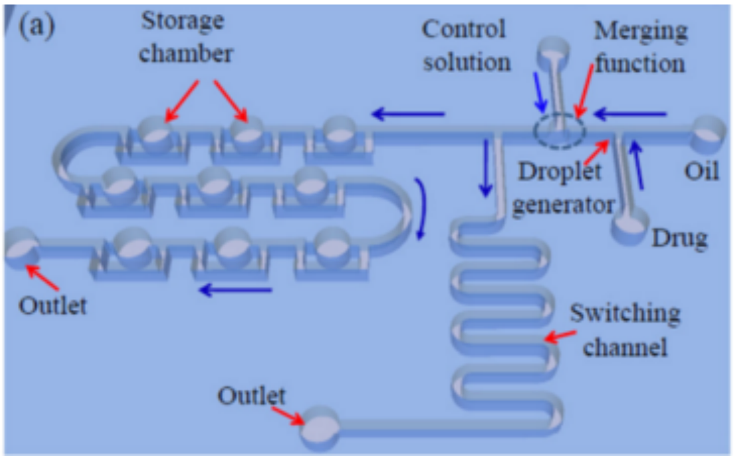


Figure 2

Figure 3 shows the dimension of the chambers. The Matlab program is designed specifically for use with input data of these chambers.

Dimensions of the chamber

D1	= 50 μm
D2	= 120 μm
D3	= 10 μm
Thickness	= 50 μm

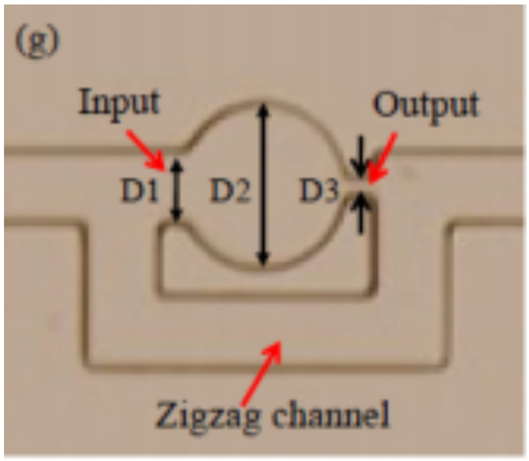


Figure 3

Matlab Program:

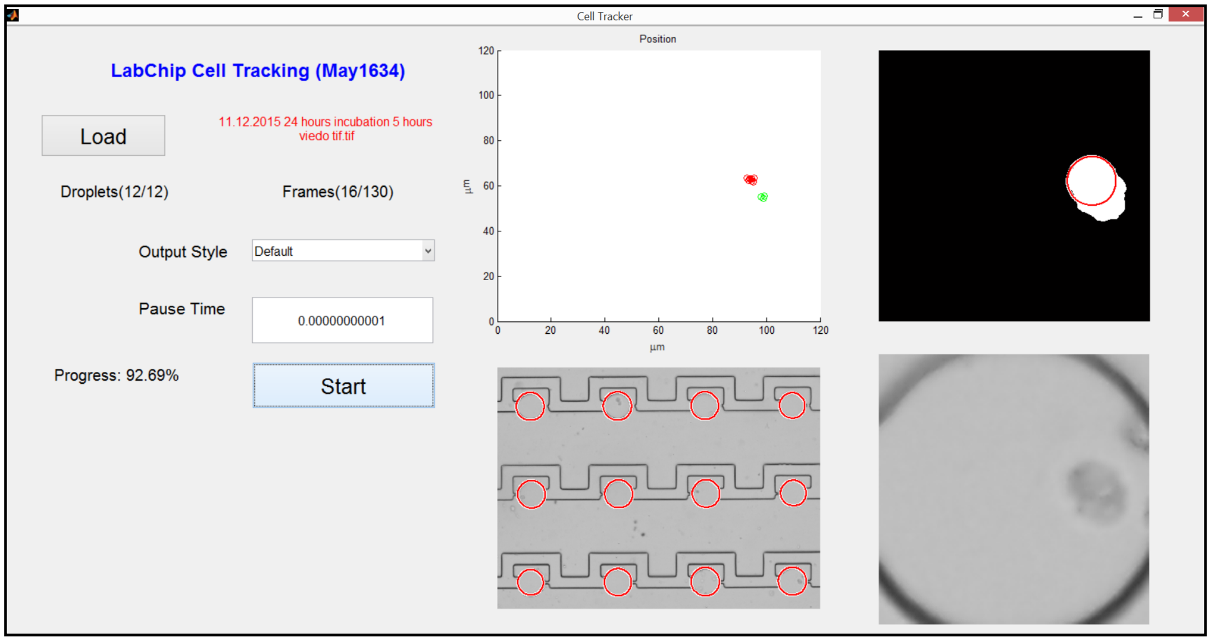
The input for the Matlab Program will be the output from the data collection, which are the tiff files of a selection of the chambers on the chip. As the images of the stack are read by the program, the individual droplets are detected, and the functions are called for detecting the cells inside the droplets. Each droplet detected is cropped to, a handful of the initial frames are processed only for the presence of cells at all, if the majority of these frames aren't empty the droplet is processed for every frame in the image stack. Empty droplets are skipped.

