More than a million people in the United States suffer from congenital heart defects. The causes and formation of these defects are poorly understood, and this limits the range of potential treatment options. Congenital defects are also often not significant until later in life, and a better understanding of their origins could lead to earlier and more effective treatments. In order to better understand congenital defects, we are studying the developing heart in an embryonic quail model. Similar to humans, the embryonic quail heart develops in a tubular shape, forms two chambers and separates into four chambers. Previously, we have used Optical Coherence Tomography (OCT) and histology to study the structure, genetics and hemodynamics of the developing heart. Here, we expand our studies into characterizing the propagation of electrical signals in the heart using Optical Mapping (OM), a technique that visualizes changes in membrane potential using a membrane-bound, voltage-sensitive fluorescent dye, di-4-ANEPPS. As illustrated in Figure 3, the emission spectra of di-4-ANEPPS shifts in Figure 1.

Optical Mapping is a powerful technique for determining patterns of conduction in the heart using a voltage-sensitive fluorescent dye to visualize changes in membrane potential. Optical mapping signals are inherently noisy due to the short temporal sampling times and the need to measure small changes in fluorescent intensity. This challenge is exacerbated in the embryonic heart, which is only several cell layers thick. To increase SNR, the signals are normalized and both temporally and spatially filtered. The processed data allows extraction of various measurements, including activation times and action potential durations. Using these measurements, action potential propagation across the heart can be represented in various different manners including isochronal activation maps, and propagation velocity vectors fields. These representations allow efficient and effective analysis of optical mapping data.

The shape of a typical cardiac action potential is shown and explained in Figure 2. While action potentials vary significantly, it is the general characteristics of these action potentials across the heart that OM allows us to study.

In order to translate transmembrane voltages into optical signals, we use a membrane-bound, voltage-sensitive fluorescent dye, di-4-ANEPPS. As illustrated in Figure 3, the emission spectra of di-4-ANEPPS shifts in

![Diagram](image.png)

**Figure 1.** Simplified Diagram of the interconnection of the characteristics of the developing heart and tools that can be used to study each. Optical Mapping is used to study the electrophysiology of the developing heart, bridges structure to hemodynamics.
An Optical Mapping system consists of a light source, optics to focus and collect light from the heart sample, filters to narrow the light bandwidth received and an Electron Multiplying-CCD (EMCCD) to record fluorescent images of the heart in high temporal and spatial resolution. An image of our optical mapping system is shown in Figure 4.

In this report, we describe a data processing scheme for Optical Mapping data. The principle challenge of this work is that in order to generate ever higher resolution representa-

---

**Figure 2.** The general shape of a cardiac action potential. Activation begins with a fast sodium depolarization of the cell membrane (Phase 0), followed by inactivation of these sodium channels in Phase 1. Calcium moves inward in Phase 2, bringing the action potential to a plateau. Finally, the cell is repolarized by rectifier potassium channels and returns to the resting potential in Phase 4.

---

**Figure 3.** Absorption and Emission Spectra of di-4-ANEPPS. The rough placement of optical filters is also shown. Changes in transmembrane potential cause the emission spectra of di-4-ANEPPS to translate in wavelength, leading to a change in measured fluorescence. Modified from.

---

**Figure 4.** A picture of an optical mapping system with labels. Filters are not shown, but are found in the optics just above the sample stage. All data from the EMCCD camera is recorded by an attached computer. The sample stage is elevated to allow simultaneous OM and OCT recording.
tions of the heart, there is a corresponding decrease the area that each pixel covers, which decreases the SNR of the signals recorded at each pixel and requires various types of processing to increase the apparent SNR.

Methods
All data was recorded using an Andor iXon 860 EMCCD Camera and captured as a series of TIFF images. All data processing was done in Matlab R2011b using processing code shown in the Appendix. All of the hearts used in the examples are at an early stage of development (between stages 12 and 15) and are roughly tubular in shape.

Data Collection
Spatial recording resolutions for the data collected ranged between 42x42 pixels and 128x128 pixels, and temporal resolutions were 500Hz for all data sets. The collected optical signals varied in quality, but tended to generally have an SNR=3 or less. To precisely and reliably fit the data and extract information from it, we sought to increase to an SNR=10 or greater. Additionally, the time scales for these experiments are typically on the order of seconds, and so photobleaching was be accounted for through data processing as well.

Preprocessing
The data preprocessing is done in four steps, which are illustrated in the flowchart in Figure 5 and a processing example shwon in Figure 6. The first step is normalizing the individual pixel curves to zero in the time domain. This task was accomplished by averaging a flat region of the curve prior the first action potentials and subtracting this average from the entire curve. Second, a temporal filter was applied to remove high-frequency noise. Typically, data was taken at a sampling rate of 500Hz and filtered with

![Figure 5. Data processing steps are shown in a flow chart grouped by color. Orange steps are preprocessing, green steps are related to action potential detection, and purple, red and blue steps relate to the generation of activation maps, duration maps and conduction propagation vector maps respectively.](image-url)
a low-pass Butterworth filter at 200Hz. This
cutoff was determined empirically based upon
conservative analysis of the frequency spectra
of numerous datasets. The third step was to
correct for photobleaching (the photochemical
destruction of fluorophores), which leads to a
roughly linear decrease in fluorescence across
the time course of each data recording event.
To correct for this effect, we performed an af-
fine fit of each pixel curve and subtracted this

