

## Introduction

Physiological hand tremor has been measured to be 38  $\mu\text{m}$  in retinal microsurgery [1]. The frequency of hand tremor can vary between 4-9 Hz [2]. Hand tremor poses a risk to the patient in microsurgery because the surgeon must navigate through and operate on sub-millimeter structures. Surgical robots help remove hand tremor. At Johns Hopkins, the Galen Steady-Hand Robot has been developed for microsurgery in the head and neck with sub-millimeter precision [3]. Steady-Hand Robots (SHR) are cooperative systems where the robot and the surgeon both hold the tool. The surgeon pushes on the tool, the robot reads the force applied and moves in the desired direction.

To compute the hand tremor from a surgical video the tool must be tracked. The 3D trajectory of the tool can be transformed in the frequency domain and the motion above the tremor threshold isolated. There are many publications on tracking tools with and without markers. Bouarfa et al [4] developed method for tool tracking with color markers. The authors paint the shafts of laparoscopic tools different colors and compute color histograms for each marker color. These color histograms are used to find the probability each pixel in an image is a marker. The CAMshift algorithm is used to find the most likely marker positions in the image.

## Problem

We want to compare the magnitude and frequency of tremor in manual and robot-assisted surgery. In many microsurgical applications, surgical site is observed through a stereo microscope which is not rigid and can move during procedures. We propose an algorithm that can accurately track color markers on surgical tools and in the background, compensate for microscope motion, and perform frequency analysis on the tool trajectories. This algorithm is used to compare suturing task performed manually and with the Galen robot.

## Methods

Our algorithm performs frequency analysis on tool motion observed through a calibrated stereomicroscope. The microscope was calibrated using OpenCV functions.

Algorithm Pipeline:



### Color Marker Tracking

We need to track the tool motion and microscope motion. In our experimental setup the background is stationary; therefore, apparent background movement in the microscope video corresponds to the microscope movement.

Blue color markers were painted on the background, and green color markers were painted on the surgical tools. The surgical tools used have multiple green markers so they can be tracked from many different angles. The color markers were manually segmented in several images (between 1 and 7) from the experimental videos. Color histograms were created using the Hue and Saturation values from the segmented marker pixels. A Gaussian mixture model (GMM) was used to compute the probability a pixel with a given hue and saturation was from a color marker. Two separate GMMs were created for detecting background and tool markers. The number of modes of the GMM (3 or 4) and the channel with marker detections were tuned manually for each experimental video.

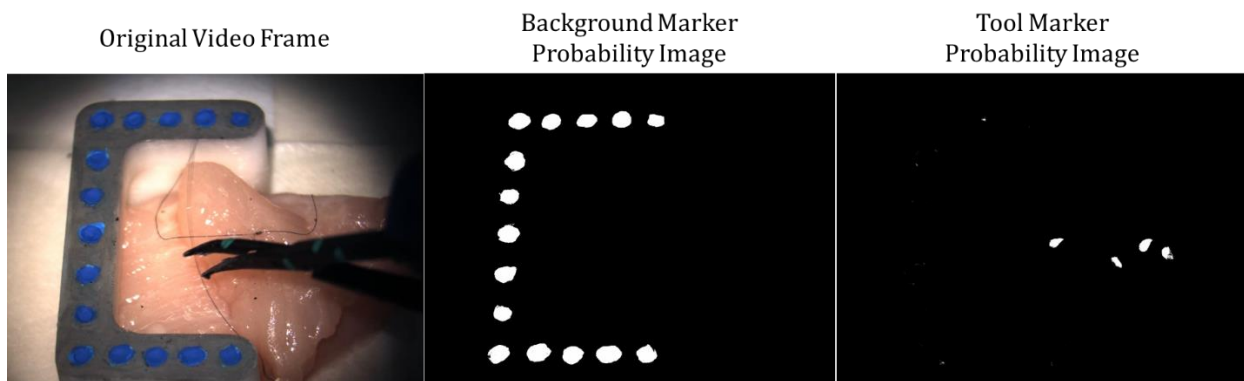


Fig 1: frame from microscope video, background marker probability image, tool marker probability image

For each frame in the microscope video a background marker probability image and a tool probability image were created. The tool markers were detected using an MSER blob detector on the probability images. We matched the tool marker points detected in the left image to nearby markers detected in the right image. If there was no nearby marker point detected in the opposite image the detected marker was ignored.

The background markers were detected by matching a background marker template to the current frame. A left and right template were used to match to the left and right videos. The templates were cropped marker probability images with all 15 markers visible. The centers of the markers in the template image were detected using an MSER blob detector. A simple greedy search was used to match the blobs in the left template and the right template.

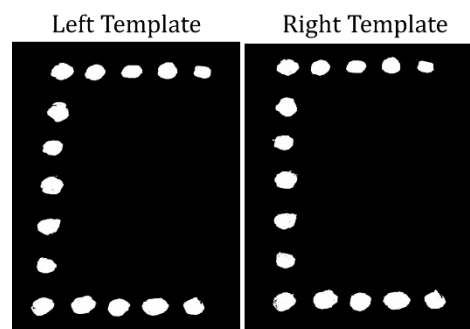


Fig 2: left background template, right background template

For each video frame, the algorithm found the best translation and rotation the templates to match the background probability image. The marker points detected in the images were

rotated and translated by the best transform found. These points were taken as the background marker image detection points. This allowed positions for all 15 background markers to be found in every video frame, even if some markers were occluded. Once the tool and background markers were detected and matched in the left and right images, the 3D marker locations were triangulated.

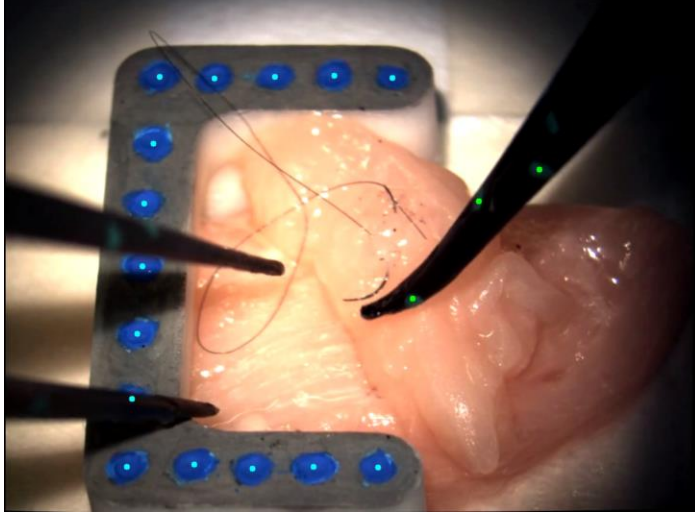


Fig. 3: Video frame with marker and tool detections

### Camera Motion Compensation

The background in the experimental video was stationary. Any apparent background motion in the video was microscope motion. The camera motion compensation module found the rigid transform from the triangulated 3D background points in each frame to the background points in the first frame. This transform was applied to the detected tool points in each frame to remove the effects of camera motion. The Procrustes algorithm [5] was used to find initial guesses for the transforms  $T_{initial}$ . Bundle adjustment was applied on a shifting window of 80 frames to find the optimal transform from each frame to the first frame.

