Imaging of Residual Limbs Using Motion Processing with IMUs

by

Rebecca Hope Steinmeyer

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology

June 2017

Signature of Author: Signature redacted

Department of Mechanical Engineering
May 12, 2017

Certified by: Signature redacted

Hugh Herr
Associate Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by: Signature redacted

Rohit Karnik
Professor of Mechanical Engineering
Undergraduate Officer

Imaging of Residual Limbs using Motion Processing with IMUs

by

Rebecca Hope Steinmeyer

Submitted to the Department of Mechanical Engineering
on May 12, 2017 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

ABSTRACT

Accurate imaging of residual limbs is necessary for the design of well-fitting sockets for prosthesis wearers. Unfortunately, current methods of acquiring residual limb geometry are often expensive and inaccessible. A measurement method is proposed using coordinated IMUs to achieve residual limb imaging through motion processing. The IMUs are fixed to an object which traces the surface of the residual limb. Trajectories are calculated for each IMU, and a correction method is applied using all IMUs fixed to the instrument surface to mitigate measurement drift. The IMU trajectories are then used to generate a triangulated geometry to digitally represent the residual limb. This method was simulated to guide instrument design and provide insight on performance and measurement process. The eventual goal is a glove with IMUs at the fingertips which may be used by an untrained individual, who may simply wear the glove and lightly survey the surface of the residual limb with their hand to produce data which will then be used to generate a digital limb geometry. Using the results of the simulation, a design is proposed for the glove.

Thesis Supervisor: Hugh Herr
Title: Associate Professor of Media Arts and Sciences

Acknowledgements

I would first like to thank Professor Hugh Herr of the Biomechatronics Group in the MIT Media Lab for his gracious support and motivation. Professor Herr's research was a main source of inspiration for my choice to pursue mechanical engineering, and the chance to be involved in the Biomechatronics Group excited me long before I arrived at MIT.

I also wish to express my thanks to current and past members of the Biomechatronics Group Dr. Luke Mooney, Dr. Kevin Moerman, and Stephanie Ku for their advice and guidance throughout this process, and to all the members of the Biomechatronics Group and Media Lab community with whom I have had the pleasure to engage. I would also like to thank the Media Lab for its support.

I would also like to express my gratitude to the faculty of the Department of Mechanical Engineering for their inspiration and teaching throughout my undergraduate experience at MIT.

Finally, I would like to thank my family for their unyielding love and encouragement, and my friends within and outside of MIT for filling this journey with joy.

Table of Contents

List of Figures

List of Tables

1 Introduction

The number of amputees in the United States is estimated to be approximately two million, and this number continues to grow every year [1]. Of this population, most prosthesis-wearers use their prosthesis for upwards of eight hours per day [2]. Furthermore, amputees cite comfort as their most common prosthesis concern [3]. A well-fitting socket is therefore crucial for high quality of life for amputees.

Accuracy and comfort in socket fitting depends highly on the skill of the prosthetist, and the generation of an accurate limb geometry is currently an expensive process [4]. However, although they are expensive, digital methods for collecting a three-dimensional residual limb geometry have been proven to be both accurate and consistent [5, 6]. Such methods include noninvasive MRI procedures [7], laser scanning techniques [8, 9], and ultrasound [10].

The current state-of-the-art is not only expensive, but is also difficult to access. An ideal system would allow an individual to access inexpensive and accurate residual limb geometry at any point in time in order to ensure consistent socket comfort.

In order to allow accessibility and accuracy in limb geometry measurements, a novel residual limb geometry measurement method is here proposed. The method uses coordinated six degree of freedom inertial measurement units (IMUs) to calculate a trajectory in three-dimensional space tracing the surface of the limb. The IMUs are placed on the fingertips of a glove such that an individual wearing the glove may simply gently rub the residual limb, taking care to trace over as much of the surface area as possible. The trajectories of each IMU are calculated and used to generate a triangulated three-dimensional geometry representative of the residual limb. A simulation was conducted to evaluate the feasibility of this measurement

method, to provide realistic simulated measurements for a given instrument, and to inform instrument design for accurate residual limb geometry reconstruction.

2 Geometry Measurement Method

A geometry may be measured and reconstructed by calculating a position trajectory of an object guided across its surface. The proposed method to measure residual limb geometry using motion processing with six degree of freedom inertial measurement units (IMUs), each consisting of a three-axis accelerometer and a three-axis gyroscope. The measurement instrument will consist of multiple IMUs fixed to objects with known relative IMU positions and orientations.

During the measurement process, this instrument is guided across the surface of the residual limb in a light back-and-forth rubbing motion over a short period of time. Trajectories are calculated for each IMU, and the relative trajectories of IMUs fixed to the same surface are used to apply a correction method in order to prevent drift in calculating the instrument trajectory. A triangulation method is applied to the corrected position trajectory of the instrument in order to generate a three-dimensional shape, which represents the geometry of the residual limb.

2.1 Motion Processing with IMUs

A six degree of freedom IMU consisting of a three-axis accelerometer and a three-axis gyroscope measures linear acceleration and angular rotation from the accelerometer and gyroscope, respectively. Assuming initial orientation and position are known for each sensor,

orientation over time is first calculated using the gyroscope data. Calculated orientation is then used to rotate accelerometer data to the Earth reference frame, and velocity and position are calculated using numerical integration.

Ideally, a measurement process includes an accurate calibration routine to calculate determine sources of error intrinsic to the IMU and a motion processing method which takes these error sources into account. The simulation in this thesis assumes the use of an accurate calibration method, and utilizes motion processing methods that account for common sources of IMU error.

2.1.1 Background Mathematics

Quaternions may be used as a computationally inexpensive representation of IMU rotation for orientation calculations [11]. The motion processing and simulation methods to be described proximately all rely on quaternions to calculate vector rotation and IMU orientation calculations. Quaternions provide an alternate method to Euler angles to describe three-dimensional rotation, proving advantageous through a significant reduction in computational expense and the elimination of ± 180° uncertainties.

A quaternion is a four-element vector which fully describes a single rotation. As depicted in the simple graphic in Figure 2-1, quaternion rotation may be visualized as a rotation of angle θ about a three-dimensional vector $\boldsymbol{V}$. The full quaternion vector $\boldsymbol{q}$ is calculated as follows:

$$\boldsymbol{q} = \left[\cos\frac{\theta}{2} \quad V_x \sin\frac{\theta}{2} \quad V_y \sin\frac{\theta}{2} \quad V_z \sin\frac{\theta}{2}\right]$$

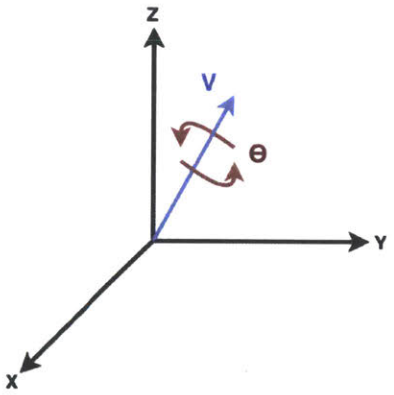


Figure 2-1: Quaternion Reference

Two quaternion operations are of special importance for quaternion rotation. The first is the quaternion conjugate, denoted by $\boldsymbol{q}^*$, which is calculated for a given quaternion $\boldsymbol{q}$ as follows:

$$\boldsymbol{q}^* = [q_1 \quad -q_2 \quad -q_3 \quad -q_4]$$

The second is the quaternion cross product, which for two quaternions $\boldsymbol{q_A}$ and $\boldsymbol{q_B}$ is calculated as follows:

$$\boldsymbol{q_A} \times \boldsymbol{q_B} = \begin{bmatrix} -q_{A_3}q_{B_3} - q_{A_2}q_{B_2} - q_{A_4}q_{B_4} + q_{A_1}q_{B_1} \\ q_{A_2}q_{B_1} + q_{A_3}q_{B_4} + q_{A_1}q_{B_2} - q_{A_4}q_{B_3} \\ q_{A_4}q_{B_2} + q_{A_1}q_{B_3} + q_{A_3}q_{B_1} - q_{A_2}q_{B_4} \\ q_{A_1}q_{B_4} + q_{A_4}q_{B_1} + q_{A_2}q_{B_3} - q_{A_3}q_2 \end{bmatrix}^T$$