Figure 6. (A-D) provide representative examples of data through each of the pre-processing steps.
The plots show: (A) Original Data, (B) Normalized Data, (C) Temporally Filtered Data, (D) Photo-
bleaching Corrected Data, and (E) Spatially Filtered Data.
from each respective curve. These three steps serve to increase SNR and the uniformity of each pixel curve compared to its neighbors in preparation for spatial averaging, in the fourth and final preprocessing step. Spatial averaging was done using a disc-averaging filter with a radius of three pixels. While the filter choice was empirical, a disc shape was ultimately chosen over square filters because of the typically curved shape of the heart, and an averaging filter was chosen over a Gaussian filter because it achieved an increase in SNR without effectively applying a low-pass filter to the data (as a Gaussian would have).

Figure 7. (A-D) Four examples of the effectiveness of the fitting method. Red lines intersect on the 50%-max point on the rising edge of the action potential (Activation Point). Blue lines intersect on the 20%-max point on the falling edge. The difference between these to be the action potential duration. All curves are from the same data set.
**Action Potential Detection**

After preprocessing, a maxima is found on the user-defined interval containing the action potential. We then use the maximum point to split the user-defined interval into rising and falling edge regions. In the rising edge region, a polynomial was fitted to the curve to identify the baseline and maximum values, as well as to determine the 20% and 50%-maximum values. A polynomial fit was chosen because the variation between different action potential shapes and was deemed to be adequate by manual validation of the fit for several data sets. The activation point corresponding to the 50% value was found by subtracting the 50% value from the curve, taking the absolute value and finding the curve minimum in that region. This method is acceptable in this situation because our curve is effectively monotonic over the interval analyzed, but would not be robust without the monotonic condition. Examples showing the results of the fitting and detection algorithm are shown in Figure 7. The 50% time point is used as the action potential activation time. The same method is used on the falling edge interval to identify the time point of the 20%-max on the falling side of the action potential. The difference between the times points on the rising and falling edges gives what we arbitrarily defined as the 80% action potential duration.

**Activation and Duration Maps**

Using the activation and duration information gleaned in the previous section, we can develop different representations of the datasets. We chose to represent both the action potential activation times and durations as contour maps, in an attempt to clearly display of all the information while still representing it in two dimensions for ease of interpretation. Examples of these representations are shown in Figure 8.

**Conduction Propagation Vector Maps**

The method used to develop the conduction propagation vector fields closely follows that of Bayly et al. A wave front was fit around each nexus pixel using a minimum of six pixels in the neighborhood surrounding the nexus point with an ideal of around twenty pixels if possible. The partial derivatives of the fit surface at the nexus point were then used to calculated the propagation velocity.
in the x and y directions from this point. A vector map was plotted at each desired pixel using this information with the Matlab quiver function, which automatically scaled the vector field based upon the largest vector size. Unfortunately this representation fails when a handful of vectors have a much larger magnitude than the mean length. As such, we represent the magnitude of the vectors on a logarithmic scale is the example plots in Figures 9-11.

Results
Figures 8-10 show the outputs of processing three different data sets where the heart was untreated or treated with saline or alcohol to alter heart rate. In general, comparison between randomly selected points in the raw data and the processed representations suggests that these figures accurately represent the raw data collected. The representations also reasonably matched expectations based upon known electrophysiology.

Figure 9. Various representations of an embryonic quail heart. The units displayed on the side of all images are in pixels. (A) Averaged composite image of an entire data set. (B) Isochronal activation map of the heart (in sec). Each contour shows 0.02 sec of time. (C) Conduction propagation vector map of the heart. Vector magnitudes are displayed on a logarithmic scale. (D) Contour map of action potential durations across the heart (in msec). Each contour shows 20 msec in time.
**Processing**

Despite significant filtering of abberantly processed points, some small processing error was inevitable because of the nature of the raw data and the use of a polynomial fit. Figure 7 demonstrates that the polynomial fitting method is generally sound for a variety of curves, but tends to be inaccurate for curve shapes with particularly sharp rise times. Further work has suggested that using several cumulative distribution functions (CDFs) may be a better fitting method than split polynomials.

**Comparison to expected electrophysiology**

Several expected features of embryonic heart electrophysiology can be discerned from the example data sets. For example, the large contours in the future atrioventricular region (curve of the heart) in the activation maps in Figures 9B, 10B and 11B as compared to smaller side contours indicate faster conduction propagation.

---

**Figure 10.** Various representations of an alcohol-treated embryonic quail heart. The units displayed on the side of all images are in pixels. (A) Averaged composite image of an entire data set. (B) Isochronal activation map of the heart (in sec). Each contour shows 0.02 sec of time. (C) Conduction propagation vector map of the heart. Vector magnitudes are displayed on a logarithmic scale. (D) Contour map of action potential durations across the heart (in msec). Each contour shows 20 msec in time.
tion in the middle of the heart tube and slower propagation on the ends. Another example can be seen in the activation map in Figure 11B, where the heart is at an angle such that the initiation point is visible and propagation can be seen to emanate from a cluster of cells in this region. One unexpected feature on the activation maps is the wrapping seen on the edges of many of the maps, which is potentially an artifact of mapping the three-dimensional heart onto a two-dimensional image.