GUI displays the circles in the source, the current chamber in process, the binary image result of the detection function, and the position plot. Images and progress values are refreshed every frame. If a frame doesn't provide a reasonable change in position, or fails to detect an object at all, the frame is dropped. Number of dropped frames is tracked.

Program outputs the trajectory information to the Matlab workspace and files.

Testing:

The goal of our project is to analyze the behavior of the cells in the chambers and plot it into a graph so the result will be easier for people to study. To test our program, the behavior of the cells in the droplet will be the input data for our program. The program will turn the input data into a binary image and detect the edge of the cells. In addition, it will remove the edge of the chamber and smooth the binary image. The reason of smoothing the binary image is to decrease the noise and other effects of the images. Furthermore, the program will detect the center of the cells and plot it into the graph as the result.



LabChip Cell Tracking (May1634)

11.12.2015 24 hours incubation 5 hours viedo tif.tif

Droplets(12/12)

Frames(16/130)

Output Style: Default

Pause Time: 0.00000000001

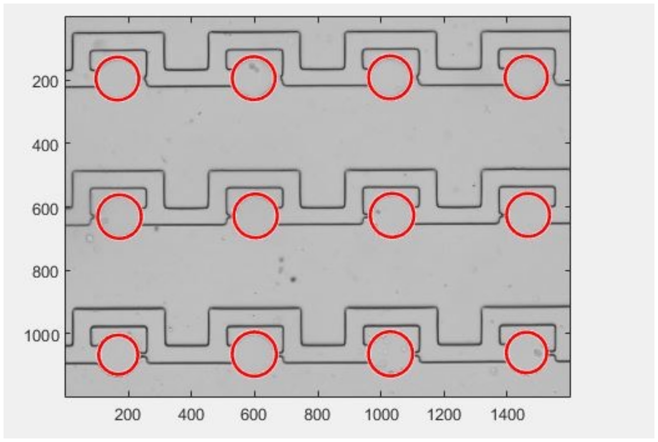
Progress: 92.69%

By comparing the original image to the binary image and the location of the center on the plot, we can visually identify if the program works reasonably well and what the errors are. Input data is one of the dominant causes of error for the program. The edge detection relies on noticeable contrast between the background of the image and the perimeter of the cells. Higher quality images for input data will result in less error in the Matlab program.

Issues & Challenges:

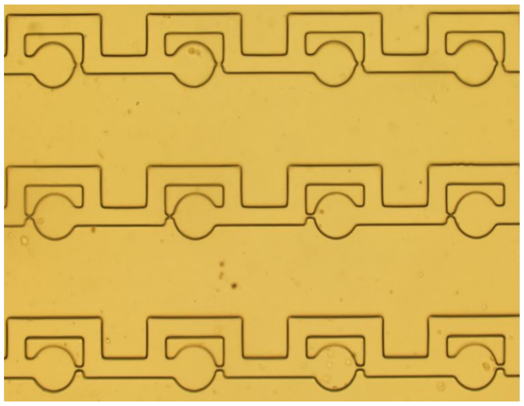
Sorting the data:

The Matlab function used to detect circles sorts the output data based on a metric that rates the quality of the circles. This sorting isn't useful for our purposes, so we had to include a function for re-sorting the data into an order based on position in the image. We opted to sort first by y-coordinates of the circles and then by x-coordinates within ranges of the y-position. This sorts the circles from the top of the image down and from left to right, with a set tolerance range on the y-position to establish rows.