N: number of frames in the video

$$T_{initial} = [T_{00}, T_{10}, \dots, T_{(N-1)0}]$$

The homogeneous transform  $T_{i0}$  aligns the background marker points in frame  $i$  to the points in frame 0.  $T_{i0}$  can be expressed as 3 Rodrigues angles and a translation.

$$T_{i0} = \begin{bmatrix} x \\ R \begin{matrix} y \\ z \end{matrix} \\ 0 \quad 1 \end{bmatrix}$$

$$rod_x, rod_y, rod_z = Rodrigues(R)$$

The vector  $P_{i0}$  is used to parameterize  $T_{i0}$ .

$$P_{i0} = 1 \times 6 \text{ vector}$$

$$P_{i0} = [rod_x, rod_y, rod_z, x, y, z]$$

$P_{initial}$  is a vector with the transformations for all N video frames.

$$P_{initial}: 1 \times 6N \text{ vector}$$

$$P_{initial} = [P_{00}, P_{10}, \dots, P_{(N-1)0}]$$

$points_i$  : 3D triangulated points of background markers in frame i

The goal is to minimize the error:  $\min(T_{i0} * points_i - points_0)$

sumError is the sum of Euclidean distances between the rotated points in frame i and points in frame 0:

$$sumError_i = \sum_{j=1}^{15} dist(points_0[j], T_{i0} * points_i[j])$$

The translational error in frame i:

$$t_i = mean(T_{i0} * points_i) - mean(points_0)$$

The rotational error in frame i:

- Find rotation between the  $T_{i0} * points_i$  and  $points_0$  using Procrustes algorithm [5]

$$\Delta R_i: \text{rotation between } T_{i0} * points_i \text{ and } points_0$$

The Rodrigues angles of  $\Delta R$  are the rotational errors of  $T_{i0}$

$$\Delta R_{xi}, \Delta R_{yi}, \Delta R_{zi} = Rodrigues(\Delta R_i)$$

Return the error of each parameter scaled by the overall distance between the points in the point cloud:

$$Error_i = [sumError_i * \Delta R_{xi}, sumError_i * \Delta R_{yi}, sumError_i * \Delta R_{zi}, sumError_i * t_{xi}, sumError_i * t_{yi}, sumError_i * t_{zi}]$$

$$Error: 1 \times 6N \text{ vector}$$

$$Error = [Error_0, Error_1, \dots, Error_{N-1}]$$

Bundle adjustment finds the vector P that minimizes the vector Error.

### Frequency Analysis

The DFT of the stabilized tool trajectory was used to analyze the hand tremor. Three 1-dimensional DFTs were taken of the x-coordinates, y-coordinates, and z-coordinates of the trajectory. The 3 DFTs were summed to get the maximum possible movement at each frequency. Physiological hand tremor is between 4-9Hz [2]. The microscope video is

recorded at 30 fps so we can recover motion frequencies up to 15 Hz. A lowpass filter was applied to the frequency results to isolate the tremulous motion above 5 Hz.

## Experiments

To test the marker detection and frequency analysis, stereo microscope video was recorded of a novice suturing a chicken breast manually and with the Galen robot. To get stable background fiducials, a chicken holder was 3D printed and painted (see figure 4).

To test the camera motion compensation, a stereo microscope video was recorded of the background marker moving along a grid with known side lengths. The movement between the stabilized 3D points and the original triangulated 3D points was compared to the known movement of the background marker.

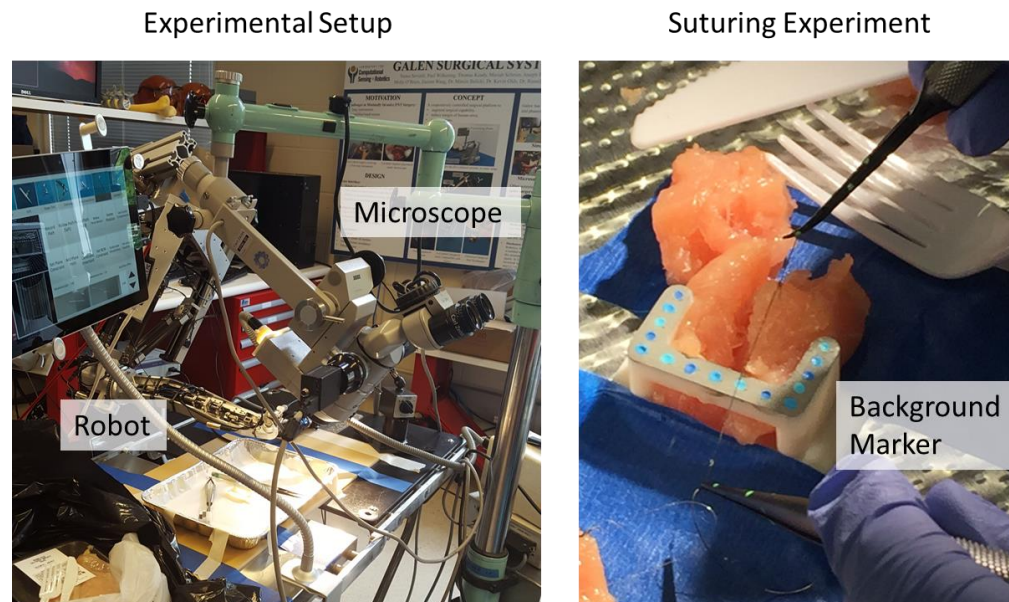


Fig. 4: Lab setup with robot and microscope. Background marker being used in suturing experiment.

## Results

### Triangulation Accuracy

Fig 5 shows the 3D background marker positions triangulated from a frame in the suturing\_manual\_05-04-17-1 video sequence. The ground truth distances between the markers were measured using digital calipers.