**Vector Fields**

Some difficulty was encountered in calculating the propagation vector fields at the intrinsic resolution of the data sets. Results tended to be noisy and the number of points involved resulted in a map that was difficult to read. In an attempt to address this, the activation map matrix was interpolated to reduce resolution. This significantly increased the efficacy of vector generation, and smoothed out much of the noise in the vector field. Additionally, the wide

---

**Figure 11.** Various representations of a saline-treated embryonic quail heart. The units displayed on the side of all images are in pixels. (A) Averaged composite image of an entire data set. (B) Isochronal activation map of the heart (in sec). Each contour shows 0.02 sec of time. (C) Conduction propagation vector map of the heart. Vector magnitudes are displayed on a logarithmic scale. (D) Contour map of action potential durations across the heart (in msec). Each contour shows 20 msec in time.
range of magnitudes made representation somewhat difficult in many cases, and so the example vector fields shown all have their magnitudes shown on a logarithmic scale.

**Conclusion**

We successfully managed to generate several different representations of cardiac conduction from high-noise datasets. From these representations, we can discern some expected features of cardiac conduction, and further study the electrophysiology of the developing heart. Areas in which this work could be improved include development of better curve fitting methods using cumulative distribution functions, increases in speed using multicore processing and further improvement of image representations.

While interesting in and of itself, it is important to note that this work to develop Optical Mapping is primarily intended to create a tool to facilitate further study. Using methods to perturb heart rate and hence hemodynamics, we can study potential routes for the development of congenital defects. This research has the potential to help us to understand how congenital defects form and may allow the development of clinical strategies for prevention and treatment.

**Acknowledgements**

I would like to thank Dr. Andrew Rollins (CWRU) for providing the opportunity to be a part of this work and Dr. Michael Jenkins (CWRU) for his patient explanation as well as thoughtful help developing the methods used here. I would also like to thank Andreas Werdich (CWRU) for providing me with the literature and base code used to develop the conduction vectors. I would also like to thank Dr. Shi Gu, Matt Ford, Lindsy Peterson and other members of the Rollins lab for their feedback and support.

**References**


Appendix: Optical Mapping Processing Code (Matthew McPheeters)

%%% HEADER

%% Optical Mapping Processing Code
%% Calculates Action Potential Durations, Activation Maps and Conduction Velocity Vectors
%%
%% Todo List
%% Add Timing Bar (at end)
%% Reduce Sequential Variables - carry one matrix through pre-processing (at end)
%% Clean up Activation Matrix Plotting (extra triangles and numbers)
%% Use uint16 instead of uint32?
%% Fix how we find lower edge of window - fitting + derivative to identify
%% Overlay Activation Maps on Images
%% Fix lower bound finding
%% Elimination criteria have gotten to be too many!
%% Problem with vector fields is scaling (ie, if one vector is really big, all the rest are impossible to see) - use a logarithm on the velocity perhaps?
%%

tic % Start Timing

% Clear-out Matlab
%close all; % Close all figures
clear all; % Clear all variables
clc; % Clear command window
clf; % Clear all figures

omwait = waitbar(0,'Reading Image Data'); % Initialize Wait Bar

%%% Prepare Parameters

% 1. Image Read
im_read_prefix = 'image_X';
im_read_suffix = '.tif';

image_num = 4000; % Number of images to read
cut_image = 0; % Image number to cut at beginning and end when reading
image_xdim = 128; % Number of pixels in the x-dimension of the image
image_ydim = 128; % Number of pixels in the y-dimension of the image

% Independent Variables / Data Information [Dependent]
sampling_rate = 500; % [Hz]
time_spacing = 1/sampling_rate; % Seconds
time = 1/sampling_rate:1/sampling_rate:(image_num-cut_image)/sampling_rate; % Time Vector

% Bin data
bin = 1; % Bin size

% 2. Normalization
offset = 15; % Points to the start of the averaging window from the beginning of the plot

% 3. Temporal Filtering
Fc = 200; % Cutoff frequency

% 4. Spatial Filtering
% 5. Photobleaching Linear Correction
% 6. Find Interval
AP_intStart = 3.6*sampling_rate; % AP Interval Start Point - randy
AP_intEnd = 4.6*sampling_rate; % AP Interval End Point - randy
max_add = 20; % Number of point past the max to add to the rising edge interval
max_threshold = 1; % Threshold = N * Stdev (where N is this number)

% 7. Fit Edges
threshold_multiplier = 2; % Threshold = N * Stdev (where N is this number)

% 8. Measure Action Potential Durations
% 9. Gate Out Noise Points
threshold = 1500; % Threshold for initial data not to be considered noise
dur_threshold_factor = 2; % Threshold multiplier to gate out durations vs. the average duration

% 10. Calculate Conduction Propogation Velocity Vectors
do_vectors = 1; % If =1, will do vector calculations and plotting - can take a long time
pixelinterval = 4; % Pixel Interpolation
% 11. Gate out noise points, compared to average (Velocity Vectors)
vector_gate_factor = 20;
% 12. Plot Conduction Velocity Vectors