Potentially different types of input data:

Many of the image processing functions our Matlab program relies on to detect the cells only work on grayscale images, but some of the input data our client provided have an RGB colormap. Our program restricts the input data to uncompressed tiff files, but automatically converts RGB tiff images into grayscale tiff by checking for bit depth.



Overlapping:

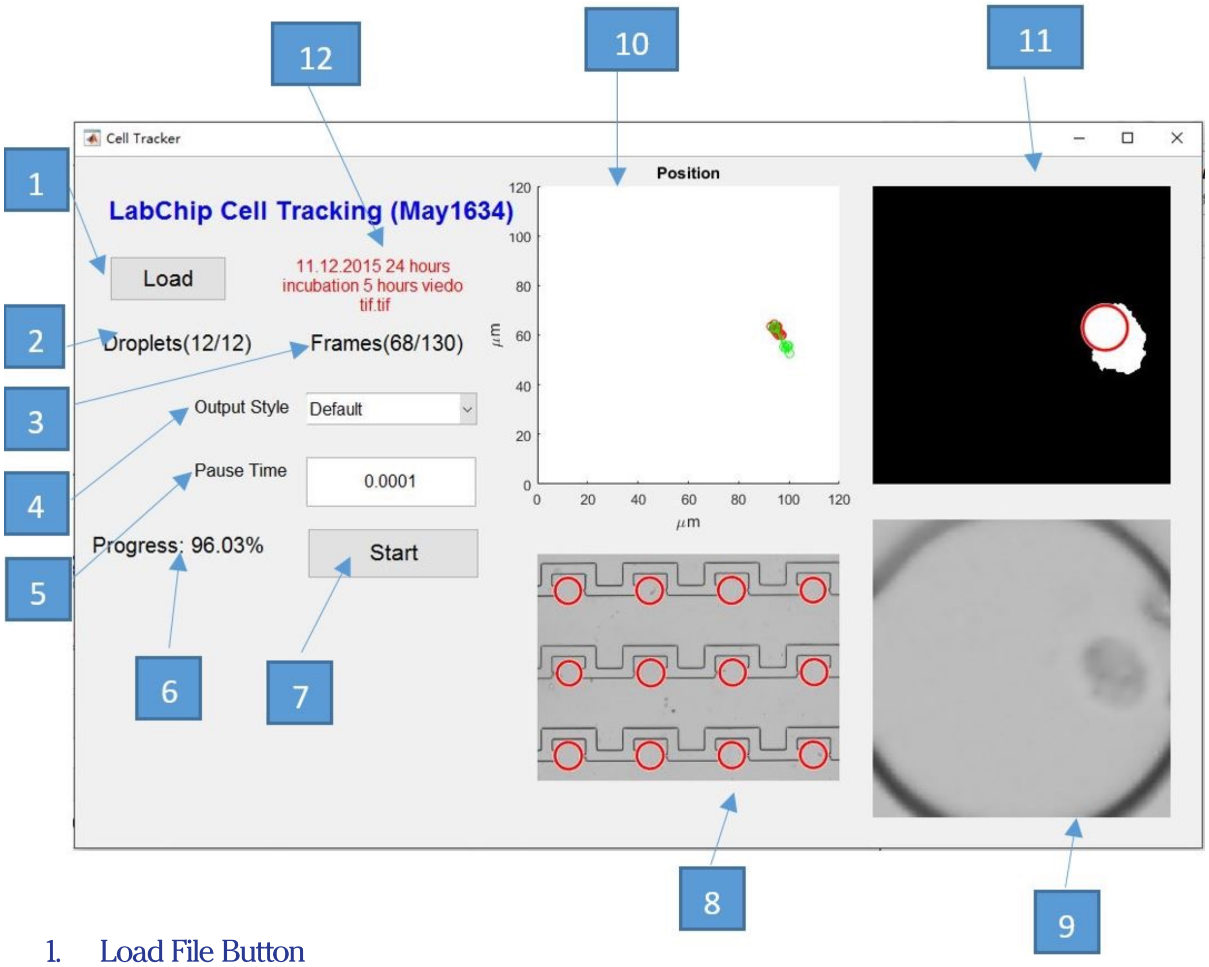
Possibly the biggest issue for our program was the cells being close enough to be detected as overlapping, which causes the edge detection process to detect them as one object. In an attempt to overcome this, we process the binary image through the same function for finding circles that we utilize to find the droplets. It still has a significant amount of error, but because individual cells tend to be reasonably circular, finding multiple circles inside a large object is more accurate than using the centroid of the large object.