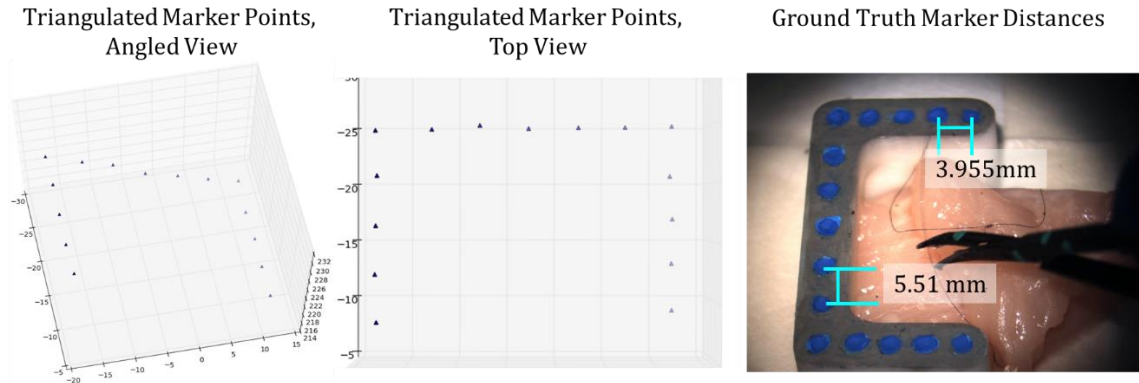


Fig 5: Background markers triangulated from suturing\_manual\_05-04-17-1 clip

The distance between all adjacent triangulated markers was computed (see figure 5 left and center). The ground truth distance between markers along the 2 shorter ends of the chicken holder is 3.955 mm. The ground truth distance between markers along the longer end of the chicken holder is 5.51 mm. The average distance in the triangulated chicken holder is  $4.02 \pm 0.22$  mm along the short ends and  $5.39 \pm 0.24$  mm along the longer ends. The triangulation error was  $< 0.15$  mm. This shows that the 3D triangulated points are accurate.

Measured Marker Distances:

	Triangulated Distance	True Distance	Error
<b>Short Side</b>	$4.02 \pm 0.22$ mm	3.955 mm	0.065 mm
<b>Long Side</b>	$5.39 \pm 0.24$ mm	5.51 mm	0.13 mm

### Camera Motion Compensation

To test the camera motion compensation, the background marker was moved a known distance in a stereo microscope video. The triangulated background points were stabilized. The translation between the stabilized points and the original points was compared to the known distance the marker was moved. The tracked translation is the average distance between each stabilized and original point on the background marker. The tracked movement is the norm of the tracked translation.

True Movement	Tracked Movement	Error
6.3 mm	6.804 mm	0.504 mm
12.6 mm	13.03 mm	0.430 mm
18.9 mm	20.33 mm	1.43 mm
25.2 mm	25.8247 mm	0.625 mm

### Frequency Analysis

The tool markers were detected in each video frame. Markers near each other were grouped frame to frame to create tool trajectories. There were multiple green dots on both surgical tools. This led to multiple markers being detected in each frame and markers from both tools being detected together. This is problematic because the final trajectory could have the tool movement from one time counted in multiple trajectories and in the robot sequences there are a short trajectories detected from the manual tool in the left hand. An intermediate solution to this was to only analyze the longest trajectory in the video. This trajectory would not have multiple detections from the same frame or from different tools. As part of my future work I plan to implement a tool appearance model that can find the tool position and orientation given a subset of markers. The appearance model will let us detect the tool position and orientation consistently in every frame.

In the experimental video from 05-04 the background marker was not fixed rigidly to the table. The background points were not truly stable so I did not apply the motion compensation to the detected tool points. In this video, there was minimal camera motion so the frequency analysis is still meaningful. A lab mate, Radhika, designed a background marker tray that will rigidly hold the background marker in future experiments.

Without microscope motion compensation:

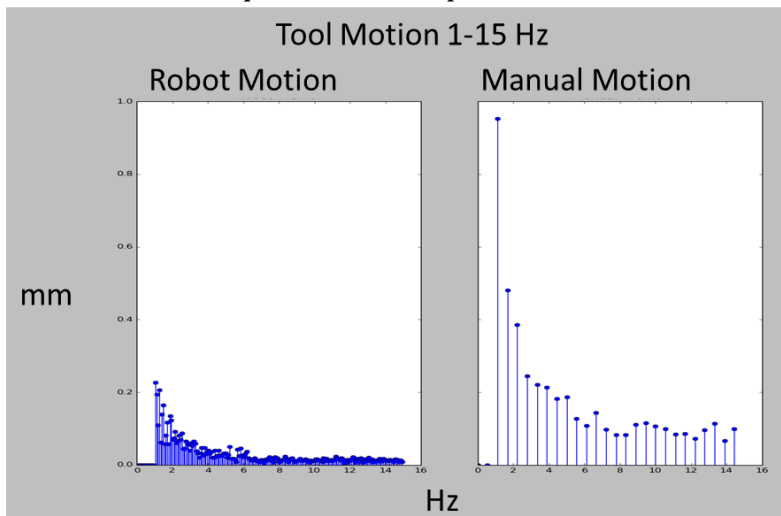


Fig 6: Tool motion from 1-15 Hz

Video: suturing\_robot\_05-04-17-1, suturing\_manual\_05-04-17-1



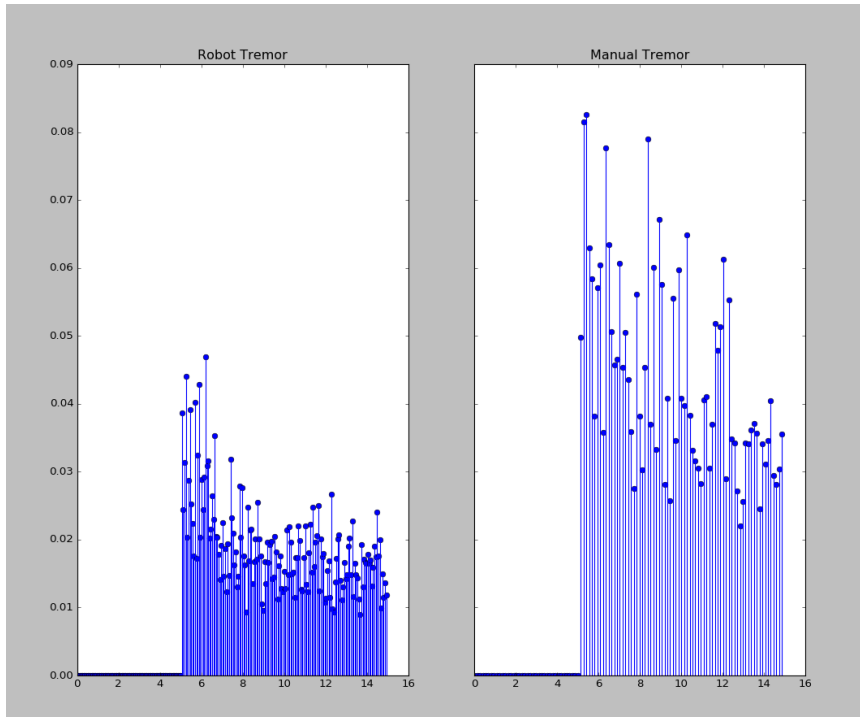


Fig. 7: Tool tremor from 5-15 Hz  
 Video: suturing\_robot\_05-04-17-1, suturing\_manual\_05-04-17-1