% 13. Plot Action Potential Durations
dur_percentile_cutoff = 5; % Percentile to cut off of contour vector
dur_contour_res = 20; % Contour Resolution in msec
% 14. Plot Activation Map
AP_percentile_cutoff = 2; % Percentile to cut off of contour vector
AP_contour_res = 0.020; % Contour Resolution in sec
% 15. Plot Averaged Image

%% 1. Read Image Data
image_data = zeros(image_xdim,image_ydim,image_num); % Make empty array
for i = cut_image+1:image_num-cut_image*2
    image_data(:,:,i) = imread([im_read_prefix num2str(i) im_read_suffix]);
end
step1_data = image_data; % TEMP DATA

% 2% Bin data if necessary (move to after normalization?)
if bin > 1
    bin_image_data = zeros(floor(size(image_data,1)/bin),floor(size(image_data,2)/bin),size(image_data,3));
    for k = 1:size(image_data,3)
        bin_image_data(:,:,k) = imresize(squeeze(image_data(:,:,k)),[floor(size(image_data,1)/bin) floor(size(image_data,2)/bin)],'bilinear');
    end
    step1_data = bin_image_data; % TEMP DATA
end
\[
\text{for } i = 1:1:\text{floor(size(image\_data,1)/bin)} \\
\text{for } j = 1:1:\text{floor(size(image\_data,2)/bin)} \\
\text{bin\_image\_data}(i,j) = \text{mean}(\text{mean}(image\_data(i:i+bin-1,j:j+bin-1,k))); \\
\text{end} \\
\text{end} \\
\text{end} \\
\text{image\_data = bin\_image\_data;} \\
\text{image\_xdim = floor(size(image\_data,1)/bin);} \quad \% \text{Number of pixels in the x-dimension of the image} \\
\text{image\_ydim = floor(size(image\_data,2)/bin);} \quad \% \text{Number of pixels in the y-dimension of the image} \\
\]
image_data(i,j,:) = reshape(temp_data,1,1,image_num);
end
waitbar(0.10+0.05*i/image_xdim,omwait);
end
step4_data = image_data; % TEMP DATA

%% 5. Spatial Filtering

waitbar(0.15,omwait,'Apply Spatial Filter');
%%

%filter_spatial = fspecial('gaussian', [3, 3], 1); % 3x3 pixel Gaussian filter with a std dev = 1
%filter_spatial = fspecial('average', [3, 3]); % 3x3 pixel averaging filter
%filter_spatial = fspecial('average', [5, 5]); % pixel averaging filter
filter_spatial = fspecial('disk', 3); % disk pixel averaging filter, radius = 3 pixels

for m = 1:1:image_num
  image_data(:,:,m) = filter2(filter_spatial,image_data(:,:,m));
end
step5_data = image_data; % TEMP DATA

%% 6. Find Interval

waitbar(0.2,omwait,'Find the Interval the Action Potentials Occur In');
%%

divide_pt = zeros(image_xdim,image_ydim); % Pre-allocate array

for i = 1:1:image_xdim
  for j = 1:1:image_ydim
    temp_int = image_data(i,j,AP_intStart:AP_intEnd);
    temp_int = smooth(temp_int,10,'moving');
    if max(temp_int) > max_threshold * std(image_data(i,j,AP_intStart-20:AP_intStart))
      divide_pt(i,j) = find(temp_int == max(temp_int),1) + max_add; % Only both to do any of this if the max is bigger than the variance * N
      if divide_pt(i,j) < (AP_intEnd-AP_intStart)*0.2
        divide_pt(i,j) = 0;
      elseif divide_pt(i,j) > (AP_intEnd-AP_intStart)*0.8
        divide_pt(i,j) = 0;
      end
    else
      divide_pt(i,j) = 0; % I know that this was zero to begin with, I said it explicitly to make my logic clear to you
    end
  end
  waitbar(0.2+0.05*i/image_xdim,omwait)
end

%% 7A. Fit to Polynomial on Rising Edge of Action Potential, Find 20% and 50% Point

waitbar(0.25,omwait,'Fit the Rising Edge of the Action Potentials');
%%
time = 1/sampling_rate:1/sampling_rate:image_num/sampling_rate; % Time Vector - Here temporarily b/c of prob-
lems
action_potentials = zeros(image_xdim,image_ydim); % Create empty array for storing the time that action potentials occur
rising_edge = zeros(image_xdim,image_ydim); % Create empty array for storing the beginning of the width
percent20_values = zeros(image_xdim,image_ydim); % Store 20% values to pass to falling edge
percent50_values = zeros(image_xdim,image_ydim); % Store 50% values to pass to falling edge
avg_variance = zeros(image_xdim,image_ydim); % Store average variance values to pass to falling edge

degree = 10; % Degree of the fit polynomials - more than 5 seems to make the derivative too noisy

