Exploration of the Left Ventricle Centroid Through a Cardiac Cycle

David Faulkenberry, Connor Johnson, Tyler Meathe

Background

With the inception of three-dimensional (3D) echocardiography in 1974, researchers and clinicians have been able to better image and quantify the heart (1). Because 3D echocardiography is able to provide images from any spatial point of view, it has the ability to provide more information about the anatomy and physiology of the heart than that of traditional 2D echocardiography, resulting in a more accurate quantization of chamber volumes (LV & RV) and better visualization of the dynamic motions of the heart, particularly heart valve movement (2). As improvements in 3D ultrasound and image processing continue to take place, it will become easier to recognize and diagnose early symptoms of cardiovascular disease, resulting in better patient outcomes.

Because ultrasound echocardiography can provide a dynamic view of left ventricle (LV) wall thickening and wall motion abnormalities, it is of interest to not only qualitatively analyze these features, but also quantitatively analyze them, so as to reduce subjective bias and standardize readings for research and clinical applications. There are a number of different approaches in quantifying LV function; however, tracking the trajectory of the centroid of the LV has been of recent interest. Because there is no anatomic center of the heart, there are a number of different strategies that have been researched and deployed to determine the centroid.

In one study, a research group tracked the trajectory of the centroid over a cardiac cycle using 2D-Echo in canines. This group experimented with six different methods to determine the centroid. These methods included (i.) averaging the endocardial coordinates, (ii.) averaging the epicardial coordinates, (iii.) finding the center of mass (center of area) subtended by the endocardial border, (iv.) finding the center of mass (center of area) subtended by the epicardial border, (v.) taking the average of the coordinates between the epicardial and endocardial border (mid-myocardium), and (vi.) finding the center of mass subtended by the mid-myocardium. After analyzing each method, the researchers found that a center of mass approach combined with an endocardial tracing for the area yielded the best results. This method was found to be less sensitive to isolated outliers because the area subtended by the boundary was not greatly changed by malposition of a single boundary point. Additionally, the tracing of the endocardium border was found to be more accurate than using epicardium points as the inner wall best defines the blood pool in the ventricle. After tracking the centroid in multiple healthy canines, the researchers found that there was variation of the trajectory in each patient and that there was no consistent path (3). This conclusion confirms the fact that all hearts have slight differences in wall contraction and motion, making it difficult to autonomously detect the centroid and analyze the trajectory of it.

In this study, an algorithmic method requiring minimal user input was designed to output the coordinates of the centroid of the left ventricle throughout a cardiac cycle using 3D

ultrasound patient data. This method utilizes a center of mass approach combined with semi-autonomous short axis endocardial detection to identify the centroid. Successfully locating and tracking the center of mass of the left ventricle will allow physicians and other medical staff to better quantify abnormalities by defining healthy cardiac standards. This information will help isolate regional wall mechanics and contribute to a better understanding of ventricular loading and contraction (4). Features extracted from centroid trajectories show different behaviors in healthy and unhealthy hearts, and these differences can be exploited for diagnostic purposes in classifying unhealthy cardiac functioning.

Methods

Overview

The proposed algorithm comprises seven main steps: (i) resampling the original data files to exhibit an even number of pixels in the x,y, and z direction; (ii) defining a user-input for apex and base selection that determines slice selection; (iii) filtering, edge detection, and cropping techniques for easier downstream analysis and data extraction; (iv) identifying a circular boundary in left ventricle slices and pinpointing the center of these circles to approximate slice area; (v) using a pie slice method to gain a better area approximation; (vi) using area calculations to approximate a final LV volume; and (vii) employing volume approximations to identify the center of mass of the left ventricle. The code is written in the MATLAB programming language.

Resampling

The original dicom (.dcm) data files were loaded into MATLAB using the ReadDicom3D() source code obtained from *Andy's Tutorials* (5). The data was originally stored in 4D 8-bit unsigned integer format and fit into a Cartesian system. However, the data was improperly sampled. In other words, the voxel size in the x,y, and z directions were not equivalent, as the number of pixels in the three directions were 224×208×208. First, we used the *squeeze()* function to extract 3D positional data from the 4D slices. Next, the data was resampled onto a new Cartesian plane with a specified number of samples in each dimension of the new matrix, which was set at 208 pixels. Therefore, the resampled data contained 208 pixels in the x,y, and z directions. As shown in Figure 1, the resampled image on the right is more circular and dimensionally even compared to the ovular cross-section shown in the left side of the figure.

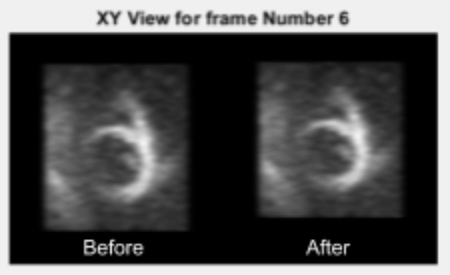

Figure 1: XY slice through LV before and after resampling

Slice Calculation

After resampling of the data set has been completed and all dimensions have an even number of pixels, the algorithm then requires the user to input two points, one point for the apex of the endocardium and another for the base of the ventricle. The user is to input these points on the slice that exactly cuts the ventricle in half in the long-axis orientation. These user inputs define the length of the long-axis in terms of pixel coordinates. Images of the figures that prompt user input can be seen below in Figures 2 and 3.

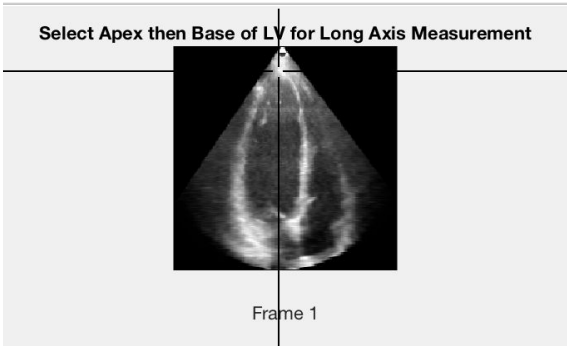

Figure 2: User-Input for Apex of LV

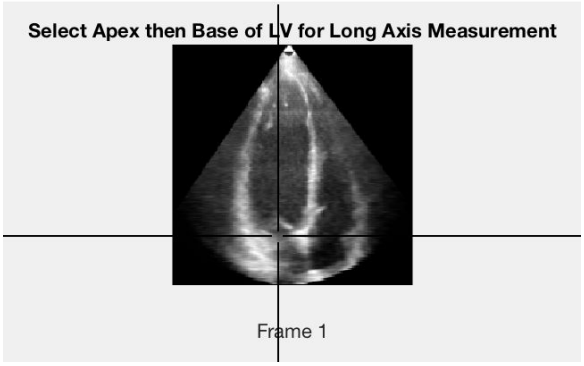

Figure 3: User-Input for Base of LV

After successfully acquiring the coordinates of the apex and base, the algorithm then uses these points to divide the LV into 8 equal sections, starting at the base of the LV and orienting upward. A representation of this division can be seen in Figure 4. The z-coordinate of the user input for the base of the LV is subtracted from the z-coordinate of the apex input and this distance in pixels is converted into a physical measurement in centimeters, using the the conversion factor of the height span divided by the number of pixels. This calculation defines the total long axis length. For calculating the slices that the algorithm will use to cut through the long axis, the long axis pixel inputs are averaged to get a middle slice and subsequently divisions are calculated between the midslice and apex input as well as the midslice and base input. Additional divisions occur between the these new slices and both the top and bottom inputs, as well between

the midslice as seen in Figure 4. The MATLAB code demonstrating these slice calculations can be seen in Appendix B.