Conclusion:

By monitoring the movement of cells in microtissues in the microfluidic device, we get some basic understanding of the cells' responses to the extracellular matrix and how cells communicate. Our Matlab program is meant to be used to automatically provide trajectory information to researchers who want to study the movement of cells using the LabChip device. The program provides the useful variables into the Matlab workspace for immediate continued work, as well as providing the data as external files so the user can have the data available later without having to run the tracking program again.

Additional Documents:

Appendix I: Operation Manual



1. Load File Button
2. Number of Droplets
3. Frames
4. Output Style
5. Pause Time
6. Progress percentage
7. Start button
8. Droplets picture
9. Individual droplet
10. Result plot
11. Find the cell area
12. File name

The user collects input data by taking photos of the LabChip device viewed through an optical microscopes. The photos are typically taken every 1 or 2 minutes over several hours. These photos are stored in stacks of images saved as uncompressed tiff stacks.

Running the Matlab script produces the initial GUI. The user selects the file of the input data by clicking the “Load” button. The filename is displayed in red next to this button.

The output style dropdown menu allows the user to select the type of plot. The default plot shows the position relative to the size of the droplet. The origin-centered plot shows the trajectories of all the objects detected in the droplet with the initial position of every trajectory at the center of the plot.

Once the file is loaded and the output style is selected, the user enters the pause time. The pause time is the minimum delay between the frames during the processing. This feature is included because the program may be used on different systems, some of which can process the images fast enough that the progress of the program can’t be visually followed in the figures the GUI displays. Forcing the delay allows the user to visually observe the tracking process no matter what system the program is run on.

Once the pause time is set, the tracking process begins by clicking the “Start” button. The program finds the droplets by looking for well-defined circles. The droplet locations are sorted based on position in the image. The lower left image displays the source image in grayscale with the detected circles displayed in red.

The image is cropped to the current droplet being checked, and resized up to provide more pixels to work with in the cell detection. The current droplet is displayed in the lower right image in the GUI. The cell detection is performed on this images and the final result of this is displayed in the upper right image.

This is done for a small number of frames, and if the majority of these frames are empty, the droplet is skipped. If the droplet isn’t skipped, then all the frames in the stack are processed and the detected cell trajectories are plotted in the plot on the upper left in the GUI.

During the testing phase of the process, the progress of the testing is displayed with the current frame checked out of the number of frames being checked.
When the droplet isn't empty, the location of the testing progress instead displays the progress of the frames by showing the current frame out of the total number of frames. Next to this progress, the number of the current droplet being checked is displayed out of the total number of droplets in the input data.

Next to the start button, the total progress is displayed as a percentage. The progress markers, the images, and the plot are all updated every time the frame is done being processed.

Once the droplet has been completely processed, the relevant data is sent to the Matlab workspace and saved as external files. Then the program moves to the next droplet. As stated, empty droplets are skipped.

Appendix II: Previous Versions of the Design

Edge detection proved to be the best cell detection method we could come up with. The first attempts at the code were made under the assumption that the input data would be from dyed cells under a fluorescence microscope. This would have allowed the code to look for specific color. The fluorescence microscope would be too expensive to use enough to get a lot of useful input data from, and dyeing the cells wouldn't improve the contrast a significant amount without the use of the fluorescence microscope.

After realizing the fluorescence microscope wasn't a viable option, we didn't want to abandon the method of looking for color immediately. We looked into the possibility of manually coloring the cells in the input data before running a script to track the cell position. Manually modifying over a hundred frames of an image would be very time consuming and would have defeated the purpose of automatically processing the images in order to obtain trajectory data. We decided that any image processing done would have had to be part of the Matlab program.