for i = 1:1:image_xdim
    for j = 1:1:image_ydim
        if divide_pt(i,j) ~= 0
            % % that's done in the loops because of the variable intervals
            poly_min = AP_intStart;
            poly_max = AP_intStart + divide_pt(i,j);
            center_pt = poly_min + ceil((poly_min + poly_max) / 2);
            center_time = center_pt / sampling_rate;
            time = time - center_time; % Center time values
            % Fit data
            polyfit_coeff = polyfit(time(1,poly_min:poly_max),squeeze(image_data(i,j,poly_min:poly_max))',degree);
            fitted = zeros(1,size(time(1,poly_min:poly_max),2)); % Fitted Data
            for k = 0:1:degree
                fitted = fitted + polyfit_coeff(1,k+1) * ( time(1,poly_min:poly_max) ) .^ (degree - k); % Add results of each
            end
            sec5_fitted = floor(size(fitted,2)/20);
            % FIX HOW THIS IS FOUND
            bottom_edge = mean(fitted(1,sec5_fitted*2:sec5_fitted*3)); % Take the mean of 10% to 15% of the selection
            top_edge = max(fitted(round(length(fitted)*0.1):round(length(fitted)*0.9))); % Only search the middle 80%
            avg_variance(i,j) = (std(fitted(1,sec5_fitted*2:sec5_fitted*3)) + var(fitted(1,sec5_fitted*1:sec5_fitted*18)))/2;
            percent20_values(i,j) = bottom_edge + (top_edge - bottom_edge)/5; % Find the 20% value
            percent50_values(i,j) = bottom_edge + (top_edge - bottom_edge)/2; % Find the 50% value
            fitted_absnorm20 = abs(fitted - percent20_values(i,j));
            fitted_absnorm50 = abs(fitted - percent50_values(i,j));
            start_point = find(fitted_absnorm20 == min(fitted_absnorm20),1); % !! Is unclear if we want to just choose
            activation_point = find(fitted_absnorm50 == min(fitted_absnorm50),1); % !! Is unclear if we want to just choose
            % Find and record action potential location
            action_potentials(i,j) = time(1,(poly_min + activation_point - 1)); % Subtract 1 because first value of deriv IS

        end
    end
end
poly_min (depending upon which way you take the derivative to be).
   rising_edge(i,j) = time(1,(poly_min + start_point - 1));
else
   action_potentials(i,j) = 0; % If 50% is not greater than threshold, record 0
   rising_edge(i,j) = 0; % If 50% is not greater than threshold, record 0
end

% Had to put this in the loops because of the variable intervals
rising_edge(i,j) = rising_edge(i,j) + center_time;
action_potentials(i,j) = action_potentials(i,j) + center_time;
time = time + center_time; % Un-center time values
else
   action_potentials(i,j) = 0; % I know that this was zero to begin with, I said it explicitly to make my logic clear
to you
   rising_edge(i,j) = 0; % I know that this was zero to begin with, I said it explicitly to make my logic clear to you
end
waitbar(0.25+0.1*i/image_xdim,omwait);
end

%% 7B. Fit to Polynomial on Falling Edge of Action Potential, Find 20% Point

waitbar(0.35,omwait,'Fit the Falling Edge of the Action Potentials');
%%
time = 1/sampling_rate:1/sampling_rate:image_num/sampling_rate; % Time Vector - Here temporarily b/c of prob-
-lems
falling_edge = zeros(image_xdim,image_ydim); % Create empty array for storing the time that falling edges occur
degree = 10; % Degree of the fit polynomials - more than 5 seems to make the derivative too noisy

for i = 1:1:image_xdim
   for j = 1:1:image_ydim
      if divide_pt(i,j) ~= 0
         % Had to put this in the loops because of the variable intervals
         % Set region to look for falling edge of action potential
         poly_min = AP_intStart + divide_pt(i,j) + 1;
         poly_max = AP_intEnd;
         center_pt = poly_min + ceil( (poly_min + poly_max) / 2 );
         center_time = center_pt / sampling_rate;
         time = time - center_time; % Center time values
         if percent50_values(i,j) > avg_variance(i,j)*threshold_multiplier; % Don’t bother with the rest of the loop un-
   less this is true
         % Fit data
         polyfit_coeff = polyfit(time(1,poly_min:poly_max),squeeze(image_data(i,j,poly_min:poly_max))',degree);
         fitted = zeros(1,size(time(1,poly_min:poly_max),2)); % Fitted Data
         for k = 0:1:degree
            fitted = fitted + polyfit_coeff(1,k+1) * ( time(1,poly_min:poly_max) ) .^ (degree - k); % Add results of each
   order one by one, so n can be changed easily
         end
      end
   end
   waitbar(0.25+0.1*i/image_xdim,omwait);
end
fitted_absnorm = abs(fitted - percent20_values(i,j));
falling_point = find(fitted_absnorm == min(fitted_absnorm),1); % !! Is unclear if we want to just choose the first value

falling_edge(i,j) = time(1,poly_min + falling_point - 1); % Subtract 1 because first value of deriv IS poly_min (depending upon which way you take the derivative to be).
else
    falling_edge(i,j) = 0; % If 50% is not greater than threshold, record 0
end

% Had to put this in the loops because of the variable intervals
falling_edge(i,j) = falling_edge(i,j) + center_time;
time = time + center_time; % Un-center time values
else
    falling_edge(i,j) = 0; % I know that this was zero to begin with, I said it explicitly to make my logic clear to you
end

waitbar(0.35+0.1*i/image_xdim,omwait);

waitbar(0.45,omwait,'Measure Action Potential Durations');