Max Tremulous Motion	Magnitude	Frequency
Robot Held Tool	0.050 mm	0.14mm
Hand Held Tool	5.22 Hz	6.67 Hz

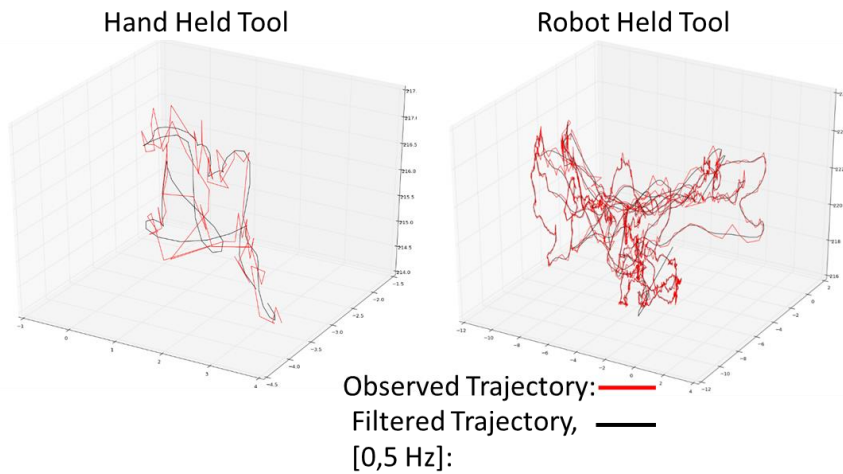


Fig. 8: 3D tool trajectory from hand held tool (left) and robot held tool (right).  
 Background markers with and without microscope motion compensation

**Conclusion**  
Significance



The maximum amplitude of tremulous motion for the robot in the sequence suturing\_robot\_05-04-17-1 is 50 microns at 5.22 Hz. The largest tremor in the hand-held tool is 140 microns at 6.67 Hz. For the robot trajectory analysis, the magnitude of the tremor slowly decays as the frequency increases. But in the hand-held results there is a spike in movement at frequencies around 6Hz, 10 Hz, and 14 Hz. We can see the effect of physiological hand tremor on the tool movement. Comparing the true and filtered tool trajectories, we can see that the smoothed trajectory follows the true robot trajectory much more closely than it follows the manual trajectory. The robot is moving in the direction of intentional motion but disregarding the unintentional hand tremor.

These results are significant because they illustrate how a surgical robot reduces hand tremor and make microsurgery safer. This algorithm can be used to measure surgeon hand tremor and by extension surgeon skill. The inverse trajectory can illustrate the difference between the surgeon's desired path and the actual path with hand tremor. These 3D trajectories can also be analyzed to find applications where surgical robots can make the most difference. Surgeries with complex paths where the desired and actual trajectories are very different are good candidates to have a surgical robot integrated into the workflow.

### Extensions/ Limitations

One limitation of this work is that the tools and background were tracked using color markers. These markers may not be realistic in a real surgical setting. One good thing about the tool tracking is that it does not require any external tracking systems or bulky markers. The painted tools have the same profile as the original tools. A more complicated computer vision approach would be required to track the background without markers in *in-vivo* videos.

My next step in this project is to implement an appearance model for the tool using the known color markers. With the appearance model, we will be able to get the tool position and orientation with detecting a subset of all the markers. This way we will not have to take just the longest continuous trajectory, we will know the tool position throughout the entire video.

A limitation is the code is not real-time. But re-implementing the python code in C++ would make it much faster. The bundle adjustment is already implemented with a sliding window so that could be adjusted to get near real-time performance. Tracking the background markers does not work if all the markers are occluded, but since the markers were designed for a lab setting it is reasonable to require they aren't all covered during the experiment.

### **Acknowledgements**

Thank you Dr. Taylor, Dr. Reiter, and Yunus for your technical support. Thank you Paul for helping me set up the microscope and robot each time I recorded data. Thank you Abhinav for designing the background marker. Thank you Radhika for printing the background marker and for designing and printing the tray. Thank you Alexis for helping me throughout CIS 1 and CIS2!

## Management Summary

### Who did what?

I was a one person team so I did all of the technical work. Abhinav, an undergraduate student, designed the background marker for me. Radhika printed the background holder and designed and printed a tray to hold the background marker.

### Accomplished versus planned:

I met all my final deliverables. Originally I had planned to track the tool using an optical tracker and compare the results with the video tool tracking. We changed these deliverables part way through the semester because I was falling behind schedule and the optical tracking results would not necessarily be more accurate than the video tracking results. I was able get back on schedule and finish the project by the end of the semester.

	Original	Updated
Min	A system capable of measuring tool movement (using existing tracking system)	An algorithm to triangulate 3D points from stereo video and track background motion (with fiducial points)
	Frequency results from tracked tool movement (using existing tracking system)	Tool tracking algorithm using microscope video for painted tool
	An algorithm to triangulate 3D points from stereo video and track background motion (with fiducial points)	
Expected	Tool tracking algorithm using microscope video	Validation of triangulated points using known world motion (robot or measured)
	Frequency analysis of the tool tip motion from a stereo video	Frequency analysis of the tool tip motion from a stereo video
Max	An algorithm to get accurate tool tip motion and tremor from microscope video	Comparison of hand-held and robot-held tool tremor
	Comparison of hand-held and robot-held tool tremor	

### Next steps:

The next step for this project is to run the frequency analysis on video of surgeons suturing animal vessels. I will run the code on new experimental video this summer. The next step to improve the code is to implement an appearance model for the tool.

### Lessons Learned:

The biggest lesson I learned was to prioritize the deliverables at the beginning of a project. I spent a bit of time early in the semester trying to figure out when I could set up the optical tracking system and how to use the OT system. In the end the optical tracking results weren't necessary. In the future I will find out what deliverables are essential, complete those, then try to finish additional work that would be interesting but not essential.