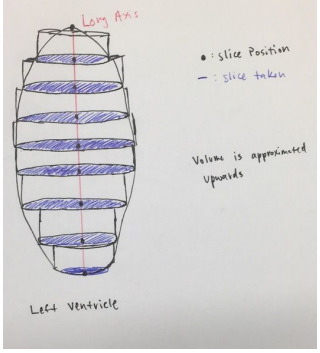

Figure 4: Image of the LV Divided into 8 Equal Slices

After determining the slices that will cut through the ventricle, the algorithm then averages the 7 subsequent slices after the calculated slices to get a thicker slice that more accurately represents the ventricle's anatomy. This thick slice reduces noise, as it is averages some of the noise out of the image and enables better filtering.

Filtering, Edge Detection, and Cropping

As a result of ultrasound images having speckle, it is important to filter these images to remove the degrading noise. This speckle can be seen in Figure 5, with the graininess of the picture representing the speckle. The first filter applied to the data set was the Blackman filter with N=21. This filter blurred some of the high frequency information, resulting in images that better represented the anatomy of the endo- and epicardium borders of the left ventricle.

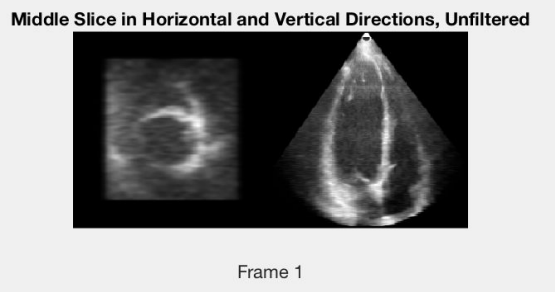

Figure 5: Unfiltered Images of Both Short- and Long-Axis Views

After the Blackman filter was applied to the slices, the slices then underwent threshold filtering, where all brightness values under a certain value were assumed to be 0. This threshold value was determined differently for each slice, with a higher brightness threshold for slices closer to the apex and base of the ventricles compared to a lower brightness threshold for slices closer to the middle of the ventricle. This initial thresholding filter ranged from 0.67 to 0.26 in

terms of brightness. An example of this thresholding can be seen below in Figure 6 with almost all pixels besides the mitral annulus being removed and set to 0, resulting in an image that only contains the desired short-axis view of the ventricle. To improve detection of the mitral annulus further, Otsu thresholding was also applied to the data using a simple MATLAB command that binarizes the image with a global threshold computed using the Otsu method, which chooses a threshold to minimize the intraclass variance of the thresholded black and white pixels. An example of this filter can be seen in Figure 7, with the image becoming completely binary.

Figure 6: Example of Minimum Brightness Thresholding

Figure 7: Example of Otsu Thresholding

After binarizing the image, a high pass filter was then convolved with the data to edge detect the walls of the LV, resulting in an image with both an endocardial and epicardial border. This edge detection was accomplished using the MATLAB command, *edge()* with the Laplacian of Gaussian (LoG) filter. An image detailing this edge detection can be seen below in Figure 8. Once the edges of the ventricle were determined, cropping was then necessary to remove the epicardial border, as it corrupted the image when center of area calculations were performed. The amount of cropping varied from slice to slice, but was generally similar in terms of which pixels should be cropped and set to a 0 brightness. Figure 9 demonstrates cropping for slice 1 of frame 1.

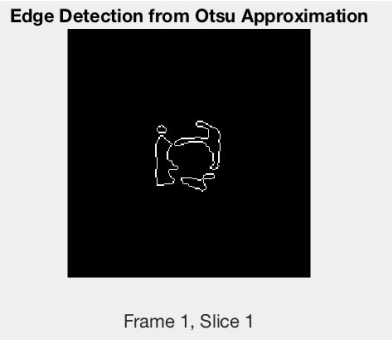

Figure 8: Example Image of Edge Detection

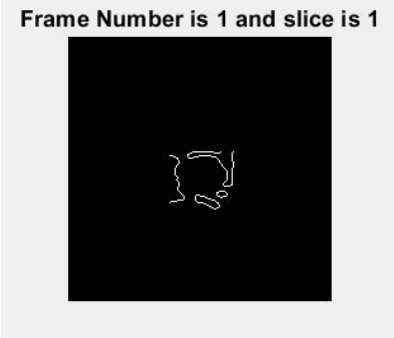

Figure 9: Example of Cropped Image

Area Approximations

Once the edges were detected and the resulting figure was appropriately cropped, a circle was used to roughly approximate the outline of the left ventricle for the slice being analyzed. The MATLAB *imfindcircles()* command yielded potential circles that would fit the input edge image. This function can return any possible number of circles that fit the edge, so the potential radius of the chosen circle was limited to a range which made physical sense. The results of this process can be seen in Figure 10. From this circle approximation, the center point was taken and the "Pie Slice" Method was applied.

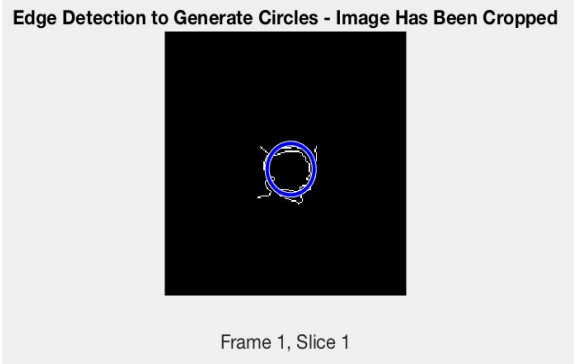

Figure 10: Example of Circle Approximation from Edge Detection

The Pie Slice Method was implemented through the use of the *eightradii* written function. This function takes the center point of the circle found to best fit the edge detection and then generates eight different radii, each 45 degrees apart, and the eight radii are averaged to provide a more robust approximation of the area of each slice. The generation of the radii involves eight different while loops. The loops start at the circle center and then search the rows and the columns of the image, pixel by pixel, until they find a "1" representing an edge point. From Figure 11 it can be seen that these points may be inside of the circle fit, or outside of it. A distance function then returns the radius length from the center to this found point. In the chance that the initial search doesn't find an edge to hit, should it escape through a diagonal line of "1" pixels, or if there isn't an edge that was found by the filtering there, the starting point is adjusted by either one row or one column, depending on which radius is being calculated, and then the search occurs again. This technique worked well initially, but faced challenges at slices closer to the heart's base. Here, the edge of the valve was visible and interfering with edge detection. Two concentric circles would appear from edge detection, one being the inside wall of the left ventricle and the other being the valve outline. This wouldn't cause issues for the circle fitting, given the radius limitations, but it would greatly reduce the area calculated from the Pie Slice Method. To resolve this issue, the starting point for the edge search was moved from the center of the circle to the midpoint between the center and the fit circle's edge. This allowed detection to avoid influence from the valve edge and speed up the processing time by reducing iterations through the loop.