The original edge detection function used the centroid of the objects in the resulting binary image as the position data. This method didn't handle overlap well at all, but any method we looked into for dealing with overlap in a very accurate way would have required a lot of image processing that would have slowed down the program too much. For this reason, we opted for a process that drops unreasonable changes in position and frames that don't provide data, and the number of dropped frames are tracked to assist in quantifying error.

Appendix III: Other Considerations & Final Thoughts

We wasted a lot of time dealing with the original plan for 3D tracking. We kept trying to think of possible ways we could get 3D trajectory data, but we were unable to get any kind of 3D input data and the 2D input data didn't allow for any useful method for tracking 3D position.

Admittedly, since cells move in three dimensions, being able to track that movement in all those dimensions would be highly valuable. We spent a large amount of time trying to figure out options for tracking 3-dimensional migration trajectories. We couldn't assume the area of the cell in the 2D image was fixed, so tracking change in that area isn't an option for tracking z-position. We thought we might be able to assume fixed focal length and use that assumption to track z-position, but due to environmental factors like temperature, the focal length on the input data was never reliably constant. When these strategies were abandoned, we still tried to keep working on 3D trajectories. We wasted a lot of time focusing on finding ways of getting 3D input data, but none of those ideas were able to work out in time for us to write code for it.

In the end, we wasted a lot of time trying to meet our client's original desire for 3-dimensional migration trajectory analysis, and it caused our final program to be less than what we had hoped. If we had focused on 2-dimensions exclusively from the start of the project, our final program could have been much better. We simply wasted time trying to find a way to make a more useful project a reality.

After finally giving up on the 3-dimensional Matlab tracking, we were able to develop a decent program and GUI for tracking cell migration trajectory in 2-dimensions. The program is decently functional, but in need of major improvements before it can be realistically utilized as a research tool.