These operations are combined to calculate the rotation of a vector $\boldsymbol{v}$ by quaternion $\boldsymbol{q}$, producing a resultant rotated vector $\boldsymbol{v_{rot}}$:

$$\boldsymbol{v_{rot}} = (\boldsymbol{q} \times [0 \; v_x \; v_y \; v_z]) \times \boldsymbol{q}^*$$

In addition to the quaternion methods described here, another important process for IMU motion processing methods is the conversion of matrix to a new reference frame. This is

particularly useful for processing accelerometer data, which in its raw form is given in terms the intrinsic IMU basis vectors. Not only must the data therefore be converted to the Earth reference frame to calculate velocity and position, but if the sensor is calibrated such that any mechanical misalignment of the accelerometer or gyroscope axes is known, the raw data must be modified to account for this error. Consider a vector $\boldsymbol{x}$ with components in defined in a coordinate frame denoted by 3x3 matrix $\boldsymbol{A}$, which is described in Earth frame coordinates. The vector $\boldsymbol{x_{Earth}}$ describes the vector $\boldsymbol{x}$ in terms of Earth frame coordinates, and may be calculated as follows:

$$\boldsymbol{x_{Earth}} = \boldsymbol{A} \times \boldsymbol{x}$$

Motion processing and simulated data generation for IMUs also rely on numerical integration and differentiation. The processes to be described rely primarily on the forward Euler method for numerical integration and the forward finite difference method for numerical differentiation. Both were chosen for their low computational cost, and with a high enough sampling rate allow sufficient accuracy in motion processing and simulated data generation.

2.1.2 Calibration and Sensor Error Management

IMUs are subject to significant intrinsic error, which poses significant challenge to accurate motion processing. Four common sources of error are axis misalignment, constant offset, sensitivity scale factor, and noise, though additional error may result due to other error sources including nonlinearity and moving bias [12]. The simulation presented assumes accurate calibration to determine axis misalignment, offset, and sensitivity, neutralizing their effects on raw IMU data and filtering to mitigate the effects of noise.

Common methods for IMU calibration include the use of high-accuracy turntables [13] and in-field calibration methods which require no external devices [14]. Accuracy varies

significantly across calibration methods; for the purposes of residual limb requirement, since high accuracy is crucial, a high-precision calibration method is assumed. The simulation therefore mitigates error by executing the motion processing methods using the exact error parameters applied to the raw simulated data by the virtual IMU.

2.1.3 Trajectory Calculation

IMU trajectory calculation consists of two distinct steps. First, orientation is calculated using the angular velocity data from the gyroscope and a known initial orientation (or an approximate initial orientation found using the direction of gravity from the accelerometer data). Position is then calculated by rotating the accelerometer data from the IMU reference frame (simply according to its basis vectors) into the Earth reference frame, then applying numerical integration to calculate velocity and position over time. The motion processing methods described here assume a calibrated system such that constant offset, sensitivity scaling, and axis misalignment are known.

When calculating orientation, first the known offset is removed from the raw data. The raw data is then scaled to workable units (for example, radians per second) using the known sensitivity, and axis misalignment error is removed. The angular velocity data is used in conjunction with quaternion orientation representation to calculate orientation over time. The final form of the calculated orientation consists of three 3x1 unit vectors (combined to form a 3x3 orientation matrix), to describe the x, y, and z basis vectors of the IMU coordinate frame.

To calculate position, offset is first removed from the raw data, then the data is scaled to workable units and axis misalignment error is removed. Accelerometer data is then rotated from the IMU frame of reference to the Earth frame, and the gravity vector is removed by simple subtraction. The method then uses numerical integration to calculate velocity and position.

Known initial velocity may also be taken into account at this point; however, initial velocity will be assumed to be zero during physical data collection.

2.2 Coordinated IMU Trajectory Correction

The propensity of IMUs for dramatic error accumulation over a short period of time merits the introduction of multiple sensors for a single trajectory calculation. A common method for IMU trajectory calculation correction is the coordination of the IMU with the Global Positioning System (GPS) to improve position accuracy [15]. However, the small-scale requirements of residual limb geometry reconstruction and the goal of a self-contained system suggest that the GPS correction method is impractical for this application. Another method of IMU trajectory correction is the introduction of redundant IMUs, which has been demonstrated to notably improve IMU trajectory measurements [16, 17].

The motion processing method used in limb measurement must have an implicit correction method integrating the redundant IMUs to prevent unacceptable levels of measurement drift. Two distinct methods of trajectory correction were designed for this simulation: correction by averaging the trajectories from multiple IMUs, and correction by constantly constricting the positions and orientations of IMUs on a fixed surface to their known relative values.

2.2.1 Averaging Correction Method

The averaging correction method is a computationally inexpensive position and orientation correction strategy, relying on the idea that points on a fixed surface experience position and orientation changes at the same rate.

Orientation correction specifically relies on the fact that all fixed points on a surface rotating in space experience the same angular velocity. The averaging correction method for orientation first performs the basic orientation calculation method (described in Section 2.1.3). An approximate angular velocity is calculated by taking the difference between orientations at consecutive time steps. This angular velocity is averaged across all IMUs, and each IMU orientation trajectory is recalculated using the overall average IMU angular velocity.

The position averaging correction process is slightly more complicated: since each IMU experiences its own orientation trajectory throughout the motion path, each set of accelerometer data (one per IMU) must first be converted to Earth reference frame coordinates (as seen in Section 2.1.3). Once the accelerometer data is in Earth frame coordinates, similar to the orientation correction process, the velocity is calculated using the Euler method and an average velocity is calculated across all IMUs. In the Earth frame, each IMU experiences the same angular velocity, so this velocity is assigned to each IMU and the Euler method is used again to calculate position at each IMU.

2.2.2 Instrument Shape Correction Method

The instrument shape correction method applies a correction to all IMU orientation and position calculations at every time step. The goal of the correction process is to restrict all orientation and position methods at every point in time to their known relative positions and orientations on the measurement instrument shape. The method requires knowledge of exact IMU positions and orientations on the instrument; as such, precision in instrument manufacturing becomes absolutely necessary.

The orientation shape correction method relies on the fact that all IMUs fixed to the measurement maintain the same relative orientation throughout the entire measurement process.

