Movement Analysis in Heart Tissue

Students:
David Berestetsky, 318269495
Elad Tsur, 208437285
Ofir Eldar, 314722950

Supervisor:
Noam Rotstein

In collaboration with Dr. Oren Caspi’s lab
Faculty of medicine, Technion

Project duration:
Winter 22-23 - Spring 23

Report Date: June 25, 2023
Table of Contents

**Introduction** .......................................................................................................................... 3

**Project Goal** .......................................................................................................................... 3

**Theory and Background** ......................................................................................................... 3

**Programming Tools:** ........................................................................................................... 4

**Pre – processing** ................................................................................................................... 4

- BW thresholding ....................................................................................................................... 5
- connected-components .............................................................................................................. 5
- for finding the scale bar ............................................................................................................ 5

**Methods** .................................................................................................................................. 6

**Method 1- Follow the rectangles perimeter** ............................................................................ 6

- Detect edges: Canny .................................................................................................................. 7
- Detect lines: Hough transform .................................................................................................. 7
- Filter lines: .................................................................................................................................. 8
- Clustering: ................................................................................................................................. 8
- Linear regression: Ransac .......................................................................................................... 9

**Method 2- Template matching of the rectangles** .................................................................... 11

**Method 3- Points tracking with optical flow** ......................................................................... 12

- Lucas-Kanade method .............................................................................................................. 13
- Processing the results from optical flow: ................................................................................ 13
- Kernel density estimation ........................................................................................................ 14

**Post-processing** ................................................................................................................... 15

**Results** .................................................................................................................................... 16

**Validation** .............................................................................................................................. 20

**Summery** .................................................................................................................................. 20

**Bibliography** .......................................................................................................................... 21
Introduction

**Project Goal**
Optimization and Automation of Movement recognition on pre-captured lab videos.

The program will analyze the active force of the captured tissue, by recognizing movement of rectangular-silicone implants inside of it.

**Theory and Background**
Stem cells are being cultivated in Dr. Oren Caspi's medical research laboratory at the faculty of medicine. These stem cells are specifically categorized as cardiomyocytes. Using these cardiomyocytes, 3D tissues are constructed around two silicon rectangles. The movement of the rectangles is driven by the pulses generated by the tissues. It is established that there exists a direct correlation between the displacement of the columns and the level of force exerted by the tissue.

Our goal is to find the force based on the following equation:

\[ F(\delta) = \left( \frac{6EI}{-x^3 + 3Lx^2} \right) \delta \]

Where:
- \( E \) – Elasticity of the material
- \( L \) – Length of columns
- \( I \) – Width of columns
- \( x \) – Location of the tissue relative to the columns
- \( \delta \) – Displacement of the columns

Videos of the tissues are being captured using a robotic microscopy system.

96 samples are taken per experiment.

The videos are in a multi-TIFF format which will serve as the input to our software.
Programming Tools:
- OpenCV for Image processing tools.
- Matplotlib for Plotting utilities.
- Sklearn for statistical tools.
- Scipy and PeakUtils for signal processing
- Numpy

Pre – processing
In the pre-processing stage, we identify the general area of the rectangles using image thresholding and find connected components.

(1) Original —> (2) BW —> (3) Remove the white background around the tissue

(4) Splitting the image in half and taking each rectangle separately (from images 2&3)

(5) Taking the biggest component and expand it

(6) Using img5 as mask on img1, leaves us with the general area of a single rect.
BW thresholding

For BW thresholding we use open-cv threshold function:

Applies fixed-level threshold to a multiple-channel array. The function is typically used to get a bi-level (binary) image out of a grayscale image.

In our implementation we use Otsu’s method\[1\] for determining the optimal threshold value.