%% 8. Measure Action Potential Durations

durations = zeros(image_xdim,image_ydim); % Create empty array for storing the time that falling edges occur
durations = falling_edge - rising_edge;
durations = 1000.*durations; % Convert sec to msec

durations = 0;  % Set measured duration to zero

%% 9. Gate out noise points (Action Potentials and Durations)

waitbar(0.5,omwait,'Gate Out Noise Activation Points and Action Potential Durations');

avg_duration = mean(mean(durations)); % Find mean action potential duration

% For loop to gate out noise points (Action Potentials and Durations)
for i = 1:image_xdim
    for j = 1:image_ydim
        if durations(i,j) > 10000 % sec
            durations(i,j) = 0;
        end
    end
end

% For loop to gate out noise points (Action Potentials and Durations)
for i = 1:image_xdim
    for j = 1:image_ydim
        if action_potentials(i,j) < threshold
            action_potentials(i,j) = 0; % Set activation time to zero
        end
    end
end
elseif average_value(i,j) > threshold && percent50_values(i,j) > avg_variance(i,j)*threshold_multiplier
    action_potentials(i,j) = action_potentials(i,j);
durations(i,j) = durations(i,j);
elseif average_value(i,j) > 15990 % Saturation Gating
    action_potentials(i,j) = 0; % Set activation time to zero
durations(i,j) = 0; % Set measured duration to zero
end

% Set durations to 0 if they’re unreasonable large - should vary this
if durations(i,j) > dur_threshold_factor*avg_duration % sec
    durations(i,j) = 0;
end

%% 10. Calculate Conduction Propagation Velocity Vectors
if do_vectors == 1
    waitbar(0.55,omwait,’Calculate Conduction Propagation Velocity Vectors’);

    M = action_potentials; % Input action potential activation times

    FrameXLim = [3 image_xdim-2]; % Define Region of Interest (ROI) in X
    FrameYLim = [3 image_ydim-2]; % Define Region of Interest (ROI) in Y

    estvel = 0.0005; % Pixels/sec - This is supposed to be entered manually, but perhaps it could be calculated?

    % CVV Signal is a Boolean Matrix that tells the CVV function where to calculate vectors
    SIGNAL = zeros(image_xdim,image_ydim); % Pre-allocate
    for i = 1:1:image_xdim
        for j = 1:1:image_ydim
            SIGNAL(i,j) = (action_potentials(i,j) > 0);
        end
    end

    % Run Function
    [VFRAMEi,VFRAMEj,RMSEFRAME,XFRAMEINTERP_0,YFRAMEINTERP_0,SIGNALFRAMEINTERP_0] = fitpixels7(M,FrameXLim,FrameYLim,estvel,SIGNAL,pixelinterval);

    %% 11. Gate out noise vectors, compared to average (Velocity Vectors)
    waitbar(0.6,omwait,’Gate Out Noise Propagation Vectors’);

    total_vect = 0; % Initialize

    % Find average vector length
    for i = 1:size(XFRAMEINTERP_0,1)
        for j = 1:size(XFRAMEINTERP_0,2)
total_vect = total_vect + sqrt(VFRAMEi(i,j)^2 + VFRAMEj(i,j)^2);
end
end
avg_vect = total_vect / (size(XFRAMEINTERP_0,1).*size(XFRAMEINTERP_0,2)); % Sum of all vector lengths divided by the total number of vectors

VFRAMEi_2 = VFRAMEi; % Create Punching Bag
VFRAMEj_2 = VFRAMEj; % Create Punching Bag

% Gate out large vectors
for i = 1:size(XFRAMEINTERP_0,1)
    for j = 1:size(XFRAMEINTERP_0,2)
        if sqrt(VFRAMEi(i,j)^2 + VFRAMEj(i,j)^2) > avg_vect*vector_gate_factor
            VFRAMEi_2(i,j) = 0;
            VFRAMEj_2(i,j) = 0;
        end
    end
end

waitbar(0.8,omwait,'Plot Conduction Velocity Vectors');

%% 12. Plot Conduction Velocity Vectors
figure(3);
quiver(XFRAMEINTERP_0,YFRAMEINTERP_0,VFRAMEi,VFRAMEj);
quiver(XFRAMEINTERP_0,YFRAMEINTERP_0,VFRAMEj_2,VFRAMEi_2);
xlim([1 image_xdim]); ylim([1 image_ydim]);
xlabel('x-dimension (pixels)'); ylabel('y-dimension (pixels)'); title('Cardiac Propagation Velocity Vectormap');
end % do_vectors conditional loop ends here

%%
waitbar(0.85,omwait,'Plot Action Potential Durations');

%% 13. Plot Action Potential Durations
non_zero_durations = durations(durations~=0);
min_durations = ceil(prctile(non_zero_durations,dur_percentile_cutoff)*100)/100;
max_durations = floor(prctile(non_zero_durations,100-dur_percentile_cutoff)*100)/100;
duration_position = min_durations:dur_contour_res:max_durations; % Vector defines contours shown on activation map
figure(1); colormap(jet);
[cs1, h1] = contourf(durations,duration_position);
colorbar; % The colorbar option
clabel(cs2, h2, duration_position); % The labeling option
xlabel('x-dimension (pixels)'); ylabel('y-dimension (pixels)'); zlabel('Time (msec)'); title('Action Potential Durations');