Two unique IMUs with with orientations described according to the method in Section 2.1.3 have orientation matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively:

$$\boldsymbol{A} = \begin{bmatrix} A_{x_1} & A_{y_1} & A_{z_1} \\ A_{x_2} & A_{y_2} & A_{z_2} \\ A_{x_3} & A_{y_3} & A_{z_3} \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} B_{x_1} & B_{y_1} & B_{z_1} \\ B_{x_2} & B_{y_2} & B_{z_2} \\ B_{x_3} & B_{y_3} & B_{z_3} \end{bmatrix}$$

The matrix $\boldsymbol{N}$ describes the normalized difference between the two orientations, using the distance between the x, y, and z basis vectors of each IMU:

$$\boldsymbol{N} = \begin{bmatrix} \sqrt{(A_{x_1} - B_{x_1})^2 + (A_{x_2} - B_{x_2})^2 + (A_{x_3} - B_{x_3})^2} \\ \sqrt{(A_{y_1} - B_{y_1})^2 + (A_{y_2} - B_{y_2})^2 + (A_{y_3} - B_{y_3})^2} \\ \sqrt{(A_{z_1} - B_{z_1})^2 + (A_{z_2} - B_{z_2})^2 + (A_{z_3} - B_{z_3})^2} \end{bmatrix}$$

For IMUs fixed on a given surface, although each IMU experiences orientation changes throughout the span of measurement, the normalized difference between each IMU orientation will remain the same.

The instrument shape correction method for orientation calculates the initial normalized orientation difference across all IMUs, and constructs a shift matrix **S** for each IMU of the same size as the orientation matrix. For two unique IMUs with orientation matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we may denote the orientation shift matrices as $\boldsymbol{S_A}$ and $\boldsymbol{S_B}$, respectively:

$$\boldsymbol{S_A} = \begin{bmatrix} S_{A_{x_1}} & S_{A_{y_1}} & S_{A_{z_1}} \\ S_{A_{x_2}} & S_{A_{y_2}} & S_{A_{z_2}} \\ S_{A_{x_3}} & S_{A_{y_3}} & S_{A_{z_3}} \end{bmatrix} \qquad \boldsymbol{S_B} = \begin{bmatrix} S_{B_{x_1}} & S_{B_{y_1}} & S_{B_{z_1}} \\ S_{B_{x_2}} & S_{B_{y_2}} & S_{B_{z_2}} \\ S_{B_{x_3}} & S_{B_{y_3}} & S_{B_{z_3}} \end{bmatrix}$$

At each time step, an orientation shift is applied to each IMU, and the normalized difference between each IMU pair is calculated given the updated (shifted) orientation for each IMU. For two IMUs with orientation matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ at a given time step, and with an initial normalized difference between the two IMUs given by matrix $\boldsymbol{N_0}$, the following matrix is minimized to calculate the orientation shift matrices $\boldsymbol{S_A}$ and $\boldsymbol{S_B}$ for the two IMUs:

$$\begin{bmatrix} \sqrt{\left(\left(A_{x_1}+S_{A_{x_1}}\right)-\left(B_{x_1}+S_{B_{x_1}}\right)\right)^2+\left(\left(A_{x_2}+S_{A_{x_2}}\right)-\left(B_{x_2}+S_{B_{x_2}}\right)\right)^2+\left(\left(A_{x_3}+S_{A_{x_3}}\right)-\left(B_{x_3}+S_{B_{x_3}}\right)\right)^2}-N_{0_x} \\ \sqrt{\left(\left(A_{y_1}+S_{A_{y_1}}\right)-\left(B_{y_1}+S_{B_{y_1}}\right)\right)^2+\left(\left(A_{y_2}+S_{A_{y_2}}\right)-\left(B_{y_2}+S_{B_{y_2}}\right)\right)^2+\left(\left(A_{y_3}+S_{A_{y_3}}\right)-\left(B_{y_3}+S_{B_{y_3}}\right)\right)^2}-N_{0_y} \\ \sqrt{\left(\left(A_{z_1}+S_{A_{z_1}}\right)-\left(B_{z_1}+S_{B_{z_1}}\right)\right)^2+\left(\left(A_{z_2}+S_{A_{z_2}}\right)-\left(B_{z_2}+S_{B_{z_2}}\right)\right)^2+\left(\left(A_{z_3}+S_{A_{z_3}}\right)-\left(B_{z_3}+S_{B_{z_3}}\right)\right)^2}-N_{0_z} \end{bmatrix}$$

The orientation shift matrices are calculated to minimize the above system for each set of two IMUs fixed to the measurement instrument. The total resulting system, empirically solved in MATLAB, consists of $3\times\binom{n}{2}$ nonlinear equations, where n is the number of total IMUs fixed to the instrument. The minimized system provides a unique shift variable assigned to each of the total $9n$ orientation components in the system.

After the shift matrices are calculated, the correction method updates each orientation matrix. For the two IMU example, the updated orientation matrices are given by $\boldsymbol{A_{new}}$ and $\boldsymbol{B_{new}}$:

$$\boldsymbol{A_{new}} = \boldsymbol{A} + \boldsymbol{S_A} \qquad \boldsymbol{B_{new}} = \boldsymbol{B} + \boldsymbol{S_B}$$

The position shape correction method, like the orientation correction method, relies on the knowledge that relative positions of the IMUs remain constant throughout the measurement span. As in the averaging position correction method, at each point in time the x, y and z positions must first be converted to the Earth reference frame. Following a similar process to the

orientation correction method, consider two unique IMUs at a given time with position vectors $\boldsymbol{C}$ and $\boldsymbol{D}$, respectively (assume both position vectors have already been converted to Earth frame coordinates):

$$\boldsymbol{C} = [C_x \quad C_y \quad C_z] \qquad \boldsymbol{D} = [D_x \quad D_y \quad D_z]$$

The matrix $\boldsymbol{M}$ describes the normalized difference between the two positions, which for two 1x3 vectors amounts to simply:

$$\boldsymbol{M} = \begin{bmatrix} |C_x - D_x| \\ |C_y - D_y| \\ |C_z - D_z| \end{bmatrix}$$

As in the orientation correction method, each IMU is assigned a shift vector, in this case denoted by $\boldsymbol{S_C}$ and $\boldsymbol{S_D}$:

$$\boldsymbol{S_C} = [S_{C_x} \quad S_{C_y} \quad S_{C_z}] \qquad \boldsymbol{S_D} = [S_{D_x} \quad S_{D_y} \quad S_{D_z}]$$

Assume that $\boldsymbol{M_0}$ is the initial normalized position difference between two specific IMUs. The correction routine then calculates position shift by minimizing the following matrix for each IMU pair:

$$\begin{bmatrix} |(C_x + S_{C_x}) - (D_x + S_{D_x})| - M_{0_x} \\ |(C_y + S_{C_y}) - (D_y + S_{D_y})| - M_{0_y} \\ |(C_z + S_{C_z}) - (D_z + S_{D_z})| - M_{0_z} \end{bmatrix}$$

Once the shift matrices have been solved for each IMU, the correction method updates each the position vector:

$$C_{new} = C + S_C \qquad D_{new} = D + S_D$$

Both the orientation and position shape correction methods are applied at every time step, across all IMUs. This method is significantly more computationally expensive than the averaging correction method.

2.3 Three-Dimensional Geometry Reconstruction

After generating an array of corrected position matrices for each IMU, I generated an instrument position matrix. The instrument position was calculated by taking the average position trajectory across all IMUs, after adjusting each IMU trajectory to begin at the origin. Finally, the instrument position is adjusted to account for the offset between the instrument origin and the geometry surface.

3 Simulation

The goal of the simulation is the generation and processing of realistic IMU data in order to test the viability of the proposed three-dimensional imaging method for residual limbs. The user first sets inputs for desired quantities, according to the metrics in Table 3-1.