In order to do so, the cv.threshold() function is used, where cv.THRESH_OTSU is passed as an extra flag.

carded-components

To find connected-components we use open-cv connectedComponentsWithStats\[2\] function:

In our implementation we use the default method available in opencv, Bolelli (Spaghetti) algorithm\[3\]

Finding the scale bar

The algorithm for finding the scale bar in the image is:

1. Take a patch of the lower right quarter of the first frame, where the scale bar should be located.
2. Convert the patch to a binary image using cv.threshold
3. find contours using cv.findContours
4. for each contour:
   4.1 use cv.approxPolyDP to approximate the contour with a polygon
   4.2 If the polygon approximation has 4 sides and is convex (using cv.isContourConvex):
      4.2.1 classify the contour as a rectangle
5. for each contour classified as a rectangle:
   5.1 calculate the area using cv.contourArea
6. classify the contour with maximal area as the scale bar

The documentation of the various open-cv methods used in this algorithm can be found at \[4\]

We use the found area to calculate the pixel to micron ratio, the scale of the scale bar is set to 200 microns by default and is configurable by the user of the software.
In addition, if we cannot find the scale bar (sometimes it simply doesn’t exist) we set the default pixel to micron ratio to 1.

Methods
We tried 3 different methods of detecting the rectangles movement:

1. Follow the rectangles perimeter.
2. Perform template matching of the rectangles.
3. Follow specific points inside the rectangles using optical flow.

Our final chosen method is method number 3

Method 1- Follow the rectangles perimeter
For each frame in the video we detect the edges of the rectangles, we output the position of the lines throughout the video.

Our pipeline of detection:

Detect edges: Canny → Detect lines: Hough transform → Filter lines

Clustering: K means → Linear regression: Ransac

This pipeline is applied on each frame. For each frame we output the rectangles edges using line parameters:

\( \rho \) - distance from the left upper corner of the image.

\( \theta \) - the angle of the line relative to the horizontal axis.
Detect edges: Canny
Detecting all the edges on the rectangle with Canny Detector from OpenCV library [9].

The algorithm works in few stages:
1. Finds horizontal and vertical gradients, from which an edge gradient and angle are being calculated for each pixel.
2. Finds local maximums (edge candidates).
3. Placing two thresholds
4. Choosing the convex candidates above the max value, and the candidates between the thresholds if they are connected to candidates above the max value.

Detect lines: Hough transform
Hough transform is applied on the edges, we use open-cv HoughLines function.

This transform involves passing lines from the image to a parameter space and finding their intersections. Each point on a line in the image space corresponds to a sinusoid in the parameter space. By finding the intersections of these sinusoids, we obtain sets of θ (angle) and ρ (distance from the origin) values that represent the line segments of the rectangles. To ensure consistency, we convert the ρ values to positive values and restrict the θ values to a range between 0 and 2π.
Filter lines:

Once we obtain all the lines from the Hough transform, we apply a filtering process to remove lines that do not have a perpendicular counterpart. By selecting a threshold, we identify pairs of lines with an angular difference of approximately π/2 (plus or minus the threshold), indicating that they are perpendicular to each other. These filtered lines are then considered as part of the rectangle structure.

Clustering:

In order to sort the filtered lines to the 3 rectangle edges we used the k means algorithm [10] to gather our lines to clusters. The algorithm defines k random means for each line, the algorithm finds the closest mean by calculating the Euclidean distance for each of the means. After the first step the algorithm will recalculate the k means. This process will repeat itself until there will be no change to the classification. The final classification of the lines will be returned as the clusters. In our case k=4, for the 4 rectangle edges groups we want to detect.

Double Clustering:

After applying K-means we saw that the clusters can be divided to 2 sets each. As can be seen in the image above, the horizontal clusters can intuitively be divided into an “upper” and “lower” clusters, and the vertical clusters can be divided into “inner” and “outer” clusters.

We decided to re-apply the k-means algorithm on each cluster and divide each cluster to 2 different clusters.

We had 2 approaches for choosing the best cluster out of each pair of clusters:
1. Take the cluster with most lines, this method was inconsistent and gave lot of fluctuations in the graphs.
2. Take the inner cluster for each rectangle edge.