%%
waitbar(0.9,omwait,'Plot Activation Map');
%% 14. Plot Activation Map

```
non_zero_action_potentials = action_potentials(action_potentials~=0);
min_action_potentials = ceil(prctile(non_zero_action_potentials,AP_percentile_cutoff)*100)/100;
max_action_potentials = floor(prctile(non_zero_action_potentials,100-AP_percentile_cutoff)*100)/100;
cont_position = min_action_potentials:AP_contour_res:max_action_potentials; % Vector defines contours shown on activation map
```

```
figure(2); colormap(jet);
[cs2, h2] = contourf(action_potentials,cont_position);
colorbar; % The colorbar option
%clabel(cs2, h2, cont_position); % The labeling option
xlabel('x-dimension (pixels)'); ylabel('y-dimension (pixels)'); title('Isochronal Activation Map');
```

%% 15. Plot Averaged Image

```
avg_image = flipud(mean(image_data,3));
figure(4);
imshow(mat2gray(avg_image));
time(15) = toc;
```

%% End

```
waitbar(0.95,omwait,'Plot Averaged Image');
```

%% 15. Plot Averaged Image

```
avg_image = flipud(mean(image_data,3));
figure(4);
imshow(mat2gray(avg_image));
time(15) = toc;
```

%% End

```
waitbar(1,omwait,'Done!');
close(omwait); % Close Wait Bar
toc % End Timing
```
Appendix: Conduction Propagation Vector Code (Andreas Werdich & Matthew McPheeters)

%% Estimates conduction velocity components VFRAMEi,VFRAMEj in matrix space (i,j) for region of interest (ROI) defined by FrameXLim and FrameYLim.
% M is activation matrix (activation times) and estvel is an estimated
% conduction velocity that is manually determined for each ROI based on the
% distance of a few isochrones. SIGNAL is a binary image of size M. Only
% pixels with SIGNAL=1 are used in the estimation of conduction velocity.
function [VFRAMEi,VFRAMEj,RMSEFRAME,XFRAMEINTERP_0,YFRAMEINTERP_0,SIGNALFRAMEINTERP_0]=fitpixels6(M,FrameXLim,FrameYLim,estvel,SIGNAL,pixelinterval)

%% Define ROI
internal=1;%USE MATLAB2009 FIT FUNCTION
estvel=mean(NEWDISTANCE)/dt; %[pixels/frame]
borderpixels=2;%extend frame area to provide neighbors for border pixels
pixelinterval=2;%interpolation

%cut frame if there is no room for borderpixel
if FrameXLim(1)<=borderpixels,FrameXLim(1)=3;end
if FrameXLim(2)>(size(M,2)-borderpixels),FrameXLim(2)=size(M,2)-borderpixels;end
if FrameYLim(1)<=borderpixels,FrameYLim(1)=3;end
if FrameYLim(2)>(size(M,1)-borderpixels),FrameYLim(2)=size(M,1)-borderpixels;end

XF=[FrameXLim(1)-borderpixels:FrameXLim(2)+borderpixels]’;
YF=[FrameYLim(1)-borderpixels:FrameYLim(2)+borderpixels]’;
[XFRAME,YFRAME]=meshgrid(XF,YF);
ZFRAME=M(YFRAME(1,1):YFRAME(end,1),XFRAME(1,1):XFRAME(1,end));%activation times

%Mark signals in frame+border
SIGNALFRAME=SIGNAL(YF(1):YF(end),XF(1):XF(end));%extended frame
%% interpolate activation times in frame
%create new grid
%interpolate activation map
ZFRAMEINTERP=interp2(XFRAME,YFRAME,ZFRAME,XFRAMEINTERP,YFRAMEINTERP);
%interpolate SIGNALFRAME
SIGNALFRAMEINTERP=interp2(XFRAME,YFRAME,SIGNALFRAME,XFRAMEINTERP,YFRAMEINTERP);
SIGNALFRAMEINTERP(SIGNALFRAMEINTERP<1)=0;
%correlate SIGNALFRAMEINTERP with ZFRAMEINTERP
ZFRAMEINTERP(SIGNALFRAMEINTERP==0)=0;

%% fitting algorithm
%definitions outside look to save time

%initialize velocity components that include border
Vi=zeros(size(ZFRAMEINTERP));
Vj=zeros(size(ZFRAMEINTERP));
RMSE=zeros(size(ZFRAMEINTERP));

minpixels=6;%minimum number of pixels required for fit
minfitpixels=20;%minimum number of pixels desired for fit
%create fitoptions object
if internal==1
    OPTIONS = fitoptions(’poly22’);
end
%modify options structure
%lower boundaries
%L=[0,-Inf,-Inf,-Inf,-Inf,-Inf];
%U=[LIST(end),Inf,Inf,Inf,Inf,Inf];
% set(OPTIONS,’Lower’,L,’Upper’,U);
opts.Weights = zeros(1,0);