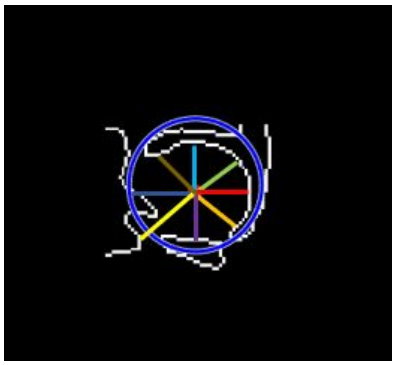

Figure 11: Pie Slice Method Depiction

Volume Estimation

To produce a volume from the eight slice areas generated by the circular and Pie Slice Method, each area was multiplied by the length of the long axis divided by eight, and then these volumes were added together. A depiction of this physically can be seen in Figure 4. The mass of the left ventricle was then determined by multiplying this volume by the density of blood at 1.005 g/L.

Center of Mass Determination

To produce the center of mass of the left ventricle, the center of mass equation in Figure 13 was used in the x, y and z directions. For both x and y, the centers of the fit circles were taken from each slice, and then weighted with the mass of that slice. For the z direction, the height midpoint of each slice was used and again weighted with the calculated mass of each slice. These midpoints were taken with respect to the length of the long axis. The center of mass was taken for each time frame and its movement over the time of the cardiac cycle was analyzed.

$$x_{cm} = \frac{\sum_{i=1}^{N} m_i x_i}{M} \qquad y_{cm} = \frac{\sum_{i=1}^{N} m_i y_i}{M} \qquad z_{cm} = \frac{\sum_{i=1}^{N} m_i z_i}{M}$$

Figure 13: Center of Mass Equations

Results

Data Analysis

Using the algorithm that was developed, data was obtained from ten trials. The data was stored in structural arrays in MATLAB and the measured values from each frame were recorded into an Excel spreadsheet. The specific measurements that were recorded were the total volume estimations using both the circle and pie slice methods, the converted mass with the pie slice method, and the estimated center of mass of the left ventricle in the x,y, and z directions. A statistical analysis was performed on the acquired data. For the volume estimations, the mean and standard deviation for each frame was calculated. The percent error was drawn between the volume estimations calculated using both the cylindrical approximation and pie slide method, and the volume estimations calculated by the class using the 4DViz software. For the x, y, and z center of mass estimations, the mean and standard deviation were calculated for each frame. In addition, statistics were compiled over the entire cardiac cycle, including the overall mean, standard deviation, standard error, range, and confidence level.

Volume Estimation

After the volume estimations for each frame were calculated, the statistics comparing the different techniques to the 4DViz datasets gave insight into the accuracy of our methods. The classroom estimations of the left ventricular volume were the only benchmark values we were able to compare to in order to prove the validity of our algorithm, so they were utilized as accepted values for left ventricular volumes (see Appendix A). As expected, the volume estimations using the cylindrical method were much larger than the volume estimations calculated using the 4DViz software, overestimating each frame of left ventricular volumes by over 35% on average. These results are expected because this edge detection method is set to

find the largest circle in the region and use the area of this circle when estimating the volume of the ventricle. Thus, this method can often produce a larger estimation compared to more specific edge detection methods that seeks to directly trace the edge of the wall. This is the reason why a more specific edge detection method was developed. The volume estimations calculated using the pie slice method throughout the cardiac cycle closely matched the class estimates using the 4DViz software, representing a much more accurate representation of the volume. Figure 14 demonstrates how closely the pie slice method and 4DViz method estimated the volume throughout the entire cardiac cycle. The average percent error of the pie slice method was only 5%.

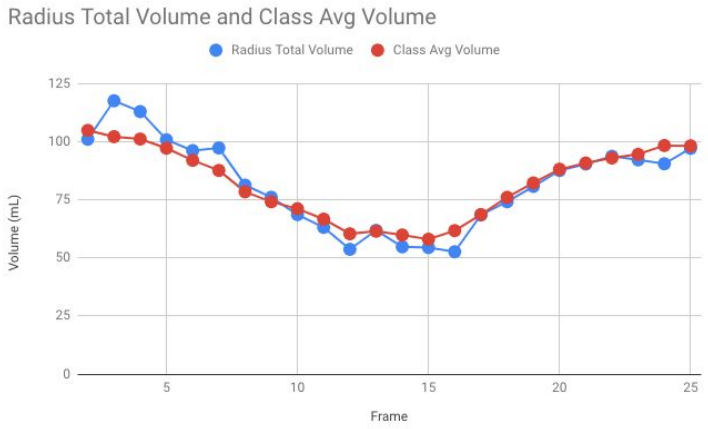

Figure 14 : Volume estimations through cardiac cycle

Center of Mass Approximation

The trajectory of the center of mass through the cardiac cycle generally followed the path that we had expected. We had expected little variance in the x and y-direction with more of a change in the z-direction through the cardiac cycle. Therefore, the center of mass remained fairly consistent through the 25 frames, with a slightly larger variation along the long axis as the heart contracts. The raw average data for each frame is shown in Appendix A. These results are shown in Figures 15, 16, and 17. The statistical results were calculated in pixels, which can be translated into centimeters. As shown in Figure 15, the x-direction of the center of mass stayed relatively stagnant at around the 99 pixel mark, with a standard deviation of only 0.90 pixels and a full range of 4.57 pixels. This is equivalent to a range of 0.42 cm, which is a very minimal change, especially since an outlier was present at a value of 95.8 pixels.

X COM Descriptive Statistics	
Column1	
Mean	99.02184597
Standard Error	0.180208182
Standard Deviation	0.901040912
Sample Variance	0.811874724
Range	4.5661315
Minimum	95.7975048
Maximum	100.3636363
Confidence Level(95.0%)	0.371931408

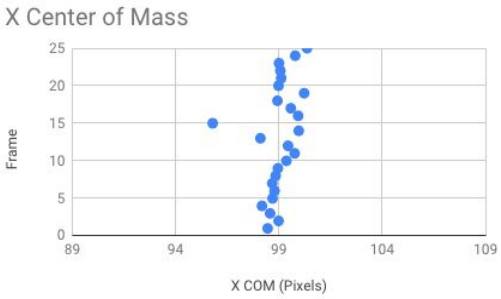

Figure 15: Trajectory of x-coordinate of the center of mass

Similarly, the y-coordinate of the center of mass stayed relatively stationary along the 115-117 pixel range, with a standard deviation of 0.99 pixels. The range of values for the y center of mass was only 3.78 pixels, representing a very small distance of 0.24 cm. These results are shown below in Figure 16.

Y COM Descriptive Statistics	
Column1	
Mean	115.7148
Standard Error	0.19892
Standard Deviation	0.994599
Sample Variance	0.989226
Range	3.777025
Minimum	114.2094
Maximum	117.9864
Confidence Level(95.0%)	0.41055

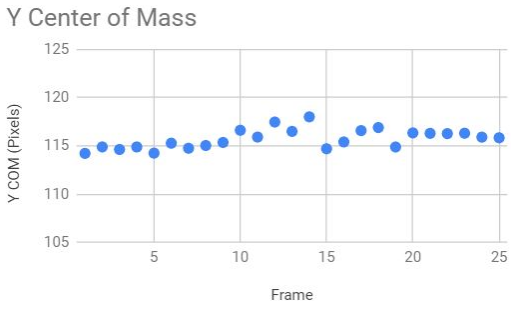

Figure 16: Trajectory of y-coordinate of the center of mass

There was a larger variance in the z, or long axis, component of the center of mass. Initially, we had expected a single peak in this direction, which would correspond to a direct vertical contraction upwards and downwards. As the heart contracts and gets smaller during systole, the base moves upwards, so we had expected that the center of mass would move directly upwards as well. We expected the center of mass to move in the opposite direction when the heart relaxes during diastole and the base moves downwards. However, this was not exactly the case. As shown in Figure 17, the trajectory of the z-component of the center of mass experienced several peaks through the cardiac cycle, undergoing several slightly upward and downward motions. This result is not consistent with a direct single vertical contraction. Rather, the direction of motion was more akin to a diagonal torsional model of the heart as it contracts, which is the actual mechanism by which the heart contracts (6,7). The clockwise and counterclockwise rotation of the LV apex and base result in a more dynamic motion of the center of mass. As seen in the descriptive statistics table in Figure 17, the z-component of the center of mass centered at the 92.76 pixel mark and had a standard deviation of 1.76 pixels. The range was

approximately 8.75 pixels, which is equivalent to 0.73cm. Therefore, the trajectory of the center of mass was slightly upwards and downwards along the long axis, with little motion in the x and y directions.