Appendix IV: Matlab Code

```
%{
Iowa State University - Senior Design Team May1634
Most recent update: April 27th, 2016

Project Adviser/Client: Dr. Long Que

Project Team:
    Jonathan Yatckoske
    Yaxiong Zhang
    Yuqian Hu
    Chun-Hao Lo
    Kaiyu Xu

Notes:
    Some functions of this script are incomplete.  Origin-centered
    plotting, for instance.

    Other functions may not work as well as desired for all input data.
%}



function varargout = CellTrackerGUI(varargin)
% CELLTRACKERGUI MATLAB code for CellTrackerGUI.fig
%      CELLTRACKERGUI, by itself, creates a new CELLTRACKERGUI or raises the
existing
%      singleton*.
%
%      H = CELLTRACKERGUI returns the handle to a new CELLTRACKERGUI or the
handle to
%      the existing singleton*.
%
%      CELLTRACKERGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CELLTRACKERGUI.M with the given input arguments.
%
%      CELLTRACKERGUI('Property','Value',...) creates a new CELLTRACKERGUI or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before CellTrackerGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to CellTrackerGUI_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

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

% Last Modified by GUIDE v2.5 17-Apr-2016 17:58:16

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CellTrackerGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @CellTrackerGUI_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
% End initialization code - DO NOT EDIT


% --- Executes just before CellTrackerGUI is made visible.
function CellTrackerGUI_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 CellTrackerGUI (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = CellTrackerGUI_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 loadButton.
function loadButton_Callback(hObject, eventdata, handles)
% hObject    handle to loadButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global info;
global filename;
global num_images;
global test;
global data;
global radius;

%loads file and stores relevant filename and info to the workspace
filename = uigetfile('*.tif');
assignin('base','filename',filename);
info = imfinfo(filename);
assignin('base','info',info);
num_images = numel(info);
assignin('base','num_images',num_images);

%displays loaded filename in the GUI
set(handles.StaticText,'string',filename);

test = 0;
data={};

radius = 57;



function StartPage_Callback(hObject, eventdata, handles)
% hObject    handle to StartPage (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 StartPage as text
%        str2double(get(hObject,'String')) returns contents of StartPage as a double
% --- Executes during object creation, after setting all properties.
function StartPage_CreateFcn(hObject, eventdata, handles)
% hObject    handle to StartPage (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 EndPage_Callback(hObject, eventdata, handles)
% hObject    handle to EndPage (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 EndPage as text
%        str2double(get(hObject,'String')) returns contents of EndPage as a double


% --- Executes during object creation, after setting all properties.
function EndPage_CreateFcn(hObject, eventdata, ~)
% hObject    handle to EndPage (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 selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox2


% --- Executes during object creation, after setting all properties.
```

```
function listbox2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox 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 key press with focus on listbox2 and none of its controls.
function listbox2_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.UICONTROL)
%       Key: name of the key that was pressed, in lower case
%       Character: character interpretation of the key(s) that was pressed
%       Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)


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


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global type;
contents=get(handles.popupmenu1,'value');
switch contents
    case 1
        type=0;
    case 2
        type=1;
end



% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu 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 pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global info;
global type;
global filename;
global num_images;

global data;
global radius;
global pausetime;
bit_depth = info.BitDepth;

colorsForTrajPlot = {'r','g','b','y','m','c'};

X = imread(filename, 1);
[centers, radii] = findDroplets(X,50,100);
[numDroplets, trash] = size(centers);

label = sprintf('Droplets(1/%d)', numDroplets);
set(handles.chamberprog, 'String', label);

percentageChange = 1/(numDroplets*num_images);
percentage = 0;


for i = 1:numDroplets
    cla(handles.axes1);
    empty = 0;
    TotCells = 0;

    x_traj = cell(150,1); xtrajreal = cell(150,1);
    y_traj = cell(150,1); ytrajreal = cell(150,1);
    lastx = cell(10,1);
    lasty = cell(10,1);
    droppedframes = cell(10,1);

    %Check if the droplet has any cells
    for k = 1:5
        X = imread(filename, k);

        if bit_depth==24
            X = rgb2gray(X);
        end

        [centers, radii] = findDroplets(X,50,100);