After the user sets parameters, the simulation sets a motion path including a position trajectory over time and iterative quaternions to update orientation at each time step.

The simulation then generates a measurement instrument of the specified shape, and places the desired number of IMUs at random locations on the instrument surface. Each IMU is assigned intrinsic error parameters according to the specified error level.

Input Parameter	Description
sim_time	Simulation time (seconds)
sampling_rate	IMU sampling rate (Hz)
imu_num	Number of IMUs fixed to measurement instrument surface
ADC_length	Resolution of IMU analog-to-digital converter (bits)
A_range	Upper bound for accelerometer measurement (g)
G_range	Upper bound for gyroscope measurement (°/s)
geom_shape	Shape of geometry for simulated measurement (Options: sphere, cube, hemiellipsoid, biologically-inspired geometry)
instrument_shape	Shape of instrument for simulated measurement (Currently only one option: sphere)
correction_method	Correction method to improve IMU measurement accuracy (Options: averaging, instrument shape, none)
error_level	IMU intrinsic sensor error (including axis misalignment, bias, sensitivity error, and noise); normalized to MPU-6050 at error level 5
error_type	Allows error to be applied only partially to accelerometers and gyroscopes for testing purposes (Options: all, noise only, gyroscope only, accelerometer only, gyroscope noise only, accelerometer noise only, none)

Table 3-1: Input Parameters for IMU Correction and Geometry Imaging Simulation

Next, the virtual IMU method produces realistic accelerometer and gyroscope data for each IMU on the surface of the instrument, following the previously calculated trajectory. The orientation and position of each IMU is calculated, and a correction method for each is applied (as described in Section 2.2). Trajectories from all IMUs are then combined to produce an overall instrument trajectory, which is corrected to account for the offset between the instrument center and the geometry surface. Finally, the simulation generates a three-dimensional triangulated figure to represent the geometry measured by the instrument.

3.1 Virtual IMU

As the virtual IMU determines the integrity of the entire simulation, I designed it to reflect real, physical IMUs as accurately as possible. Therefore, it takes into account sensor error and measurement limitations intrinsic to analog-to-digital converters. The virtual IMU accepts the trajectory and IMU parameters produced in the simulated test path and IMU setup routines, and applies them to produce realistic accelerometer and gyroscope data for each IMU on the measurement instrument.

For a series of N measurements, for each sensor, the virtual IMU accepts an Nx3 position matrix, an Nx4 gyroscope matrix, a 3x3 initial orientation, and a time vector of length N. In order to account for intrinsic IMU error, it also receives a 3x6 IMU axis alignment matrix, a 3x2 offset matrix, a 1x2 sensitivity matrix, an Nx6 noise matrix, and a scalar ADC resolution for the IMU.

In order to produce simulated gyroscope data, the angular velocity of the sensor is first calculated at each time step directly from the quaternion matrix (see Section 2.1.1 for reference on quaternion mathematics). The resultant angular velocity is adjusted to reflect the misaligned coordinate frame, and divided by the sensitivity to convert the data to least standard bits (LSB), the standard units for raw IMU data. Finally, the virtual IMU adds offset and noise to the gyroscope data.

To produce simulated accelerometer data, the initial orientation is first adjusted to reflect the misaligned coordinate frame, then rotated through time according to the quaternion matrix. The acceleration is calculated from the position matrix using numerical differentiation, and gravity is added to reflect actual physical conditions. The resultant acceleration is adjusted to

reflect the misaligned coordinate frame, and divided by the sensitivity to convert the data to LSB. Finally, the virtual IMU adds offset and noise to the accelerometer data.

3.1.1 Error Parameters

The virtual IMU incorporates errors due to sensitivity, axis misalignment, and constant offset (see Section 2.1.2 for background on IMU error). These error parameters are assigned to each IMU, and are randomly generated according to a normal distribution about the ideal (the zero error case) value with a maximum magnitude determined by the user's designated error level.

These error parameters are designed such that input error level 5 designates error intrinsic to a standard MPU-6050 IMU (a standard commercial IMU), and directly scaled to other error level values. Some relevant MPU-6050 parameters are as follows [18]:

MPU-6050 Data Sheet Parameter	Value	Units
Gyroscope Full-Scale Range	± 250, ± 500, ± 1000, ± 2000	°/s
Gyroscope ADC Word Length	16	bits
Gyroscope Sensitivity Scale Factor	131, 65.5, 32.8, 16.4	LSB/(°/s)
Gyroscope Sensitivity Scale Factor Tolerance	±3	%
Gyroscope Total RMS Noise	0.05	°/s-rms
Accelerometer Full-Scale Range	± 2, ± 4, ± 8, ± 16	g
Accelerometer ADC Word Length	16	bits
Accelerometer Sensitivity Scale Factor	16384, 8192, 4096, 2048	LSB/g
Accelerometer Initial Calibration Tolerance	±3	%

Table 3-2: Relevant MPU-6050 Data Sheet Parameters

Figure 3-1 shows the effect of the addition of IMU error at various error level parameters on accelerometer and gyroscope data generated by the virtual IMU for a simple circular translation with no rotation. Additionally, a set of raw data for a trajectory using the MPU-6050 is provided in Figure 3-2. Although the trajectory in Figure 3-2 is significantly more dynamic than the simulated path used in Figure 3-1, we may conclude by qualitative observation that Error Level 5 is a reasonable match to MPU-6050 performance.