Z COM Descriptive Statistics	
Column1	
Mean	92.75848
Standard Error	0.351333
Standard Deviation	1.756665
Sample Variance	3.085871
Range	8.750835
Minimum	87.74077
Maximum	96.4916
Confidence Level(95.0%)	0.725116

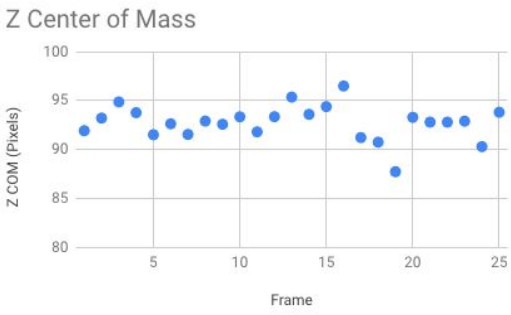

Figure 17: Trajectory of z-coordinate of the center of mass

Limitations/Discussion

When this project was first chosen, the ultimate goal was to automate the calculations entirely, creating a software environment where the user didn't have to input anything. This goal proved ambitious for the four week project and unfortunately we feel somewhat short of it. The user was required to input 2 points per frame, those being the apex of the heart and the base of the heart, as they define the volume of the left ventricle. This amounted to 50 clicks Patient 2, the best data set. While 50 clicks is far superior to attempting to solve this problem by hand, it is an amount of user input that we would like to avoid in the future, and which may make the results less consistent than they could otherwise be. This is due to the fact that the Z direction is heavily influenced by user input. Only with practice did our results start to become more consistent, and they still varied based on each group member's interpretation of the ultrasound image. While it would've been difficult to fully automate, we know now that we could've fixed the apex point to a single user click, and just had to the user click on the changing base in each frame, reducing click count from 50 to 26. Similarly, if we could continue with our software, a goal would've been to somehow detect on the inner edge of the left ventricle, not the outer, so that cropping the image could be reduced or removed entirely. This would make the software more applicable to other datasets. Finally, increasing the number of slices taken would reduce the thickness of each slice and decrease how much of the volume is approximately versus calculated. This would slightly reduce the small error in our results and just make the system better and more robust for detecting abnormalities.

Looking forward, outside of functional changes, one improvement would be to make the code more user friendly. As it sits it can really only be understood by its creators. Function writing was a step in the initial process, but as the deadline approached this was deemed less important than producing and analyzing results. However, if the approximately 1000 line code could be reduced to around 200, then it would be easier for the user to change thresholding,

orientation and cropping values for new patients, without having to ensure they are applied to each and every potential case.

All of these issues and potential adjustments aside, we are happy with the results produced by this software. It calculates left ventricle volume within a 5% accuracy of the 4DViz calculations, which require a much higher degree of user input and manipulation. Similarly, the center of mass results make sense physically. The x and y points are centered while the z point fluctuates with the cardiac cycle.

Conclusion

From the results calculated and observed from the ultrasound 3D data, it is clear that the method employed to estimate the volume of the left ventricle and subsequent center of mass calculations was successful. When comparing the calculated data to the measured data with 4DViz, there was no significant difference, as the calculated values closely matched the measured ones. With the volume data proven to be reliable, it is likely that the center of mass calculations are also accurate. In order to apply this algorithm to future patients though, it is necessary to alter the code so as to make it less reliant on user input for the long axis measurement as well as cropping and threshold filtering. With these possible changes, this algorithm has the potential to be a very powerful diagnostic tool for clinicians and physicians in diagnosing cardiovascular disease and identifying abnormalities in left ventricle wall movement.

References

(1) Maleki M, Esmaeilzadeh M. The evolutionary development of echocardiography. *Iran J Med Sci*. 2012;37(4):222-32.

(2) Shiota T. 3D echocardiography: The present and the future. *Journal of Cardiology*. 2008;52(3):169-185. doi:10.1016/j.jjcc.2008.09.004

(3) Pearlman JD, Hogan RD, Wiske PS, Franklin TD, Weyman AE. Echocardiographic definition of the left ventricular centroid. I. Analysis of methods for centroid calculation from a single tomogram. *Journal of the American College of Cardiology*. 1990;16(4):986-992. doi:10.1016/s0735-1097(10)80352-3

(4) Sehgal, Gaurav, Gabrielle Horne, and Peter Gregson. "Determination of Coordinate System in Short-Axis View of Left Ventricle." *Advances and Innovations in Systems, Computing Sciences and Software Engineering*. Springer, Dordrecht, 2007. 549-554.

(5) Wald, Andrew J. *Andy's Tutorials* http://people.duke.edu/~jcm21/bme265/andy/

(6) Nakatani, Satoshi. "Left ventricular rotation and twist: why should we learn?." *Journal of cardiovascular ultrasound* 19.1 (2011): 1-6.

(7) Sengupta, Partho P., et al. "Twist mechanics of the left ventricle: principles and application." *JACC: Cardiovascular Imaging* 1.3 (2008): 366-376.

Appendix

Appendix A: Data from Trials

Table 1: Calculated Volume Measurements for Patient 2

Frames	Radius Total Volume	Class Avg Volume	% Error
1	100.9526301	104.736	3.612291787
2	117.4750622	102.052	15.11294458
3	112.8363365	101.038	11.67712787
4	100.7075616	97.1	3.715305441
5	96.0714468	91.9	4.539115124
6	97.20906838	87.552	11.03009455
7	81.2262225	78.38	3.631312197
8	76.00013308	74.09	2.578125357
9	68.57358925	71.092	3.542467149
10	63.0771758	66.62	5.31795887
11	53.63731823	60.314	11.06987062
12	61.78242704	61.424	0.5835293065
13	54.7113849	59.814	8.530803998
14	54.41506354	57.98	6.148562371
15	52.58989124	61.666	14.71817333
16	68.41922071	68.58	0.2344404944
17	74.10522341	75.992	2.482862128
18	80.68141327	82.176	1.818763051
19	87.50812611	88.08	0.6492664523
20	90.33355383	90.718	0.4237815736
21	93.63523876	92.9	0.7914303094
22	92.07835253	94.468	2.529584065
23	90.41211268	98.246	7.973746843
24	97.02745044	98.114	1.107435796
25	100.1587161	101.47333	1.295526479
		Average % Error	**5.00458079**