        [X2{k}, BW_final, stats, centers_loc, rads] = findCells( X, centers, radii, radius, k, i);
        if isempty(centers_loc)
            empty = empty + 1;
        end
        imshow(X,'Parent',handles.axes2), viscircles(handles.axes2, centers, radii);
        testlabel = sprintf('Testing(%d/5)', k);
        set(handles.frameprog, 'String', testlabel);
        label = sprintf('Droplet (%d/%d)', i, numDroplets);
        set(handles.chamberprog, 'String', label);
        imshow(BW_final,'Parent',handles.axes3), viscircles(handles.axes3, centers_loc, rads);
        imshow(X2{k},'Parent',handles.axes4);
        label = sprintf('Progress: %.2f%%', percentage*100);
        set(handles.percentage, 'String', label);
        pause(pausetime);
    end



    %if the majority of the first 5 frames aren't empty, process all frames
    %for trajectory plots and data
    if (empty < 3)
        for k = 1:num_images
            X = imread(filename, k);

            if bit_depth == 24
                X = rgb2gray(X);
            end

            [centers, radii] = findDroplets(X,50,100);
            [numCenters trash] = size(centers);

            if (numCenters==numDroplets)
                [X2{k}, BW_final, stats, centers_loc, rads] = findCells( X, centers, radii, radius, k,
i);
```

```
    data{k} = centers_loc;
else
    data{k} = [];
    if (k>1)
        X2{k} = X2{k-1};
    else
        X2{k} = [];
    end
end

[numCells, trash] = size(data{k});
if (numCells>TotCells)
    TotCells = numCells;
end


if not(isempty(data{k}))
    for j=1:numCells
        testlast = max(find(~cellfun('isempty',lastx)));
        assignin('base','testlast',testlast);

        if (k>1 && j<=testlast)
            if (abs(data{k}(j,1)-lastx{j}) < 8) && (abs(data{k}(j,2)-lasty{j}) < 8)
                    x_traj{j} = [x_traj{j} data{k}(j,1)];
                    y_traj{j} = [y_traj{j} data{k}(j,2)];
            elseif (j==testlast)
                if (isempty(x_traj{j}))
                    x_traj{j} = [x_traj{j} data{k}(j,1)];
                    y_traj{j} = [y_traj{j} data{k}(j,2)];
                end
            else
                droppedframes{j} = droppedframes{j} + 1;
            end
        else
            droppedframes{j} = droppedframes{j} + 1;
        end

        lastx{j} = data{k}(j,1); lasty{j} = data{k}(j,2);
        assignin('base','lastx',lastx); assignin('base','lasty',lasty);
    end
else
    for jj=1:10
        droppedframes{j} = droppedframes{j} + 1;
    end
end

if numel(x_traj)>=1
    if type==0
```

```
            %default plot
            %plots the position of the cell trajectories within the
            %boundaries of the droplet
            %droplet is 120 micro-meters in diameter, the
            %trajectories are stored in pixel positions, so the
            %plot is
            %scaled to the actual units
            for j=1:TotCells
                plot(handles.axes1,x_traj{j}.*(120/334),120.-(y_traj{j}.*(120/334)),'o-', 'Color',
colorsForTrajPlot{j});
                %set plot axes and labels
                axis(handles.axes1, [0 120 0 120]);
                title(handles.axes1, 'Position')
                xlabel(handles.axes1,'\mum')
                ylabel(handles.axes1,'\mum')
                hold(handles.axes1, 'on')
            end
        end

        if type==1
            %origin-centered plots
            %incomplete:  adjusts every value of the trajectory
            %vectors by subtracting the first value from all
            %values, making every cell trajectory start at 0
            %Does not plot these new trajectories on a suitable
            %axis yet

            %used to show the trajectories of the cell vs the other
            %cell trajectories in the droplet
            for j=1:TotCells
                if not(isempty(x_traj{j}))
                    origin_xtraj{j} = x_traj{j}; origin_ytraj{j} = y_traj{j};

                    origin_xtraj{j} = origin_xtraj{j}-x_traj{j}(1); origin_ytraj{j} = origin_ytraj{j}-
y_traj{1}(1);
                                        assignin('base','test',origin_xtraj);
assignin('base','test2',origin_ytraj);

                    plot(handles.axes1,-334.+origin_xtraj{j}.*(334/120),120.-
(origin_ytraj{j}.*(334/120)),'o-','color',colorsForTrajPlot{j});
                end
            end
        end
    end
```

```
            %update all figures and images in the GUI
            imshow(X,'Parent',handles.axes2), viscircles(handles.axes2, centers, radii);
            label = sprintf('Frames(%d/%d)', k, num_images);
            set(handles.frameprog, 'String', label);
            percentage = percentage+percentageChange;
            label = sprintf('Progress: %.2f%%', percentage*100);
            set(handles.percentage,'String',label);
            imshow(BW_final,'Parent',handles.axes3), viscircles(handles.axes3, centers_loc,
rads);
            imshow(X2{k},'Parent',handles.axes4);
            label = sprintf('Droplets(%d/%d)', i, numDroplets);
            set(handles.chamberprog,'String',label);
            pause(pausetime);

            %send relevant data to Matlab workspace
            assignin('base',sprintf('data%d', i),data);
            assignin('base',sprintf('xtraj%d', i),x_traj);