## Technical Appendices

- Code is available in a zip file on the wiki

### Users Guide

#### Tremor Analysis Users Guide

Molly O'Brien

05/16/17

How to use:

TO USE: put all files in Tremor\_Analysis/experiments/Folder\_Name

In the folder:

Calibration/Left\_Imgs/\*

> Left calibration images

> image names should be the frame number

Calibration/Right\_Imgs/\*

> Right calibration images saved in

> image names should be the frame number

left\_video.avi

> left camera experimental video

right\_video.avi

> right camera experimental video

Color\_Markers/Background/images

> images with color markers

> image names should be the frame number

Color\_Markers/Background/binary\_masks

> masks around marker

> image names should be the frame number

Color\_Markers/Background/template.png

> template image

Color\_Markers/Tool/images

> images with tool markers showing

> image names should be the frame number

Color\_Markers/Tool/binary\_masks

> masks around markers

> image names should be the frame number

\* A sample folder with correct file, image and video names is in

Tremor\_Analysis/experiments/suturing\_robot\_05-04-1

\* experiments/sample\_folder is an empty folder with the correct folder structure needed for each experiment

```
# TO CALL FUNCTION:  
cd "path to folder" /Tremor_Analysis/  
python runTremorAnalysis [Folder_Name]
```

## Guide to Files

### Folder: src

#### Tremor\_Analysis

<b>Function</b>	Tremor_Analysis
<b>Input</b>	* Folder_Name: folder with videos, calibration images, and marker information
<b>Return</b>	* saves triangulated 3D points, stabilizing transforms for each frame, plots frequency analysis
<b>Description</b>	* Detects color markers * Performs camera motion compensation * Performs frequency analysis

### Folder: src/Bundle Adjustment

#### File: bundleAdjustment.py

<b>Function</b>	apply_bundle_adjustment
<b>Input</b>	* marker_points: list of array of 3D background marker positions for each frame * T: list of array of 4x4 homogeneous transform from frame n to first frame
<b>Return</b>	* T_bundle: list of array of optimal 4x4 homogeneous transform from frame n to first frame
<b>Description</b>	Given 3D background points for each frame in a video, apply bundle adjustment to get optimal transform from nth frame to first frame. Applies bundle adjustment with a sliding window of 80 frames.

<b>Function</b>	bundleAdjustment
<b>Input</b>	* observations: list of array of background points for each frame * T_initial: list of array of 4x4 initial homogeneous transforms from each frame to the first frame
<b>Return</b>	* P_optimal: 1xn*6 vector with the optimal transforms for the n frames. Each 1x6 vector represents the 4x4 homogeneous transform from points in frame n to points in frame 0.
<b>Description</b>	Given a list of points in each frame and initial transforms for each frame to the first frame, find the optimal transform from each frame to the first frame. The goal is to stabilize background points in microscope video.

<b>Function</b>	extractVector
<b>Input</b>	* T_list: list of N 4x4 homogeneous transforms
<b>Return</b>	* V: 1xN*6 vector. Each transform T_i in T_list is converted into a 1x6 vector v_i = [rx, ry, rz, x, y, z] where rx, ry, and rz are the Rodrigues angles from the rotation in T_i.
<b>Description</b>	Turn a list of N transforms into a 1xN*6 vector. The vector will be optimized in bundle adjustment

<b>Function</b>	ErrorFunction
<b>Input</b>	* P: 1xN*6 vector of parameters being optimized * x: list 3D background points in N frames
<b>Return</b>	* error: 1xN*6 vector, error associated with each transform parameter in P.
<b>Description</b>	Compute the error for a given vector of transforms P being optimized in bundle adjustment. Error for transform i is proportional to the distance and rotation between the points in frame 0 and frame i * transform i

<b>Function</b>	ExtractTransforms
<b>Input</b>	* P: 1xN*6 vector (3 Rodrigues angles, 3 translation variables)
<b>Return</b>	* T: list of N 4x4 homogeneous transforms
<b>Description</b>	Given a vector P with transforms for N frames, extract the transformation for each frame. Return a list of transforms.

<b>Function</b>	FormTransform
<b>Input</b>	* V: 1x6 array (3 Rodrigues angles, 3 translation)
<b>Return</b>	* 4x4 homogeneous transform
<b>Description</b>	Given Rodrigues angles and translation, form a homogeneous transform.

<b>Function</b>	FindError
<b>Input</b>	* points1: Nx3 array of points * points2: Nx3 array of points
<b>Return</b>	* error: 1x6 array of error, sumDistances*[rx ry rz x y z] of rotation between point clouds then translation btwn point clouds

<b>Description</b>	Given 2 pairs of corresponding 3D points, find the rotation and translation error between the point clouds. Weight the rotation and translation errors by the sum of distances between corresponding points in the two point clouds.
--------------------	--

<b>Function</b>	rotatePoints
<b>Input</b>	* points: Nx3 array of points * T: 4x4 homogeneous transform
<b>Return</b>	* points_rot: points transformed by T
<b>Description</b>	Given a list of 3D points, transform by given 4x4 homogeneous transform.

<b>Function</b>	QuatToRotM
<b>Input</b>	* q: quaternion with the scalar value first [qw, qx, qy, qz]
<b>Return</b>	* R: 3x3 rigid rotation matrix
<b>Description</b>	Turn a quaternion into a rotation matrix, assumed quaternion: $q = [w \ x \ y \ z]$ where w is the scalar, and x y z is the vector

<b>Function</b>	dist
<b>Input</b>	* x: point 1 * y: point 2
<b>Return</b>	* D: distance
<b>Description</b>	Find the Euclidean distance between two points with the same dimension.

### Folder: src/Camera Calibration

File: blobParamsApp

<b>Function</b>	createBlobDetectorFile
<b>Input</b>	* Folder_Name: directory with sample images * tool_hist: color histogram for tool markers * N_tool: the channel of the probability image with the tool blob detection
<b>Return</b>	* min_area: int, minimum area of tool marker blobs * max_area: int, max area of tool marker blobs



<b>Description</b>	Let users interactively set blob size parameters for blob detection in tool tracking.
--------------------	---

<b>Function</b>	findGoodBlobs
<b>Input</b>	* image: image from video * prob_im: tool marker probability image of image
<b>Return</b>	* min_area: int, minimum area of tool marker blobs * max_area: int, max area of tool marker blobs
<b>Description</b>	Detect blobs in marker probability image. Let the user change the desired blob size until all markers are detected.