Table 2: Center of Mass Calculations for Patient 2 in Pixels

Frames	X COM	Y COM	Z COM
1	98.45553347	114.2094249	91.92903982
2	98.98399125	114.8783686	93.21550719
3	98.57407344	114.6148522	94.86660431
4	98.17912674	114.8760814	93.7687467
5	98.69303169	114.2362137	91.5175285
6	98.78967727	115.2627392	92.63403888
7	98.67718578	114.7403606	91.54884928
8	98.84229919	115.0244461	92.90826604
9	98.94955871	115.3411608	92.58544409
10	99.36191746	116.6007166	93.34433151
11	99.76104862	115.8968634	91.80715513
12	99.44183138	117.4513095	93.3566173
13	98.10580693	116.4843968	95.35980202
14	99.95975913	117.9864494	93.59753474
15	95.7975048	114.6874184	94.38630762
16	99.93081445	115.3922918	96.49160434
17	99.57551483	116.5657695	91.22665741
18	98.92853444	116.8805006	90.7601913
19	100.2199258	114.8711252	87.74076891
20	98.98122908	116.3286877	93.29156455
21	99.10826578	116.2816841	92.80709703
22	99.07019694	116.2544812	92.80182033
23	99.00491866	116.2929633	92.90181452
24	99.79076719	115.88823	90.30289184
25	100.3636363	115.8231863	93.81172691