Method 2 turned out to be more stable.
Linear regression: Ransac

For each edge group detected in the Clustering step, we apply Ransac linear regression \cite{11} to get the most representative edges.

Ransac is an iterative method to estimate parameters of a model from a set of data points that contains outliers. Generally speaking, consists of two steps:

1. Sample a subset of the data points and compute model parameters based on this subset (in our case a line).
2. Count outliers, a data element will be considered as an outlier if it does not fit the model within some error threshold.

Those steps are repeated until enough inliers are found or max number of iterations is reached.

Simple linear regression vs. RANSAC

(a) Result by linear regression

(b) Result by proposed RANSAC
Method outputs:

This method manages to find the rectangles perimeter, but it is un-stable across many frames and struggles to follow the rectangles movement, adding noise and false detection of rectangles edges in some frames.
Method 2- Template matching of the rectangles
For each frame in the video we slide a template image over the input image (as in 2D convolution), returns a grayscale image, where each pixel denotes how much does the neighborhood of that pixel match with the template.
We create rectangle templates using the detection we described in method 1 in small variations of rotation degree and scale.
We use OpenCV’s template matching implementation, with normalized cross-correlation as a measurement method.

Although this method proved successful in finding the rectangles across frames, it was not stable enough.
Method 3- Points tracking with optical flow
After the pre-processing we take special pixels inside the rectangle area we detected and follow them through the video with **Lucas-Kanade** optical flow algorithm [5].

**Optical flow** is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera.

Optical flow works on several assumptions:
1. The pixel intensities of an object do not change between consecutive frames.
2. Neighboring pixels have similar motion.

In our case, we look for the motion of the rectangles. We find “good tracking points” inside the rectangle and then follow them with Lucas-Kanade optical flow algorithm.

In the general optical flow theory, Using the 2 assumption we get this equation:

\[ I(x, y, t) = I(x + dx, y + dy, t + dt) \]

Where:
\[ I \] - The intensity, \( x, y \) - the pixel coordinates, \( t \) - the time/ frame.

With Taylor approximation we will get the **optical flow equation**:

\[ f_x u + f_y v + f_t = 0 \]

Where:
\[ f_x = \frac{\partial f}{\partial x}, \quad f_y = \frac{\partial f}{\partial y}, \quad u = \frac{dx}{dt}, \quad v = \frac{dy}{dt} \]
Lucas-Kanade method uses another assumption:

3. all the neighboring pixels will have similar motion.

Using this method, we take a 3x3 patch around a chosen point, so all the 9 points have the same motion. We can find \((f_x, f_y, f_t)\) for these 9 points. So now our problem becomes solving 9 equations with two unknown variables which is over-determined. A LS solution for these equations will be:

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} = \begin{bmatrix}
  \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\
  \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2
\end{bmatrix}^{-1}
\begin{bmatrix}
  -\sum_i f_{x_i} f_{t_i} \\
  -\sum_i f_{y_i} f_{t_i}
\end{bmatrix}
\]

After solving this LS problem, we can determine the pixel location in the next frame.

Detection of tracking points:

We search for gray-scale pixels inside the rectangle. These pixels are good tracking points because they satisfy the assumption of the optical flow algorithm.

The detection of gray-scale pixels is done with open-cv adaptiveThreshold function: the function determines the threshold for a pixel based on a small region around it. So we get different thresholds for different regions of the same image which gives better results for images with varying illumination.

Processing the results from optical flow:

Our optical flow algorithm effectively traces a substantial number of points within each rectangle, ranging from several hundred to a few thousand.

We must aggregate the movement of points within each rectangle to derive a representative value that captures the overall movement of the rectangle in each frame.

To accomplish this, we employ Gaussian kernel density estimation on all the tracked points, followed by determining the maximum value obtained from the KDE.
Kernel density estimation

Kernel Density Estimation (KDE) [6] is a statistical technique used to estimate the probability density function of a random variable.