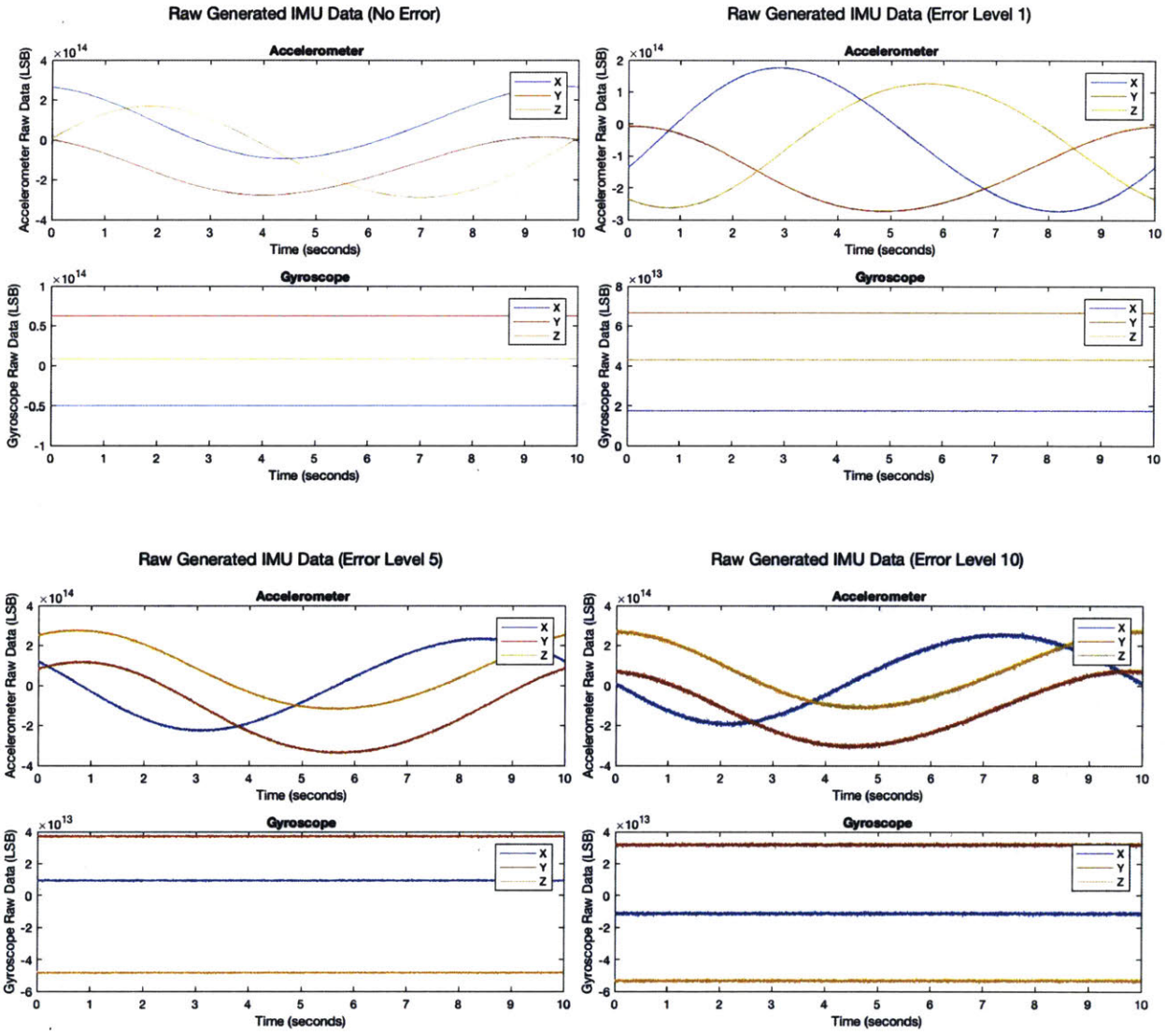


Figure 3-1: Simulated IMU Sample Error

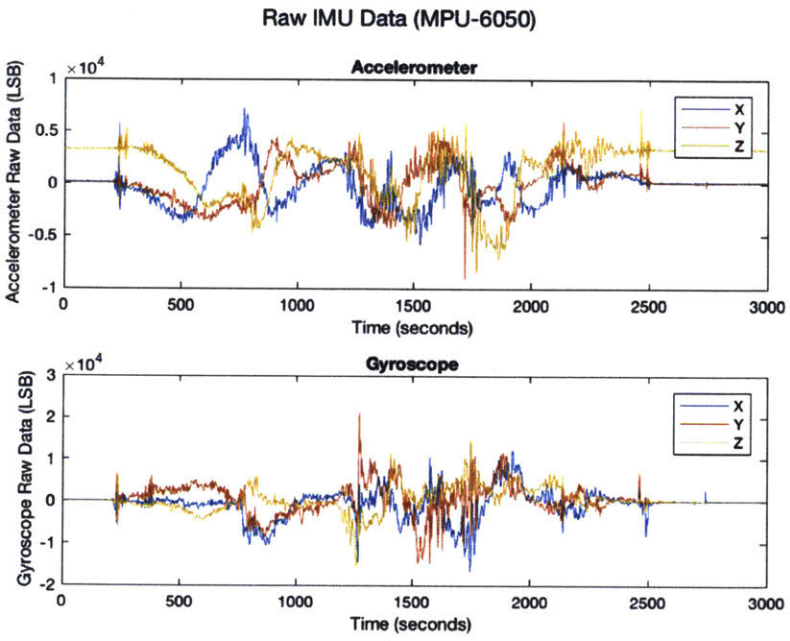


Figure 3-2: Raw MPU-6050 Data

3.2 Basic Simulated Test Path

While the simulation is intended to demonstrate three-dimensional geometry measurement, I also developed a basic circular test path to test and data generation and motion processing, and to compare motion processing correction methods (Figure 3-2).

The path consists of a circular rotation with motion in the x, y, and z directions. At angle θ (where $0 \leq \theta \leq 2\pi$), the position in three-dimensional space is calculated as follows:

$$\boldsymbol{x} = 5 * \left([\cos\theta \quad \sin\theta \quad 0] - \frac{\cos\theta + \sin\theta}{3} [1 \quad 1 \quad 1] \right)$$

The test path generation method then shifts the position such that its trajectory starts at the origin, and calculates the initial velocity of the trajectory (by taking the derivative of the above position equation).

Orientation is calculated so as to depend on the location of the instrument relative to the path's center. Basis vectors in the x, y, and z directions are then computed and used to back-solve for a rotation quaternion matrix. The rotation quaternion is calculated entirely empirically, since an attempt to solve symbolically for the expression for a quaternion resulting from basis vector rotation does not lend itself to a simple solution.

This test path was chosen for its simplicity and smoothness, and since it requires translation and rotation across x, y, and z. The fact the trajectory started and ended at the same point also allowed quick evaluation of the success of motion processing and correction methods.

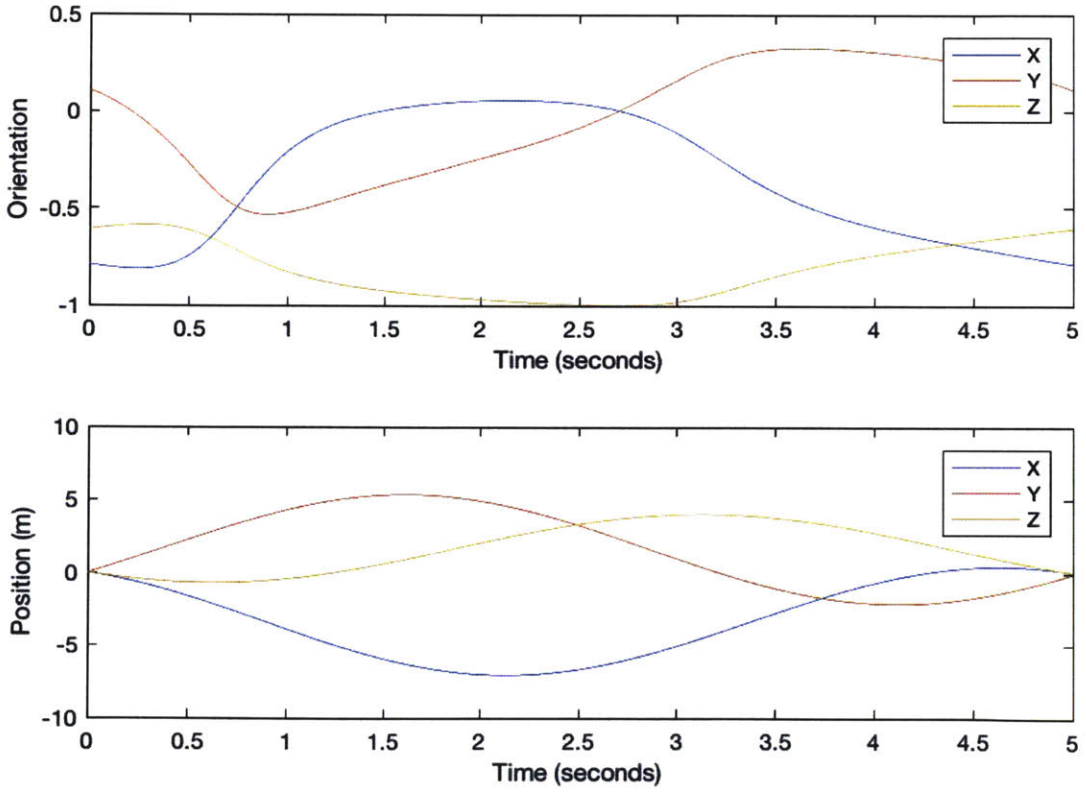


Figure 3-3: Ideal Test Path

3.3 Instrument Generation

I designed the instrument generation method to produce an instrument of the shape and number of IMUs provided by the simulation user in the overall simulation parameters.