            assignin('base',sprintf('ytraj%d', i),y_traj);
            assignin('base',sprintf('framedrops%d', i), droppedframes);
            %assignin('base','X2', X2);
        end
    else
        percentage = percentage + percentageChange*num_images;
    end

    %export relevant data to csv files
    %each file will begin with the loaded filename, followed by the
    %variable exported with a number for the droplet (i), followed by the
    %number of the cell (j)

    %this syntax can potentially export empty files, and it might be
    %preferable to export the variables into as few csv files as possible
    %instead
    for j=1:TotCells
        dlmwrite(sprintf('%s - xtraj%d - %d.csv', filename, i, j), x_traj{j});
        dlmwrite(sprintf('%s - ytraj%d - %d.csv', filename, i, j), y_traj{j});
        dlmwrite(sprintf('%s - framedrops%d - %d.csv', filename, i, j), droppedframes{j});
    end
end

set(handles.percentage, 'String',sprintf('Progress: %.2f%%', 100));



function PauseTime_Callback(hObject, eventdata, handles)
% hObject    handle to PauseTime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global pausetime;
n=get(hObject,'string');
pausetime=str2double(n);


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


% --- Executes during object creation, after setting all properties.
function PauseTime_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PauseTime (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 [ X2, BW_final, stats, centers_loc, rads ] = findCells(X, centers, radii, radius, k, i)
%findCells using edge detection and image processing to locate the cells within the frame of
the droplets
%   final version of the function must iterate through the droplets
%   identified by centers array

        %crops and enlarges droplets from the source image
        rect = [centers(i,1)-radius centers(i,2)-radius 2*radius 2*radius];
        X2 = imresize(imcrop(X, rect),2.9,'bilinear');

        %basic edge detection with chosen parameter values
        [~, threshold] = edge(X2, 'canny');
        fudgeFactor = 0.8;
        BWs = edge(X2,'canny',threshold*fudgeFactor);

        %image dilation to make the detected edges more noticeable
        se90 = strel('line',3,90);
        se0 = strel('line',3,0);

        BWsdil = imdilate(BWs, [se90,se0]);

        %fill holes in the image to form whole objects
        BWdfill = imfill(BWsdil, 8, 'holes');

        %remove objects in contact with the edge of the image, assumed to
```

```
%be the droplet edge
BWnobord = imclearborder(BWdfill, 4);

%smooth the image to assist with further image processing
seD = strel('diamond',1);
BWsmooth = imerode(BWnobord,seD);
BWsmooth = imerode(BWsmooth,seD);

%clear up initial noise/ any object with an area smaller than 300
%pixels is removed from the image
BW_final = bwareaopen(BWsmooth, 300);

%basic watershed segmentation
D = -bwdist(~BW_final);
Ld = watershed(D);

bwf = BW_final;
bwf(Ld == 0) = 0;

mask = imextendedmin(D,2);

D2 = imimposemin(D,mask);
Ld2 = watershed(D2);
bwf3 = BW_final;
bwf3(Ld2 == 0) = 0;

%clear up noise again after watershed segmentation
BW_final = bwareaopen(bwf3, 300);

%find circles from the objects found in the image at this point
[c_local, rads] = imfindcircles(BW_final, [20 40], 'Method', 'TwoStage');
overlapflag = 0;

%remove overlapping circles
for ii = 1:(size(c_local,1)-1)
    if(abs(c_local(ii,1)-c_local(ii+1,1) < rads(ii)+rads(ii+1)))
        c_new = c_local([1:ii,ii+2:end],:);
        r_new = rads([1:ii,ii+2:end]);
        overlapflag = 1;
    end
end

if (overlapflag == 1)
    c_local = c_new; rads = r_new;
end

%circles are assumed to be the cells
%this should provide basic approximate location of the cells in the
```

```
        %image
        centers_loc = c_local;

        %this returns the centroids of the detected objects
        stats = regionprops(BW_final, 'Centroid');


 function [ centers, radii ] = findDroplets( image, min_radius, max_radius )
%findDroplets finds chambers with complete droplets on the LabChip device
%   Uses the imfindcircles function to find the droplets within a radius
%   range.  Because imfindcircles sorts output by a metric that is useless
%   for our purposes, this function then resorts the circles found by
%   position in the image.
[centers_local, radii] = imfindcircles(image, [min_radius max_radius], 'Method', 'TwoStage');


if not(isempty(centers_local))
    %sort by y
    [y_co,y_index] = sort(centers_local(:,2));

    temp_i = sort(y_index);

    temp = centers_local;

    temp(temp_i) = centers_local(y_index); %sorts x-coordinate by ascending order of y-
coordinates
    temp(temp_i,2) = centers_local(y_index,2);

    %then sort by x
    if (length(temp)>2)
        for j = 1:length(temp)-1
            for i=1:length(temp)-j
                if (temp(i+1,2)-temp(i,2) < 100)
                    if (temp(i+1,1) < temp(i,1))
                        holdx = temp(i,1); holdy = temp(i,2);
                        temp(i,1) = temp(i+1,1); temp(i,2) = temp(i+1,2);
                        temp(i+1,1) = holdx; temp(i+1,2) = holdy;
                    end
                end
            end
        end
    end

    centers = temp;
else
    centers = [];
end
```