It provides a smooth, continuous estimation of the underlying distribution from a set of observed data points.

We center a kernel function at each data point and then take their average, in our case we use a Gaussian kernel: \( K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \)

In fact, a scaled version of the kernel is used: \( K_h(x) = \frac{1}{h} K\left(\frac{x}{h}\right) \)

\( h \) is called the bandwidth of the kernel; we can see an example of how different values of \( h \) affect the shape of the kernel function:

![Different bandwidth of the kernel](image-url)

Image is taken from: A gentle introduction to kernel density estimation | Let’s talk about science! (ekamperi.github.io)
Post-processing
After we get the motion of the rectangles for each frame according to KDE method, we get a raw signal:

We arrange the signal to be with zero baseline [7] and pulses with maximum orientation:
Results
The results that our method achieved correspond to previously hand-calculated results from the medical researchers and correspond with their theoretical expectations.

When we discuss the results the distance and force can be used interchangeably, since as we saw at the background and theory section, the force is linearly dependent on the distance.

We present here three sample cases that illustrate:

1. The common case – a significant movement of the tissue.
2. Video starting at the middle of a pulse
3. Very small movement of the tissue.

Examples 2 and 3, demonstrate two of the challenges we dealt with during this project and as we will see, our method achieves good result in these cases as well.

In this example, we can see that due to the significant movement of the tissue (and therefore of the rectangles) the force (and also the distance per frame) plot comes out relatively not-noisy, which enables good detection of the start, end and peak frames for each pulse.
In this example we see that the value of the force, doesn't start at zero on the first frame. This indicates that the video capturing started at the middle of a pulse, nevertheless we see that we manage to detect the start, end and peak of the pulse. We also can notice that the force exhibited in this example is order of magnitude smaller then in example 1.
In this example, we see that the force is even smaller than in example 2, this is due to very small movement of the rectangle, i.e. weak pulses by the tissue. This also causes the plot to be noisier compared to previous example. The detection for videos of such ‘weak’ tissues is valuable to the medical researchers since it gives them new data they can work with; such videos are not possible to analyze by-hand like the researchers do currently since the pulse of the tissue is so weak that it is barely noticeable visually when watching the video.
In addition, for each pulse we calculate the decay coefficient, using scipy curve fitting method [8]:

Our final outputs contain all the requested analysis from the Medicine Lab:

- Active force plot.
- Peak to peak time.
- Time to peak-up (Time from baseline to peak).
- Time to peak-down (Time from peak to baseline)
- Contraction time- (Time from start to end of a pulse)
- decay coefficient for each pulse.
- The rectangles tracking video:
Validation
In order to validate our results, we use the following methods:

1. Visually comparing the video and the plots and verifying that the peaks are approximately at the same frames.
2. Comparing the plots of the two rectangles in a video and verifying that the peaks are approximately at the same frames.
3. Comparison between the results achieved by our software and previously computed results by the medical researchers.
4. Verification that the results match the known theory to the medical researchers.
5. The medical researchers have larger organoids, for which they have ways to measure the force of the organoid, therefore it is possible to compare the force value we get to the forces of the larger organoids and check if the orders of magnitude make sense.

Summery
Utilizing the mentioned tools and methodology, we have successfully created an innovative image processing system answering the needs of the laboratory. Our system has demonstrated efficiency by delivering dependable outcomes in a prompt manner. Additionally, it has proven to be user-friendly, offering swift and effortless operation. These attributes have assisted researchers to make progress in their study, ensuring the acquisition of data for further analysis and interpretation.
Bibliography


[4] https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#ga0012a5fdaea70b8a9970165d98722b4c

[5] https://docs.opencv.org/3.4/dc/d6b/group__video__track.html#ga473e4b886d0bcc6b65831eb88ed93323


[9] https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de