The instrument generation method first calculates the *x*, *y*, and *z* coordinates of the instrument shape, centered at the origin. The specified number of IMUs are then placed at random locations on the instrument shape. The *x*, *y*, and *z* basis vectors for each IMU are calculated such that the *z* vector extends orthogonal to the IMU base.

All simulations conducted thus far have utilized a spherical instrument geometry. Therefore, the *z* basis vector is simply the unit vector in the direction extending from the origin to the IMU location on the instrument surface, and the *x* and *y* basis vectors lie tangent to the surface. For an IMU placed on a spherical instrument centered at the origin and with location $\boldsymbol{x_{IMU}}$, a set of basis vectors $\boldsymbol{x_{bas}}$, $\boldsymbol{y_{bas}}$, and $\boldsymbol{z_{bas}}$ may be calculated as follows:

$$\boldsymbol{z_{bas}} = \frac{\boldsymbol{x_{IMU}}}{\|\boldsymbol{x_{IMU}}\|} = [z_{bas_x} \quad z_{bas_y} \quad z_{bas_z}]^T$$

$$\boldsymbol{y_{bas}} = \left[\sqrt{\frac{\left(\frac{z_{bas_y}}{z_{bas_x}}\right)^2}{1+\left(\frac{z_{bas_y}}{z_{bas_x}}\right)^2}} \quad \sqrt{\frac{1}{1+\left(\frac{z_{bas_y}}{z_{bas_x}}\right)^2}} \quad 0 \right]^T = [y_{bas_x} \quad y_{bas_y} \quad y_{bas_z}]^T$$

$$\boldsymbol{x_{bas}} = \begin{bmatrix} y_{bas_y}z_{bas_z} - y_{bas_z}z_{bas_y} \\ y_{bas_z}z_{bas_x} - y_{bas_x}z_{bas_z} \\ y_{bas_x}z_{bas_y} - y_{bas_y}z_{bas_x} \end{bmatrix}^T \div \left\| \begin{bmatrix} y_{bas_y}z_{bas_z} - y_{bas_z}z_{bas_y} \\ y_{bas_z}z_{bas_x} - y_{bas_x}z_{bas_z} \\ y_{bas_x}z_{bas_y} - y_{bas_y}z_{bas_x} \end{bmatrix}^T \right\| = [x_{bas_x} \quad x_{bas_y} \quad x_{bas_z}]^T$$

Figure 3-2 shows a generated instrument with spherical shape and six IMUs randomly placed on its surface. The IMU locations are shown in blue, with basis vectors in green

extending from each location. The red center line extends from the origin to the top of the instrument, and is intended to make instrument rotations clear in the demonstrated instrument trajectory shown in Figure 3-3. In Figure 3-3, the instrument shown in Figure 3-2 follows the sample trajectory described in Section 3.2.

Although the simulation currently uses only a spherical measurement instrument, the overall simulation is robust and will accept any instrument shape. The only requirement for a new instrument shape design is that it specifies the X, Y, and Z coordinates of the shape centered at the origin, and provides IMU locations and initial IMU orientations.