hd1 = waitbar(0,’Fitting pixels’);
for i=1:size(ZFRAMEINTERP,1)
    for j=1:size(ZFRAMEINTERP,2)
        %% LOOP
        if SIGNALFRAMEINTERP(i,j)==1
            %time interval
            atime=ZFRAMEINTERP(i,j);
            %to improve wavefrontsearch, increase n by 1 if
            %length(NWFRONT)<minfitpixels
            repeatwavefrontsearch=1;n=1;
            while repeatwavefrontsearch==1 && n<=3
                SEARCHINTERVAL=[atime-n*minpixels/estvel/2,atime+n*minpixels/estvel/2];
                %find all pixels within the SEARCHINTERVAL
                WFRONT=[];
                for s=1:size(ZFRAMEINTERP,1)
                    for t=1:size(ZFRAMEINTERP,2)
                        if SEARCHINTERVAL(1)<=ZFRAMEINTERP(s,t) && ZFRAMEINTERP(s,t)<=SEARCHINTERVAL(2)
                            if SIGNALFRAMEINTERP(s,t)==1
                                WFRONT=[WFRONT;[s,t,ZFRAMEINTERP(s,t)]];
                            end
                        end
                    end
                end
                %define wavefront by reducing the number of WFRONT pixels to those that are
                %near the pixel of interest
                NWFRONT=[];
                for k=1:size(WFRONT,1)
                    pixeldistance=norm([WFRONT(k,1)-i,WFRONT(k,2)-j],2);
                    if pixeldistance<=minpixels/2+n-1
                        NWFRONT=[NWFRONT;WFRONT(k,:)];
                    end
                end
                if size(NWFRONT,1)<minfitpixels
                    repeatwavefrontsearch=1;
                    n=n+1;
                else
                    repeatwavefrontsearch=0;
                end
            end
        end
    end
end

22
% fit NWFRONT pixels
% prepare fit
% ALL POINTS
X = NWFRONT(:,2); Y = NWFRONT(:,1); Z = NWFRONT(:,3);
% fit
if size(NWFRONT,1)<minpixels
    % not enough points to perform fit
    RMSE(i,j)=0;
else
    if internal==1
        % use fit function included in Matlab Ver>2009
        [fitresult, gof] = fit([X,Y],Z,'poly22',OPTIONS);
        RMSE(i,j)=gof.rmse;
    end
    % calculate velocity components from fit
    % fitresult(x,y) = p00 + p10*x + p01*y + p20*x^2 + p11*x*y + p02*y^2
    p00=fitresult.p00;
    p10=fitresult.p10;
    p01=fitresult.p01;
    p20=fitresult.p20;
    p11=fitresult.p11;
    p02=fitresult.p02;
    % gradZ(x,y)=[Tx,Ty]
    Tx=p10+p11*i+2*p20*j;
    Ty=p01+p11*j+2*p02*i;
    Vx=Tx/(Tx^2+Ty^2);
    Vy=Ty/(Tx^2+Ty^2);
    Vi(i,j)=pixelinterval*Vy;% velocity in y-direction (row)
    Vj(i,j)=pixelinterval*Vx;% velocity in x-direction (column)
else
    % use external fit function
    p=polyfitn([X(:),Y(:)],Z,2);
    % goodness of fit
    D=(Z(1)-polyvaln(p,[X(1),Y(1)]))^2;
    for n=2:length(Z)
        D=D+(Z(n)-polyvaln(p,[X(n),Y(n)]))^2;
    end
    RMSE(i,j)=sqrt(D)/length(Z);
    P=p.Coefficients;
    % MODEL: T(x,y)=P(1)*X1^2 + P(2)*X1*X2 + P(3)*X1 + P(4)*X2^2 + P(5)*X2 + P(6)
    Tx=P(1)*2*j+P(2)*i+P(3);
    Ty=P(4)*2*i+P(2)*j+P(5);
    Vx=Tx/(Tx^2+Ty^2);
    Vy=Ty/(Tx^2+Ty^2);
    Vi(i,j)=pixelinterval*Vy;% velocity in y-direction (row)
    Vj(i,j)=pixelinterval*Vx;% velocity in x-direction (column)
end
end
end
waitbar(i/size(ZFRAMEINTERP,1),hdl)
end
close(hdl)

%% prepare output of vector components
xmin=ceil(borderpixels/pixelinterval)+0.5;
ymin=xmin;
w=size(XFRAMEINTERP,2)-2*xmin+1-0.5;
h=size(YFRAMEINTERP,1)-2*ymin+1-0.5;

% crop velocity vector matrices
VFRAMEi=imcrop(Vi,[xmin,ymin,w,h]);
VFRAMEj=imcrop(Vj,[xmin,ymin,w,h]);
RMSEFRAME=imcrop(RMSE,[xmin,ymin,w,h]);
XFRAMEINTERP_0=imcrop(XFRAMEINTERP,[xmin,ymin,w,h]);
YFRAMEINTERP_0=imcrop(YFRAMEINTERP,[xmin,ymin,w,h]);
SIGNALFRAMEINTERP_0=imcrop(SIGNALFRAMEINTERP,[xmin,ymin,w,h]);
framepixels=floor(bwarea(SIGNALFRAMEINTERP_0)); % total number of active pixels in frame