File: stereoCalibration

<b>Function</b>	stereoCalibration
<b>Input</b>	* left_dir: folder with left calibration images * right_dir: folder with right calibration images * camCalFileName: file name to save calibration results * stereoCalFileName: file name to save stereo calibration results * output: file where results & debugging info are saved
<b>Return</b>	* ObjectPoints: the 3D checkerboard corner positions used in calibration * leftCalImS: the left images were the L & R ims had checkerboards detected ( and were used in calibration) * rightCalImS: the right images that were used in calibration
<b>Description</b>	Given left and right images from a stereo camera find the stereo calibration and stereo rectification matrices

File: ICP

<b>Function</b>	findTransform
<b>Input</b>	* A: point cloud 1 (points ordered so that pt 1 in A matches pt 1 in B) * B: point cloud 2
<b>Return</b>	* R: rotation between A and B * t: translation between A and B
<b>Description</b>	Find the rotation between 2 point clouds A and B with point correspondences such that: $R * A + t = B$

<b>Function</b>	quatToRotM (see Bundle Adjustment)
-----------------	------------------------------------

<b>Function</b>	stabilizePoints
<b>Input</b>	* points3D: list of 3D point cloud found in each frame * T: $T[i]$ is the transform from frame $i$ 's coordinate to frame $(i-1)$ 's coordinates. $T[0]*T[1]*...T[i]*points3D[i]$ transforms $points3D[i]$ into the first frame's coordinates
<b>Return</b>	* points3D from all frames transformed into first frame's coordinates
<b>Description</b>	Read in 3D point clouds from video frame and frame-to-frame transforms. Apply frame-to-frame transform iteratively to transform each point cloud into the first frame's coordinates

<b>Function</b>	rotatePoints (see Bundle Adjustment)
-----------------	--------------------------------------

File: loadCheckerBoard

<b>Function</b>	loadCheckerboard
<b>Input</b>	* num: which checkerboard being used. Parameters can be saved for multiple physical checkerboards * num_ims: number of calibration images
<b>Return</b>	* objectPoints: the 3D coordinates of internal checkerboard corners in world coordinates
<b>Description</b>	Create calibration points for checkerboard in "world" coordinates

File: loadParams

<b>Function</b>	LoadCamCal
<b>Input</b>	* filename: camera calibration filename
<b>Return</b>	* foundL: boolean array, true for left calibration images where checkerboard was found * rvecsL: vector of camera rotation vectors with respect to checkerboards in left calibration images * tvecsL: vector of camera translation vectors with respect to checkerboards in left calibration images

	<ul style="list-style-type: none"> <li>* foundR: boolean array, true for right calibration images where checkerboard was found</li> <li>* rvecsR: vector of camera rotation vectors with respect to checkerboards in right calibration images</li> <li>* tvecsR: vector of camera translation vectors with respect to checkerboards in right calibration images</li> <li>* image_list: list of calibration image names</li> </ul>
<b>Description</b>	load camera calibration results

<b>Function</b>	LoadSterCal
<b>Input</b>	* filename: stereo calibration filename
<b>Return</b>	<ul style="list-style-type: none"> <li>* retval: RMS reprojection error (for a good calibration it should be ~ 0.1-1 pixels)</li> <li>* cameraMatrixL: projection matrix for left camera. = <math>[f_x \ 0 \ c_x; 0 \ f_y \ c_y; 0 \ 0 \ 1]</math>. <math>f_x, f_y</math>: x/y focal length. <math>(c_x, c_y)</math>: principle pt (center of image).</li> <li>* distCoeffsL: distortion coefficients of left camera, <math>(k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6)</math> k's: radial distortion, p's: tangential distortion</li> <li>* cameraMatrixR: projection matrix for right camera</li> <li>* distCoeffsR: distortion coefficients for right camera</li> <li>R: rotation matrix btwn 1st and 2nd image (L and R image)</li> <li>T: translation btwn L &amp; R camera coordinate systems</li> <li>E: essential matrix</li> <li>F: fundamental matrix</li> </ul>
<b>Description</b>	Load stereo calibration results

File: readInCalImgs

<b>Function</b>	readInCalImgs
<b>Input</b>	<ul style="list-style-type: none"> <li>* left_dir: folder containing left images</li> <li>* right_dir: folder containing right images</li> <li>* output: file where results &amp; debugging info are saved</li> </ul>
<b>Return</b>	<ul style="list-style-type: none"> <li>* left_ims: list with left calibration images</li> <li>* right_ims: list with right calibration images</li> <li>* image_list: list of calibration image names</li> </ul>
<b>Description</b>	read in images for stereo calibration from left_im_folder and right_im_folder

File: triangulate

<b>Function</b>	triangulate
<b>Input</b>	* camCalFile: camera calibration filename * sterCalFile: stereo calibration filename * pointsL: feature positions (in pixels) in left image * pointsR: feature positions (in pixels) in right image
<b>Return</b>	* worldPoints: feature points in 3D camera coordinates
<b>Description</b>	Given feature position in left and right image, find point position in 3D camera coordinates.

<b>Function</b>	formA
<b>Input</b>	* pointL: feature position (in pixels) in left image * projL: projection matrix for left image * pointR: feature position (in pixels) in right image * projR: projection matrix for right image
<b>Return</b>	* A: [4x3] matrix, will be used to triangulate 3D point position
<b>Description</b>	Given points in left and right images and left and right projection matrices, form matrix A to be used for triangulation.

<b>Function</b>	formB
<b>Input</b>	* pointL: feature position (in pixels) in left image * projL: projection matrix for left image * pointR: feature position (in pixels) in right image * projR: projection matrix for right image
<b>Return</b>	* b: [4x1] vector, will be used to triangulate 3D point position
<b>Description</b>	Given points in left and right images and left and right projection matrices, form matrix b to be used for triangulation.

File: undistortStereoImgs

<b>Function</b>	undistortStereoImgs
<b>Input</b>	* sterCalFile: stereo calibration filename * imageL: left image with lense distortion * imageR: right image with lense distortion
<b>Return</b>	* undistortL: left image with lense distortion removed * undistortR: right image with lense distortion removed
<b>Description</b>	Given camera calibration parameters, remove lense distortion.

File: computeCameraMovement

Function	computeCameraMovement
<b>Input</b>	<ul style="list-style-type: none"> <li>* capLeft: left video</li> <li>* capRight: right video</li> <li>* HistInfo: list of color marker histogram info. For each type of color marker the list has [color marker histogram, number of Gaussian Mixture Model modes, probability image channel for detections, histogram string key, (if tool min blob detection area, max blob detection area)].</li> <li>* camCalFile: camera calibration file</li> <li>* sterCalFile: stereo calibration file</li> <li>* outputFile: text file to write program info to</li> <li>* trialName: experiment name</li> <li>* outL: left output video</li> <li>* outR: right output video</li> <li>* template_L: background marker template for left video</li> <li>* template_points_L: center of background markers in left template image</li> <li>* template_R: background marker template for right video</li> <li>* template_points_R: center of background markers in right template image</li> </ul>
<b>Return</b>	<ul style="list-style-type: none"> <li>* marker_points3D: list of 3D background points in each video frame</li> <li>* tool_points3D: list of 3D tool points in each video frame</li> <li>* frameTransform: 4x4 homogeneous transform from each frame to the first frame</li> </ul>
<b>Description</b>	<ul style="list-style-type: none"> <li>* read in calibration files</li> <li>* take in left and right videos</li> <li>* detect color markers, match in left and right frames</li> <li>* triangulate a 3D feature point cloud for each frame</li> <li>* estimate the 3D transform between each frame</li> </ul>