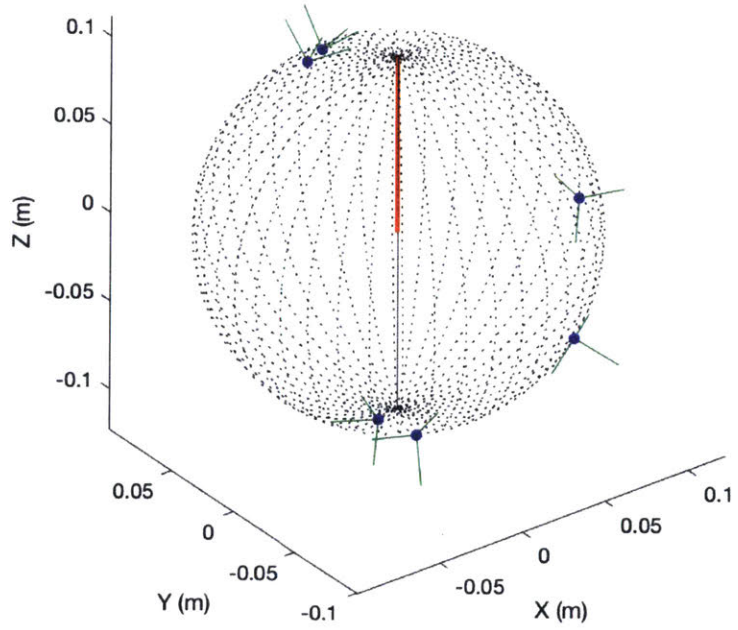


Figure 3-4: Instrument Generation

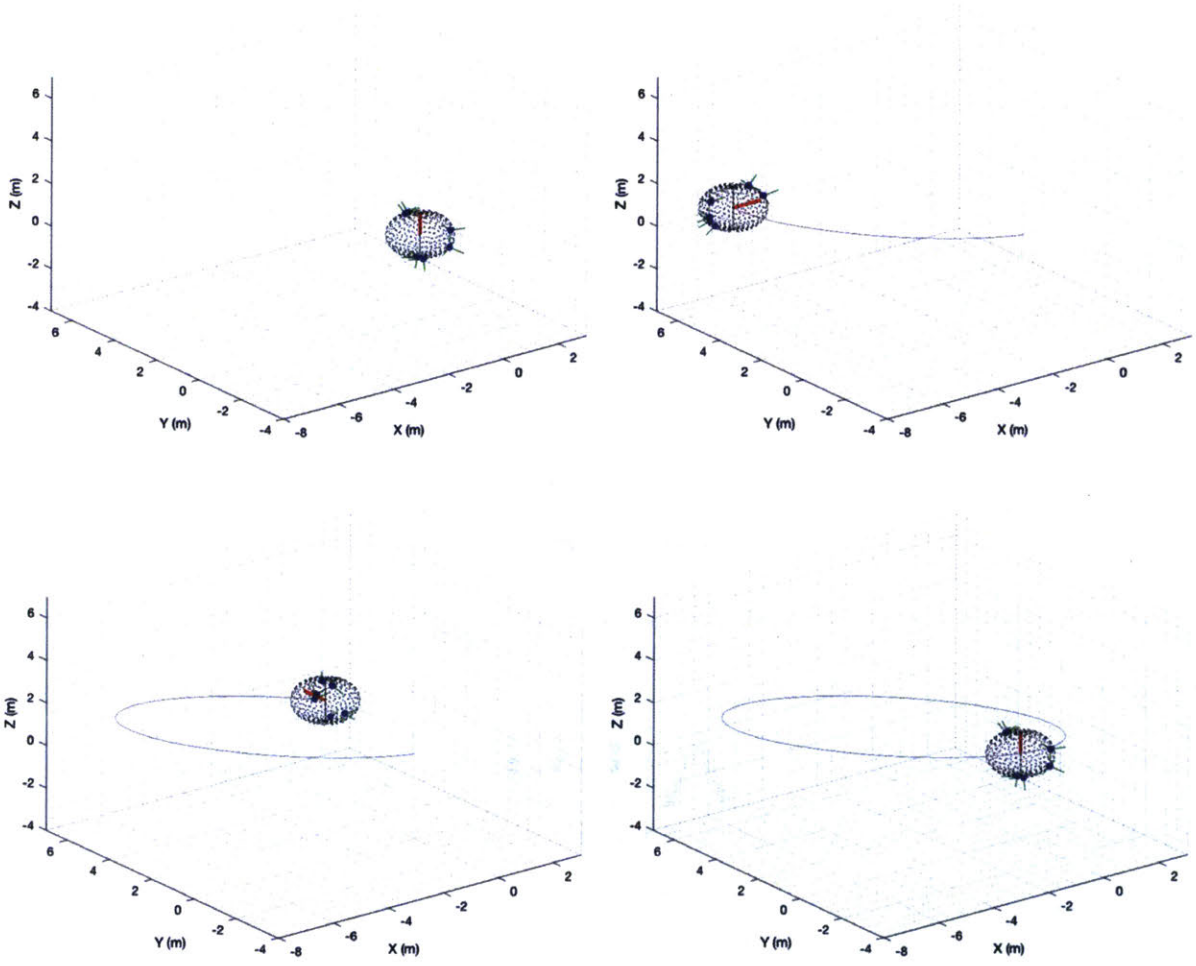


Figure 3-5: Instrument Motion and Rotation Demonstration

3.4 Three-Dimensional Sample Geometries

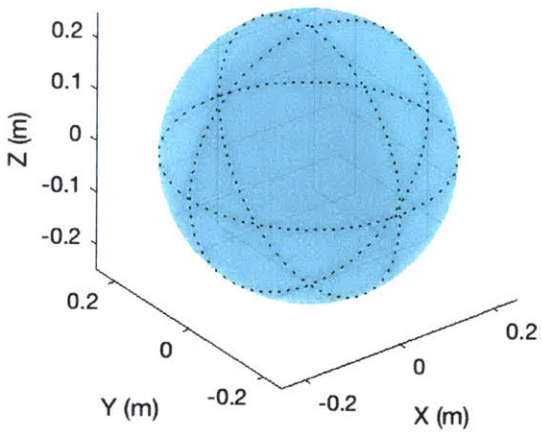
In order for the simulation to realistically represent a measurement process for three-dimensional geometries, randomized trajectories were generated for a number of geometric shapes. These trajectories were used to evaluate the overall measurement method and the success of the multiple-IMU motion processing correction routines for their intended purpose.

3.4.1 Shape Options and Demonstrative Purposes

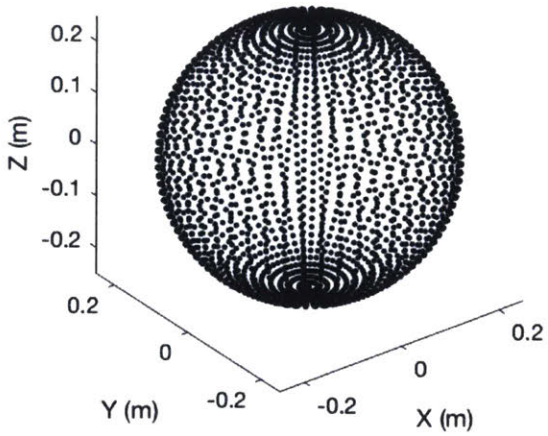
I built four sample geometries into the code, each with a specific purpose in testing functionality, limitations, and performance of surface motion tracking. The simulation contains a sphere, a cube, a hemiellipsoid, and a biologically-inspired irregular shape.

The sphere and cube were chosen to reflect the two extremes of shape measurement difficulty. As spheres are continuous and have the same curvature throughout, they provide the easiest shape for measurement. The cube, however, has multiple right-angles, which pose a greater challenge. The exact test geometries used in the simulation are shown in Figure 3-5.

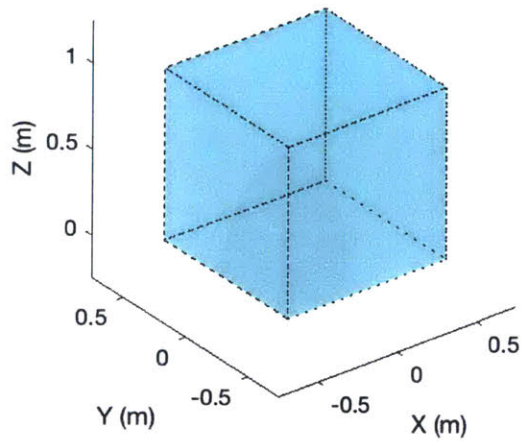
Since the intended purpose of the motion-processing imaging method is the measurement of residual limbs, the other two geometries were chosen to reflect this task. The hemiellipsoid is the simplest shape which resembles a residual limb, and therefore was chosen as a test geometry. However, in order to achieve results from the simulation as close as possible to the actual measurement scenario, a biologically-inspired irregular geometry was also generated to represent a shape similar to that of a residual limb. Both the hemiellipsoid and biologically-inspired geometries used in this simulation are shown in Figure 3-6.



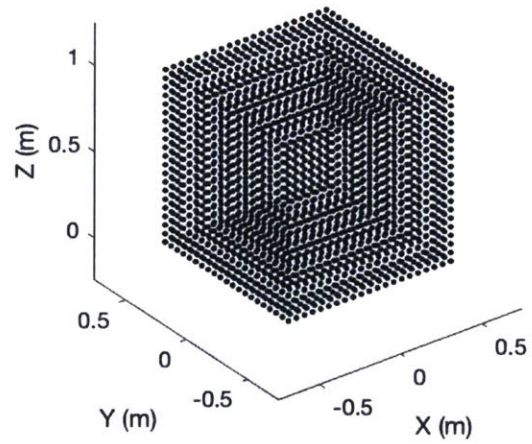
Reference Geometry



Reference Points

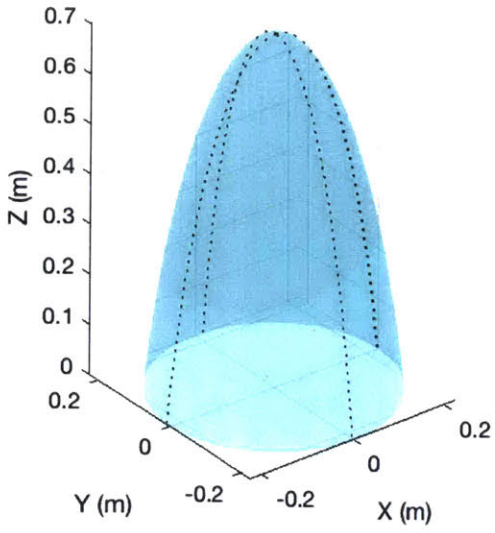


Reference Geometry

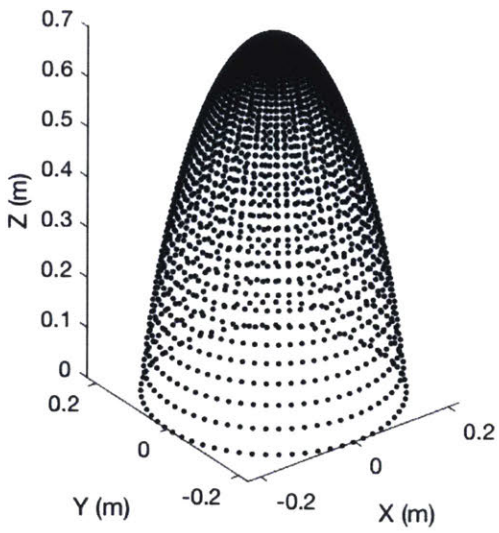


Reference Points

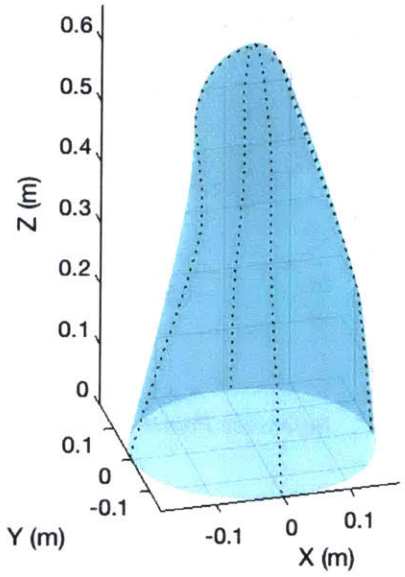
Figure 3-6: Shape Geometries (Sphere and Cube)