Appendix B: MATLAB Code

```
%% BME543 Final Project
% Group 4: Tyler Meathe, David Faulkenberry, Connor Johnson
% Detecting Center of Mass in Left Ventricle
% Assisted by Cooper Moore and Olaf Von Ramm

%% Read in Data
M = readDicom3D('patient2.dcm')
framenumber=M.NumVolumes;

% Extract actual dimensions in cm of the image
w_cm=M.widthspan;
h_cm=M.heightspan;
d_cm=M.depthspan;

% # of samples per dimension in the original data
h=M.height;
w=M.width;
d=M.depth;

% Distance between each sample point in each dimension
h_dis=h_cm/h;
w_dis=w_cm/w;
d_dis=d_cm/d;
density=1.05; % g/mL density of blood

%% Filter that Andy Used
% Filter each frame's volume in the height dimension (1D convolution)
for frame = 1:framenumber
    M.dataHeightFiltered(:,:,:,frame) = FilterHeightwise(M.data(:,:,:,frame));
end

%%  Create 3D matrix with respect to individual time frame
for time=1:framenumber
%Tested using data from the first time frame
%V=squeeze(M.data(:,:,:,time)); %Frame raw data
V=squeeze(M.dataHeightFiltered(:,:,:,time)); %Frame raw data
V=double(V); %Convert it to double

ns= 208; %number of samples in each dimension of the new data matrix

%% Resampling Data into New Matrix
rmax= w;
rpmax=ns;
cmax= h;
cpmax= ns;
dmax= d;
dpmax=ns;
NewHeart= zeros(ns,ns,ns); % Create the new data matrix in 3D


for i=1:ns
    r=((i-1).*(rmax-1)./(rpmax-1))+1;
    rref=floor(r);
    rtop=ceil(r);
```

```
    a= r-rref;
    for j=1:ns
        c=((j-1).*(cmax-1)./(cpmax-1))+1;
        cref=floor(c);
        ctop=ceil(c);
        b= c-cref;
        for k=1:ns
            Bright= V(rref,cref,k).*(1-a).*(1-b)...
                +V(rref,ctop,k).*a.*(1-b)+...
                V(rtop,cref,k).*(1-a).*(b)+...
                V(rtop,ctop,k).*a.*b;
            NewHeart(i,j,k)= Bright;
        end
    end
end

%% Viewing Original slices
Normalize=NewHeart./max(max(NewHeart));
xyview = squeeze(Normalize(:,:,ns./2));

xzview=squeeze(Normalize(:,ns./2,:));
xzview=imrotate(xzview,270);

figure(1);
imshowpair(xyview,xzview,'montage')
title('Middle Slice in Horizontal and Vertical Directions, Unfiltered');
xlabel(['Frame ',num2str(time)])

%% Choose Long Apex and Base Points
figure(2);
imshow(xzview)
title('Select Apex then Base of LV for Long Axis Measurement');
xlabel(['Frame ',num2str(time)])
[xz, yz] = ginput(2);
LongAxisTop = [xz(1), yz(1)];
LongAxisBottom = [xz(2), yz(2)];

longaxisdist=longaxisdistance(LongAxisTop(1),LongAxisTop(2),LongAxisBottom(1),LongAxisBottom(2));

Storage(time).longaxis=longaxisdist;

%% Slice Creation
slice4 = round((yz(2) + yz(1))/2);
slice2 = round((slice4 + yz(1))/2);
slice1 = round((yz(1) + slice2)/2);
slice3 = round((slice2 + slice4)/2);
slice6 = round((yz(2) + slice4)/2);
slice5 = round((slice4 + slice6)/2);
slice8 = round(yz(2));
slice7 = round((slice8 + slice6)/2);

slice=[slice1 slice2 slice3 slice4 slice5 slice6 slice7 slice8];
Storage(time).slice=slice;

%% slice number in the depth plane
for s=1:length(slice)
% Averaging 5 Slices in xy and xz
```

```
thickness = 8;
i = 0;
ThiccsliceXY = 0;
ThiccsliceXZ = 0;

%% Slice 1
if s==1
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.67
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

figure(7)
imshow(ThiccsliceXY)
title('Thick Slice Before Filtering');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 90, 140, 80, 140, ns);
```

```
%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end

%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice1=DegreeRadius;

%% Starting Storage
Storage(time).circlecenter.slice1=newcenters;
viscircles(newcenters, newradii,'Color','b');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice1=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume=CircleArea.*longaxisdist
Storage(time).circlearea.slice1=CircleArea;
Storage(time).circlevol.slice1=CircleVolume;
Storage(time).zcenter.slice1=(yz(1)+slice1)/2;

%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice1=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice1=Volume;
Storage(time).mass.slice1=MassSlice;


end

%% Slice 2
if s==2
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s)+ i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));
```

```
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.4650
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 83, 140, 70, 140, ns);

%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end

%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice2=DegreeRadius;

%% Starting Storage
Storage(time).circlecenter.slice2=newcenters;
viscircles(newcenters, newradii,'Color','b');
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
```

```
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice2=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume=CircleArea.*longaxisdist
Storage(time).circlearea.slice2=CircleArea;
Storage(time).circlevol.slice2=CircleVolume;
Storage(time).zcenter.slice2=(slice2+slice1)/2;

%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice2=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice2=Volume;
Storage(time).mass.slice2=MassSlice;
end

%% Slice 3
if s==3
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.36
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

figure(7)
imshow(ThiccsliceXY)
title('Thick Slice Before Filtering');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
```

```
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 80, 140, 74, 140, ns); %top bottom left right

%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end
%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice3=DegreeRadius;

%% Starting Storage
Storage(time).circlecenter.slice3=newcenters;
viscircles(newcenters, newradii,'Color','b');
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice3=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume=CircleArea.*longaxisdist
Storage(time).circlearea.slice3=CircleArea;
Storage(time).circlevol.slice3=CircleVolume;
Storage(time).zcenter.slice3=(slice2+slice3)/2;
%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice3=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice3=Volume;
```

```
Storage(time).mass.slice3=MassSlice;
end

%% Slice 4
if s==4
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;

% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));

%% Attempting to Threshold Filter in 2D
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.30
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

%% Displaying Thresholded Slices
figure(7)
Filtered2Dxz = imrotate(Filtered2Dxz,270);
imshowpair(Filtered2Dxy,Filtered2Dxz,'montage');

%% Filtering
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'Roberts');
imshowpair(Ixy,edgexy,'montage');

%% Crop Images to remove noise
edgexy = Crop(edgexy, 85, 145, 65, 145, ns);

%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end

%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
```

```
Storage(time).newradii.slice4=DegreeRadius;
%% Starting Storage
Storage(time).circlecenter.slice4=newcenters;
viscircles(newcenters, newradii,'Color','b');
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice4=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume = CircleArea.*longaxisdist
Storage(time).circlearea.slice4=CircleArea;
Storage(time).circlevol.slice4=CircleVolume;
Storage(time).zcenter.slice4=(slice4+slice3)/2;
%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice4=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice4=Volume;
Storage(time).mass.slice4=MassSlice;
end

%% Slice5
if s==5
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.36
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

figure(7)
imshow(ThiccsliceXY)
title('Thick Slice Before Filtering');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])
```

```
%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 80, 160, 65, 140, ns); %top bottom left right

%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end
%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice5=DegreeRadius;
%% Starting Storage
Storage(time).circlecenter.slice5=newcenters;
viscircles(newcenters, newradii,'Color','b');
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice5=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume=CircleArea.*longaxisdist
Storage(time).circlearea.slice5=CircleArea;
Storage(time).circlevol.slice5=CircleVolume;
Storage(time).zcenter.slice5=(slice4+slice5)/2;

%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end
```

```
Storage(time).newarea.slice5=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice5=Volume;
Storage(time).mass.slice5=MassSlice;
end

%% Slice 6
if s==6
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));

%% Attempting to Threshold Filter in 2D
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.26
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 83, 155, 55, 160, ns);

%% Find Circle
```

```
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end

%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice6=DegreeRadius;

%% Starting Storage
Storage(time).circlecenter.slice6=newcenters;
viscircles(newcenters, newradii,'Color','b');
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice6=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume = CircleArea.*longaxisdist
Storage(time).circlearea.slice6=CircleArea;
Storage(time).circlevol.slice6=CircleVolume;
Storage(time).zcenter.slice6=(slice6+slice5)/2;
%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice6=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice6=Volume;
Storage(time).mass.slice6=MassSlice;
end

%% Slice 7
if s==7
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
```

```
    for y = 1:ns
        %filter xy
        if ThiccsliceXY(x, y) <= 0.39
            Filtered2Dxy(x, y) = 0;
        else
            Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
        end
    end
end

figure(7)
imshow(ThiccsliceXY)
title('Thick Slice Before Filtering');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 90, 150, 60, 140, ns);

%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
    newradii(i)=radii(indradii(i));
    for j=1:2
        newcenters(i,j)= centers(indradii(i),j);
    end
end

%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice7=DegreeRadius;

%% Starting Storage
Storage(time).circlecenter.slice7=newcenters;
viscircles(newcenters, newradii,'Color','b');
```

```
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice7=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume=CircleArea.*longaxisdist
Storage(time).circlearea.slice7=CircleArea;
Storage(time).circlevol.slice7=CircleVolume;
Storage(time).zcenter.slice7=(slice6+slice7)/2;
%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice7=Area;
Volume = Area.*longaxisdist./count;
MassSlice=density.*Volume;
Storage(time).newvolume.slice7=Volume;
Storage(time).mass.slice7=MassSlice;
end

%% Slice 8
if s==8
for i=0:thickness
   NewSliceXY = NewHeart(:,:,slice(s) + i);
   ThiccsliceXY = ThiccsliceXY + NewSliceXY;
end
ThiccsliceXY = ThiccsliceXY./ thickness;


% normalizing averaged slices
ThiccsliceXY=ThiccsliceXY./max(max(ThiccsliceXY));
Filtered2Dxy = zeros(ns);
Filtered2Dxz = zeros(ns);

for x = 1:ns
   for y = 1:ns
      %filter xy
      if ThiccsliceXY(x, y) <= 0.45
         Filtered2Dxy(x, y) = 0;
      else
         Filtered2Dxy(x, y) = ThiccsliceXY(x, y);
      end
   end
end

figure(7)
imshow(ThiccsliceXY)
title('Thick Slice Before Filtering');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Displaying Thresholded Slices
figure(3)
imshow(Filtered2Dxy);
title('Slice Filtered by Thresholding (Removes Minimum Brightness)');
```

```
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

%% Filtering
figure(4);
Ixy=imbinarize(Filtered2Dxy,'global');
edgexy=edge(Ixy,'log');
imshow(Ixy);
title('Filtering Threshold Slice with Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])

figure(5)
imshow(edgexy)
title('Edge Detection from Otsu Approximation');
xlabel(['Frame ',num2str(time),', Slice ', num2str(s)])


%% Crop Images to Remove Noise
edgexy = Crop(edgexy, 96 , 140, 74, 140, ns);

%% Find Circle
figure(6);
imshow(edgexy)
title('Edge Detection to Generate Circles - Image Has Been Cropped');
[centers,radii] = imfindcircles(edgexy,[8 250]) ;
indradii=find(max(radii)==radii);
for i=1:length(indradii)
   newradii(i)=radii(indradii(i));
   for j=1:2
      newcenters(i,j)= centers(indradii(i),j);
   end
end

%% Area of Circle with 8 Pie Slices (Every 45 Degrees)
DegreeRadius = eightradii(zeros(1, 8), newcenters, edgexy, ns, newradii);
Storage(time).newradii.slice8=DegreeRadius;

%% Starting Storage
Storage(time).circlecenter.slice8=newcenters;
viscircles(newcenters, newradii,'Color','b');
title(['Frame Number is ',num2str(time),' and slice is ', num2str(s)])
newradii=newradii.*(w_cm./ns);
Storage(time).circleradii.slice8=newradii;
CircleArea=pi.*(newradii).^2;
CircleVolume=CircleArea.*longaxisdist
Storage(time).circlearea.slice8=CircleArea;
Storage(time).circlevol.slice8=CircleVolume;
Storage(time).zcenter.slice8=(slice8+slice7)/2;
%% Area of a Slice
Area = 0;
nonzeros = find(DegreeRadius ~= 0);
count = length(nonzeros);
for i=1:count
   Area = Area + (1/count).*pi.*(DegreeRadius(i).^2);
end

Storage(time).newarea.slice8=Area;
Volume = Area.*longaxisdist./count;
```

```
MassSlice=density.*Volume;
Storage(time).newvolume.slice8=Volume;
Storage(time).mass.slice8=MassSlice;
end

%% Pause to view figures before going to next time frame
pause
close all
clear centers radii newradii newcenters indradii edgexy X Y X0 Y0 Point1 Point2 Point3 Point4 Point5 Point6 Point7 Point8
DegreeRadius
end
%% Extract Data
% Center Coordinates
% x and y coordinate averaging
Storage(time).avgxy=1/8.*(Storage(time).circlecenter.slice1 + ...
    Storage(time).circlecenter.slice2 + Storage(time).circlecenter.slice3+ ...
    Storage(time).circlecenter.slice4 + Storage(time).circlecenter.slice5+ ...
    Storage(time).circlecenter.slice6 + Storage(time).circlecenter.slice7+ ...
    Storage(time).circlecenter.slice8);

% Circle Volume
Storage(time).avgcirclevolume=1/8.*(Storage(time).circlevol.slice1 + ...
    Storage(time).circlevol.slice2 + Storage(time).circlevol.slice3 + ...
    Storage(time).circlevol.slice4 + Storage(time).circlevol.slice5 + ...
    Storage(time).circlevol.slice6 + Storage(time).circlevol.slice7 + ...
    Storage(time).circlevol.slice8);

% New Volume
Storage(time).avgvolume=(Storage(time).newvolume.slice1 + ...
    Storage(time).newvolume.slice2 + Storage(time).newvolume.slice3 + ...
    Storage(time).newvolume.slice4 + Storage(time).newvolume.slice5 + ...
    Storage(time).newvolume.slice6 + Storage(time).newvolume.slice7 + ...
    Storage(time).newvolume.slice8);
% Total Mass
Storage(time).totalmass=Storage(time).avgvolume.*density;

% Center of Mass Equation for x
Storage(time).xcom=(1/Storage(time).totalmass).*(Storage(time).circlecenter.slice1(1).*Storage(time).mass.slice1 + ...
    Storage(time).circlecenter.slice2(1).*Storage(time).mass.slice2 + ...
    Storage(time).circlecenter.slice3(1).*Storage(time).mass.slice3 + ...
    Storage(time).circlecenter.slice4(1).*Storage(time).mass.slice4 + ...
    Storage(time).circlecenter.slice5(1).*Storage(time).mass.slice5 + ...
    Storage(time).circlecenter.slice6(1).*Storage(time).mass.slice6 + ...
    Storage(time).circlecenter.slice7(1).*Storage(time).mass.slice7 + ...
    Storage(time).circlecenter.slice8(1).*Storage(time).mass.slice8);
% Center of Mass Equation for y
Storage(time).ycom=(1/Storage(time).totalmass).*(Storage(time).circlecenter.slice1(2).*Storage(time).mass.slice1 + ...
    Storage(time).circlecenter.slice2(2).*Storage(time).mass.slice2 + ...
    Storage(time).circlecenter.slice3(2).*Storage(time).mass.slice3 + ...
    Storage(time).circlecenter.slice4(2).*Storage(time).mass.slice4 + ...
    Storage(time).circlecenter.slice5(2).*Storage(time).mass.slice5 + ...
    Storage(time).circlecenter.slice6(2).*Storage(time).mass.slice6 + ...
    Storage(time).circlecenter.slice7(2).*Storage(time).mass.slice7 + ...
    Storage(time).circlecenter.slice8(2).*Storage(time).mass.slice8);
% Center of Mass Equation for z
Storage(time).zcom=(1/Storage(time).totalmass).*(Storage(time).zcenter.slice1.*Storage(time).mass.slice1 + ...
    Storage(time).zcenter.slice2.*Storage(time).mass.slice2 + ...
```

```
    Storage(time).zcenter.slice3.*Storage(time).mass.slice3 + ...
    Storage(time).zcenter.slice4.*Storage(time).mass.slice4 + ...
    Storage(time).zcenter.slice5.*Storage(time).mass.slice5 + ...
    Storage(time).zcenter.slice6.*Storage(time).mass.slice6 + ...
    Storage(time).zcenter.slice7.*Storage(time).mass.slice7 + ...
    Storage(time).zcenter.slice8.*Storage(time).mass.slice8);

end
```