Function	triangulate_marker_points
<b>Input</b>	<ul style="list-style-type: none"> <li>* frameL: left image</li> <li>* frameR: right image</li> <li>* HistInfo: color histogram info for each color marker</li> <li>* camCalFile: camera calibration file</li> <li>* sterCalFile: stereo calibration file</li> <li>* outputFile: text file to write program info to</li> <li>* trialName: experiment name</li> <li>* outL: left output video</li> <li>* outR: right output video</li> </ul>

	<ul style="list-style-type: none"> <li>* template_L: background marker template for left video</li> <li>* template_points_L: center of background markers in left template image</li> <li>* template_R: background marker template for right video</li> <li>* template_points_R: center of background markers in right template image</li> <li>* frameCount: current frame number</li> <li>* prevMarkerParams: the rotation and translation of the background markers in the previous frame</li> </ul>
<b>Return</b>	<ul style="list-style-type: none"> <li>* marker_points: 3D background marker positions</li> <li>* tool_points: 3D tool marker positions</li> <li>* bestMarkerMatchParams: the best rotation and translation of the background markers in the left and right frames</li> <li>* marker_matches: background marker points in the left and right frames</li> </ul>
<b>Description</b>	<ul style="list-style-type: none"> <li>* read in left and right images</li> <li>* compute marker probability images for each different kind of marker</li> <li>* match background marker to template</li> <li>* detect centers of other markers</li> <li>* return 3D locations of markers</li> </ul>

<b>Function</b>	saveVideo
<b>Input</b>	<ul style="list-style-type: none"> <li>* name: video name</li> <li>* points: list of 3D points</li> </ul>
<b>Return</b>	Saves a video
<b>Description</b>	Takes in a sequence of 3D point clouds, saves each points cloud as a frame in the video.

<b>Function</b>	saveVideoMarkersAndTools
<b>Input</b>	<ul style="list-style-type: none"> <li>* name: video name</li> <li>* marker_points: list of 3D background marker points</li> <li>* tool_points: list of 3D tool marker points</li> </ul>
<b>Return</b>	Saves a video
<b>Description</b>	Takes in a 3D point clouds of marker and tool points, saves animation of 3D points.

<b>Function</b>	updatePoints
-----------------	--------------

<b>Input</b>	
<b>Return</b>	
<b>Description</b>	Helper function to update points in video being saved with marker and tool 3D points

<b>Function</b>	updatePoints1PtCloud
<b>Input</b>	
<b>Return</b>	
<b>Description</b>	Helper function to update points in video being saved with one point cloud

File: dist

<b>Function</b>	dist (see Bundle Adjustment)
-----------------	------------------------------

### Folder: src/Color Marker Detection

File: blobDetector

<b>Function</b>	MSERBlobDetector
<b>Input</b>	* image: probability image * min_area: minimum marker blob area * max_area: maximum marker blob area
<b>Return</b>	* keypoints: blob keypoints
<b>Description</b>	Find blobs in input image.

<b>Function</b>	pruneKeypoints
<b>Input</b>	* keypoints: keypoints
<b>Return</b>	* good_keypoints: pruned keypoints, require at distance of at least 2 pixels between keypoints
<b>Description</b>	Get one keypoint at each location (ignore multiple detections)

File: blobMatch

<b>Function</b>	BlobMatch
<b>Input</b>	* frame1: image 1 * frame2: image 2 * HistInfo: list of color marker histogram info. For each type of color marker the list has [color marker histogram, number of



	Gaussian Mixture Model modes, probability image channel for detections, histogram string key, (if tool min blob detection area, max blob detection area)]. * frameNum: frame number
<b>Return</b>	* matches1: array of feature point locations in image 1 * matches2: corresponding array of feature point locations in image 2 * kp1: array of keypoints in image 1 * kp2: corresponding array of keypoints in image 2
<b>Description</b>	Given 2 images, find blobs, match between the images.

<b>Function</b>	findMatches
<b>Input</b>	* points1: feature points in frame 1 * points2: feature points in frame 2 * d: max distance allowed between matching feature point locations in 2 images
<b>Return</b>	* matches1: array with points in points1 st $F \cdot \text{points1} - \text{points2} < d$ * matches2: array with points in points2 st $F \cdot \text{points2} - \text{points2} < d$ * matchKp1: array of keypoints in image 1 * matchKp2: corresponding array of keypoints in image 2
<b>Description</b>	Match marker feature points between left and right images.

File: createMarkerHistFile

<b>Function</b>	createMarkerHistFile
<b>Input</b>	* Folder_Name: experiment folder name * background_Hist: color histogram for background markers * tool_Hist: color histogram for tool markers * N_m: number of Gaussian mixture model modes for the background marker probability * N_t: number of Gaussian mixture model modes for the tool marker probability
<b>Return</b>	* N_m_select: channel of background probability image with background marker detections * N_t_select: channel of tool probability image with tool marker detections
<b>Description</b>	Let the user pick which probability image channel gives good color marker detections.

File: greedyMatch

<b>Function</b>	greedyMatch
<b>Input</b>	* points1: first point cloud * points2: second point cloud
<b>Return</b>	* ordered_points1: points in first point cloud orders so ordered_points1[i] matches with ordered_points2[i] * ordered_points2: points in second point cloud orders so ordered_points1[i] matches with ordered_points2[i]
<b>Description</b>	Given 2 unordered sets of corresponding points, find matches using greedy search.

File: markerHistogramExtractor

<b>Function</b>	markerHistogramExtractor
<b>Input</b>	* image dir: directory where reference marker images are * mask dir: directory where binary marker masks are
<b>Return</b>	* H_hist: histogram in Hue space * S_hist: histogram in Saturation space
<b>Description</b>	Given images, binary masks for the markers compute a color histogram for the color marker.

<b>Function</b>	ComputeGaussMixModel
<b>Input</b>	* N: int, number of clusters * Hist: histogram computed with np.histogram2d
<b>Return</b>	* em: Gaussian mixture model
<b>Description</b>	Given a histogram, compute a Gaussian Mixture Model with N clusters.