MATLAB Functions

eightradii.m

```
function DegreeRadius = eightradii(DegreeRadius, newcenters, edgexy, ns, newradii)
%% Radius 1
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
nx = 1;
while edgexy(Y,X) == 0
    if edgexy(Y, X) == 1
        Point1 = [X, Y];
        DegreeRadius(1) = radiusdegree(Point1(1), Point1(2), X0, Y0);
    else
        Y = Y - 1;
        if Y < 1
            X = X0 + 1.*nx;
            Y = Y0;
            nx = nx + 1;
        end
        if edgexy(Y, X) == 1
            Point1 = [X, Y];
            DegreeRadius(1) = radiusdegree(Point1(1), Point1(2), X0, Y0);
        end

    end
end

%% Radius 2
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
nx = 1;
ny = 1;
while edgexy(Y,X)== 0
    if edgexy(Y, X) == 1
        Point2 = [X, Y];
        DegreeRadius(2) = radiusdegree(Point2(1), Point2(2), X0, Y0);
    else
        Y = Y - 1;
        if Y < 1
            Y = Y0 - 1.*ny;
            X = X0;
            ny = ny + 1;
        end
        if edgexy(Y,X) == 1
            Point2 = [X, Y];
            DegreeRadius(2) = radiusdegree(Point2(1), Point2(2), X0, Y0);
        else
            X = X + 1;
        end
```

```
        if X > ns
            X = X0 + 1.*nx;
            Y = Y0;
            nx = nx + 1;
        end
    end
end

%% Radius 3
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
ny = 1;
while edgexy(Y,X) == 0
    if edgexy(Y, X) == 1
        Point3 = [X, Y];
        DegreeRadius(3) = radiusdegree(Point3(1), Point3(2), X0, Y0);
    else
        X = X + 1;
        if X > ns
            Y = Y0 + 1.*ny;
            X = X0;
            ny = ny + 1;
        end
        if edgexy(Y, X) == 1
            Point3 = [X, Y];
            DegreeRadius(3) = radiusdegree(Point3(1), Point3(2), X0, Y0);
        end
    end
end
%% Radius 4
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
nx = 1;
ny = 1;
while edgexy(Y,X) == 0
    if edgexy(Y, X) == 1
        Point4 = [X, Y];
        DegreeRadius(4) = radiusdegree(Point4(1), Point4(2), X0, Y0);
    else
        Y = Y + 1;
        if Y > 208
            Y = Y0 + 1.*ny;
            X = X0;
            ny = ny + 1;
        end
        if edgexy(Y,X) == 1
            Point4 = [X, Y];
            DegreeRadius(4) = radiusdegree(Point4(1), Point4(2), X0, Y0);
        else
            X = X + 1;
        end
        if X > 208
            X = X0 + 1.*nx;
            Y = Y0;
            nx = nx + 1;
        end
    end
end

%% Radius 5
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
```

```
X = round(newcenters(1));
Y = round(newcenters(2));
nx = 1;
while edgexy(Y,X) == 0
    if edgexy(Y, X) == 1
        Point5 = [X, Y];
        DegreeRadius(5) = radiusdegree(Point5(1), Point5(2), X0, Y0);
    else
        Y = Y + 1;
        if Y > 208
            X = X0 + 1.*nx;
            Y = Y0;
            nx = nx + 1;
        end
        if edgexy(Y, X) == 1
            Point5 = [X, Y];
            DegreeRadius(5) = radiusdegree(Point5(1), Point5(2), X0, Y0);
        end
    end
end
%% Radius 6
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
nx = 1;
ny = 1;
while edgexy(Y,X)== 0
    if edgexy(Y, X) == 1
        Point6 = [X, Y];
        DegreeRadius(6) = radiusdegree(Point6(1), Point6(2), X0, Y0);
    else
        Y = Y + 1;
        if Y > 208
            Y = Y0 + 1.*ny;
            X = X0;
            ny = ny + 1;
        end
        if edgexy(Y,X) == 1
            Point6 = [X, Y];
            DegreeRadius(6) = radiusdegree(Point6(1), Point6(2), X0, Y0);
        else
            X = X - 1;
        end
        if X < 1
            X = X0 - 1.*nx;
            Y = Y0;
            nx = nx + 1;
        end

    end
end

%% Radius 7
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
ny = 1;
while edgexy(Y,X) == 0
    if edgexy(Y, X) == 1
        Point7 = [X, Y];
        DegreeRadius(7) = radiusdegree(Point7(1), Point7(2), X0, Y0);
    else
        X = X - 1;
        if X < 1
```

```
            Y = Y0 - 1.*ny;
            X = X0;
            ny = ny + 1;
        if edgexy(Y, X) == 1
            Point7 = [X, Y];
            DegreeRadius(7) = radiusdegree(Point7(1), Point7(2), X0, Y0);
        end
        end
    end
end
%% Radius 8
X0 = round(newcenters(1) + (1/2).*newradii);
Y0 = round(newcenters(2) + (1/2).*newradii);
X = round(newcenters(1));
Y = round(newcenters(2));
nx = 1;
ny = 1;
while edgexy(Y,X)== 0
    if edgexy(Y, X) == 1
        Point8 = [X, Y];
        DegreeRadius(8) = radiusdegree(Point8(1), Point8(2), X0, Y0);
    else
        Y = Y - 1;
        if Y < 1
            Y = Y0 - 1.*ny;
            X = X0;
            ny = ny + 1;
        end
        if edgexy(Y,X) == 1
            Point8 = [X, Y];
            DegreeRadius(8) = radiusdegree(Point8(1), Point8(2), X0, Y0);
        else
            X = X - 1;
        end
        if X < 1
            X = X0 - 1.*nx;
            Y = Y0;
            nx = nx + 1;
        end
    end
end
end
```

MovingCOM.m

```
%% BME543 Final Project
% Group 4: Tyler Meathe, David Faulkenberry, Connor Johnson
% Detecting Center of Mass in Left Ventricle
% Assisted by Cooper Moore and Olaf Von Ramm

%% Read in Data
M = readDicom3D('patient2.dcm')
framenumber=M.NumVolumes;

% Extract actual dimensions in cm of the image
w_cm=M.widthspan;
h_cm=M.heightspan;
d_cm=M.depthspan;

% # of samples per dimension in the original data
h=M.height;
w=M.width;
d=M.depth;

% Distance between each sample point in each dimension
h_dis=h_cm/h;
w_dis=w_cm/w;
```

```
d_dis=d_cm/d;
density=1.05; % g/mL density of blood

%% Filter that Andy Used
% Filter each frame's volume in the height dimension (1D convolution)
for frame = 1:framenumber
    M.dataHeightFiltered(:,:,:,frame) = FilterHeightwise(M.data(:,:,:,frame));
end

%%  Create 3D matrix with respect to individual time frame
for time=1:framenumber
%Tested using data from the first time frame
%V=squeeze(M.data(:,:,:,time)); %Frame raw data
V=squeeze(M.dataHeightFiltered(:,:,:,time)); %Frame raw data
V=double(V); %Convert it to double

ns= 208; %number of samples in each dimension of the new cube data matrix

%% Resampling Data into New Matrix
rmax= w;
rpmax=ns;
cmax= h;
cpmax= ns;
dmax= d;
dpmax=ns;
NewHeart= zeros(ns,ns,ns); % Create the new cube data matrix in 3D


for i=1:ns
    r=((i-1).*(rmax-1)./(rpmax-1))+1;
    rref=floor(r);
    rtop=ceil(r);
    a= r-rref;
    for j=1:ns
        c=((j-1).*(cmax-1)./(cpmax-1))+1;
        cref=floor(c);
        ctop=ceil(c);
        b= c-cref;
        for k=1:ns
            Bright= V(rref,cref,k).*(1-a).*(1-b)...
                +V(rref,ctop,k).*a.*(1-b)+...
                V(rtop,cref,k).*(1-a).*(b)+...
                V(rtop,ctop,k).*a.*b;
            NewHeart(i,j,k)= Bright;
        end
    end
end

%% Viewing Original slices
Normalize=NewHeart./max(max(NewHeart));
xyview = squeeze(Normalize(:,:,ns./2));

xzview=squeeze(Normalize(:,ns./2,:));
xzview=imrotate(xzview,270);


%% Moving COM Plot
%AllPoints = zeros(1, 25);
AllPoints(time) = time;
Points = round(AllPoints);


figure(1)
X = [98.45553347
98.98399125
98.57407344
```

98.17912674
98.69303169
98.78967727
98.67718578
98.84229919
98.94955871
99.36191746
99.76104862
99.44183138
98.10580693
99.95975913
95.7975048
99.93081445
99.57551483
98.92853444
100.2199258
98.98122908
99.10826578
99.07019694
99.00491866
99.79076719
100.3636363];% Average X COM for each Frame

Y = [114.2094249
114.8783686
114.6148522
114.8760814
114.2362137
115.2627392
114.7403606
115.0244461
115.3411608
116.6007166
115.8968634
117.4513095
116.4843968
117.9864494
114.6874184
115.3922918
116.5657695
116.8805006
114.8711252
116.3286877
116.2816841
116.2544812
116.2929633
115.88823
115.8231863]; % Average Y COM for each Frame

Z = [91.92903982
93.21550719
94.86660431
93.7687467
91.5175285
92.63403888
91.54884928
92.90826604
92.58544409
93.34433151
91.80715513
93.3566173
95.35980202
93.59753474
94.38630762
96.49160434
91.22665741

```
90.7601913
87.74076891
93.29156455
92.80709703
92.80182033
92.90181452
90.30289184
93.81172691]; % Average Z COM for each Frame

    subplot(1, 2, 1)
    imshow(xyview);
    hold on
    plot(X(time), Y(time), 'r.');
    xlabel('XY View');

    subplot(1, 2, 2)
    imshow(xzview);
    hold on;
    plot(X(time), Z(time), 'r.');
    xlabel('XZ View');
    suptitle('Center of Mass Movement Over Time');
    pause(0.01)

    % Input time to get singular point on each frame, input Points to get
    % compounding image of points over time
end
```

Longaxisdistance.m

```
function distance=longaxisdistance(a,b,c,d)
distance=sqrt(((.08363464.*(b-d)).^2)+ (.0907966.*(a-c)).^2)
end
```

Crop.m

```
function edgexy = Crop(edgexy, a, b, c, d, ns)
%% Crop Images to remove noise
for x=1:a
    for y=1:ns
        edgexy(x,y)=0;
    end
end
for x=b:ns
    for y=1:ns
        edgexy(x,y)=0;
    end
end
for y=1:c
    for x=1:ns
        edgexy(x,y)=0;
    end
end
for y=d:ns
    for x=1:ns
        edgexy(x,y)=0;
    end
end
end
```

FilterHeightWise.m

```
function vol_filtered = FilterHeightwise(vol)

% Blackman window size N is the filter
 N = 21;
 % 1D filter as a 3D point spread function
 filter = zeros(1,N,1);
 filter(1,:,1) = blackman(N);
```

```
 % PSF convolution in 3D with the 1D filter (nothing will happen in the
 % [width,depth] plane as the convolution is along every height vector as
 % if its own 1D signal), and then cut off the extra N/2 added on each
 % side of the height dimension to maintain unchanged height dimension
 % size (this should have a negligible effect on our analysis).
 vol_filtered = convn(vol,filter,'same');
end
```

Radiusdegree.m

```
function radius = radiusdegree(a,b,c,d)
radius=sqrt(((0.0907966.*(a-c)).^2) + ((0.063179.*(b-d)).^2));
end
```