<b>Function</b>	ComputeMarkerProb
<b>Input</b>	* image_dir: directory where images live * GMM: GaussMixModel with prob pixel val in H&S was a marker * N: number of modes in GMM * val_thresh: min value required to be considered to be a marker, needed for blue markers because low val, blue areas are often shadows
<b>Return</b>	* probIm: binary image, white pixel = high marker prob
<b>Description</b>	Given images, hue and saturation histograms of markers, compute the the probability of each pixel in images of being a marker.

Function	markerMatch
<b>Input</b>	<ul style="list-style-type: none"> <li>* image: binary probability image, white pixels = high marker prob</li> <li>* templateIm: binary image with ground truth indiv marker positions. The marker is rigid so the relative pos of indiv markers is constant</li> <li>* templatePts: pixel locations of centers of indiv markers in markerIm</li> <li>* prevEst: boolean flag, are there previous match parameters?</li> <li>* prevParams: translation and rotation of marker match in previous frame</li> </ul>
<b>Return</b>	<ul style="list-style-type: none"> <li>* imagePts: pixel locations of centers of indiv markers in image</li> <li>* params: translation and rotation of marker match</li> </ul>
<b>Description</b>	Match a marker template image to a marker probability image. Return location of all marker keypoints. Check possible marker translations and rotations.

Function	RotatePoints2D
<b>Input</b>	<ul style="list-style-type: none"> <li>* points: array of 2D points</li> <li>* R: 2D rotation</li> <li>* t: 2D translation</li> </ul>
<b>Return</b>	* worldPoints: transformed points
<b>Description</b>	Transform 2D points

### Folder: src/Frequency Analysis

File: buildTrajectory

Function	BuildTrajectory
<b>Input</b>	* list of 3D point clouds in video frames
<b>Return</b>	* list of position trajectories
<b>Description</b>	Take in cloud of 3D points for video frame, associate points across frames to generate trajectories.

Function	PruneTraj
<b>Input</b>	* traj: list of trajectories
<b>Return</b>	* longTraj: list of long trajectories
<b>Description</b>	Only save trajectories with len > MIN_LEN.

<b>Function</b>	buildXYZ_Trajectories
<b>Input</b>	* trajectory: trajectory of 3D points
<b>Return</b>	* x_traj: trajectory of x coordinates * y_traj: trajectory of y coordinates * z_traj: trajectory of z coordinates
<b>Description</b>	Take trajectories of 3D points and separate into list of x, y, and z coordinates.

<b>Function</b>	LinkToolTrajs
<b>Input</b>	* trajectories: list of absolute trajectories
<b>Return</b>	* linked_traj: long trajectory with relative positions from all trajectories
<b>Description</b>	Given short, absolute trajectories, link into 1 continuous trajectory by subtracting the translational offset between traj_n-1[last] and traj_n[first].

<b>Function</b>	frequencyAnalysis
<b>Input</b>	* function: signal to take DFT of * FsBy2: sampling frequency / 2
<b>Return</b>	* f_tremor: the DFT magnitude of function in tremor range
<b>Description</b>	Given a function and a sampling frequency, take the DFT, convert to angle and magnitude representation, high pass filter to see tremor, compare 3D trajectory with and without tremor.

<b>Function</b>	FilterTremor
<b>Input</b>	* w: array with frequency corresponding to each element in f (Hz) * f: frequency component magnitudes
<b>Return</b>	* f: f filtered so components above tremor threshold are 0 * w: w (same as before)
<b>Description</b>	Given trajectory frequency analysis results, find the frequency components magnitude in the tremor region.

<b>Function</b>	ComparePaths
<b>Input</b>	* f: frequency component magnitudes * tremor_thresh: tremor threshold (Hz) * w: array with frequency corresponding to each element in f (Hz)
<b>Return</b>	* plot 3D trajectories

<b>Description</b>	Given a DFT and a tremor threshold plot the path with tremor and the path without.
--------------------	--

<b>Function</b>	InverseToPath
<b>Input</b>	* fx: DFT(x(t)) * fy: DFT(y(t)) * fz: DFT(z(t))
<b>Return</b>	* path: 3D path in time
<b>Description</b>	Given a DFT, find the path in world coordinates.

<b>Function</b>	ConvertDFT
<b>Input</b>	* f: DFT in OpenCV format * N: length of original signal * FsBy2: sampling frequency / 2
<b>Return</b>	* f_mag: magnitude of frequency response at each frequency * f_ang: angle of frequency response at each frequency * w: array with frequency corresponding to each element in f (Hz)
<b>Description</b>	Convert DFT from OpenCV format to magnitude and angle.

<b>Function</b>	RunFrequencyAnalysis
<b>Input</b>	* tool_points3D: 3D trajectory
<b>Return</b>	* tool_x_linked: magnitude of frequency response of x(t) from linked trajectories (see linkToolTraj) * tool_y_linked: magnitude of frequency response of y(t) from linked trajectories (see linkToolTraj) * tool_z_linked: magnitude of frequency response of z(t) from linked trajectories (see linkToolTraj) * Fs/2: sampling frequency / 2
<b>Description</b>	Call this function to run the frequency analysis on a set of 3D points.

<b>Function</b>	CompareRobotAndManual
<b>Input</b>	<i>None, manually change experiment name</i>
<b>Return</b>	* Plots hand-held and robot-held tool tracking results side by side
<b>Description</b>	Plot hand-held and robot-held tool tracking results side by side.

<b>Function</b>	beforeAfterMotionCompensation
<b>Input</b>	<i>None</i>

<b>Return</b>	Plots tracking results with and without camera motion compensation side by side.
<b>Description</b>	Plot tracking results with and without camera motion compensation side by side.

## References

- [1] S. P. N. Singhy and C. N. Riviere, "Physiological tremor amplitude during retinal microsurgery," in *Bioengineering Conference*, Philadelphia, 2002.
- [2] R. N. Stiles, "Frequency and displacement amplitude relations for normal hand tremor," *Journal of Applied Physiology*, vol. 40, no. 1, pp. 44-54, 1976.
- [3] K. C. Olds, P. Chalasani, P. Pacheco-Lopez, I. Iordachita, L. M. Akst and R. H. Taylor, "Preliminary evaluation of a new microsurgical robotic system for head and neck surgery," in *International Conference on Intelligent Robots and Systems*, Chicago, 2014.
- [4] L. Bouarfa, O. Akman, A. Schneider, P. P. Jonker and J. Dankelman, "In-vivo real-time tracking of surgical instruments in endoscopic video," *Minimally Invasive Therapy & Allied Technologies*, pp. 129-134, 2012.
- [5] R. Taylor, "Computer-Integrated Surgery: Point cloud to point cloud rigid transformations," 2016. [Online]. Available: <http://www.cs.jhu.edu/cista/445/Lectures/Rigid3D3DCalculations.pdf>. [Accessed 11 May 2017].