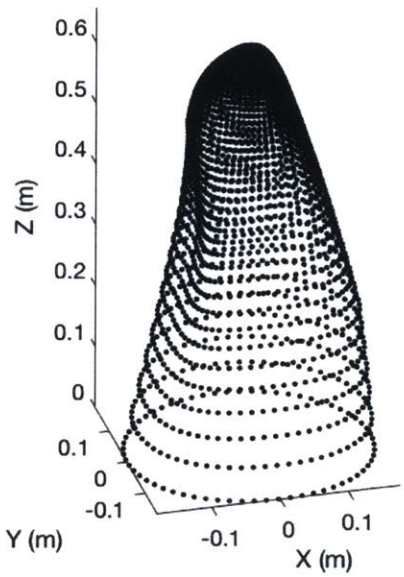
Reference Geometry



Reference Points



Reference Geometry



Reference Points

Figure 3-7: Shape Geometries (Hemiellipsoid and Biologically-Inspired Geometry)

3.4.2 Measurement Path Generation Method

The proposed measurement method for residual limb imaging involves a moving a measurement instrument in a light rubbing motion around the shape of a limb, gathering IMU data over time, which will be converted into a three-dimensional geometry. In order for the simulation to accurately reflect the measurement process, I designed a randomized path generation method to gather measurement data for a given shape geometry and over set period of time. In the case of simply generating the motion path, the position is a single point instead of the whole measurement instrument; however, this trajectory is the path that the instrument will follow in the overall measurement simulation.

In order to generate the randomized measurement motion path over a set period of time, the position is first set at an initial location on the geometry shape. A random time sub-interval is chosen, between 0 seconds and 1 second in duration, and a random direction of motion tangent to the surface is calculated. The path generation method then creates a curved trajectory for the IMU following the geometric surface in the specified direction.

The path generation method creates the curve by selecting an odd number N total points spaced a given distance apart, centered at the initial point for the trajectory and extending both forwards and backwards in the specified direction. The method determines the points on the geometric surface closest to these generated points, such that the set of points defines a curve wrapping around the geometric surface. These points are stored in vector $\boldsymbol{x}_{curve}$, $\boldsymbol{y}_{curve}$, and $\boldsymbol{z}_{curve}$, each of length n.

Consider first the case of solving for the trajectory curve in the x direction. An effective time vector $\boldsymbol{t}$ of length n is constructed with each point corresponding to a point in $\boldsymbol{x}_{curve}$. The

time vector is centered at 0, with equal spacing between the times extending in the positive and negative directions, whose values are determined by the step size and trajectory velocity.

The x position at a given time is described by a Lagrange polynomial of the form:

$$x(t) = \sum_{i=1}^{n} a_n \, t^n$$

The vector coefficients $\boldsymbol{a}$ (of length n) may be found by solving the following system:

$$\begin{bmatrix} t_0^n & t_0^{n-1} & \cdots & t_0 & 1 \\ t_1^n & t_1^{n-1} & \cdots & t_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{n-1}^n & t_{n-1}^{n-1} & \cdots & t_{n-1} & 1 \\ t_n^n & t_n^{n-1} & \cdots & t_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} x_{curve_n} \\ x_{curve_{n-1}} \\ \vdots \\ x_{curve_1} \\ x_{curve_0} \end{bmatrix}$$

The Lagrange polynomial is then used to update the sensor location point.

The y and z locations are updated according to the same method. This process repeats, generating new curves in the specified direction for x, y, and z at each time step until the end of the time sub-interval. At this point, the direction switches to a new randomized vector, and the process repeats itself. This routine continues until the predetermined maximum time, at which point the measurement and corresponding trajectory finishes.

A position trajectory for the measurement curve tracing the spherical geometry depicted in Figure 3-5 is shown in Figure 3-7. The measurement path generation method maintains constant orientation of the instrument throughout the measurement process, but could be easily modified to include arbitrary orientation motion to further demonstrate the viability of the measurement process.

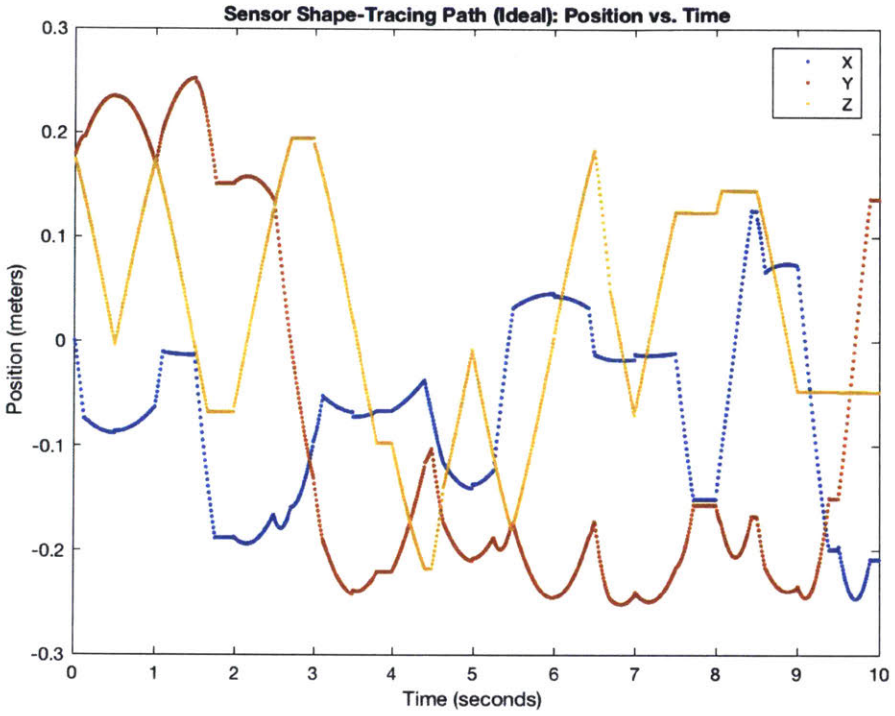


Figure 3-8: Position vs. Time Path for Sphere Geometry Measurement

There is a direct correlation between the time span of the measurement process and the accuracy of the geometry generated by the measurement routine. Figure 3-8 depicts the measurement path for the spherical geometry and resulting triangulated geometry for a series of increasing time spans. Note that this figure represents the ideal case; the geometries produced in the figure are a product of simply the generated trajectory, and have not been converted to IMU data and passed through the motion processing and correction methods.

A similar demonstration from shape measurement trajectory and geometry reconstruction for the three additional sample geometries is provided in Appendix A.

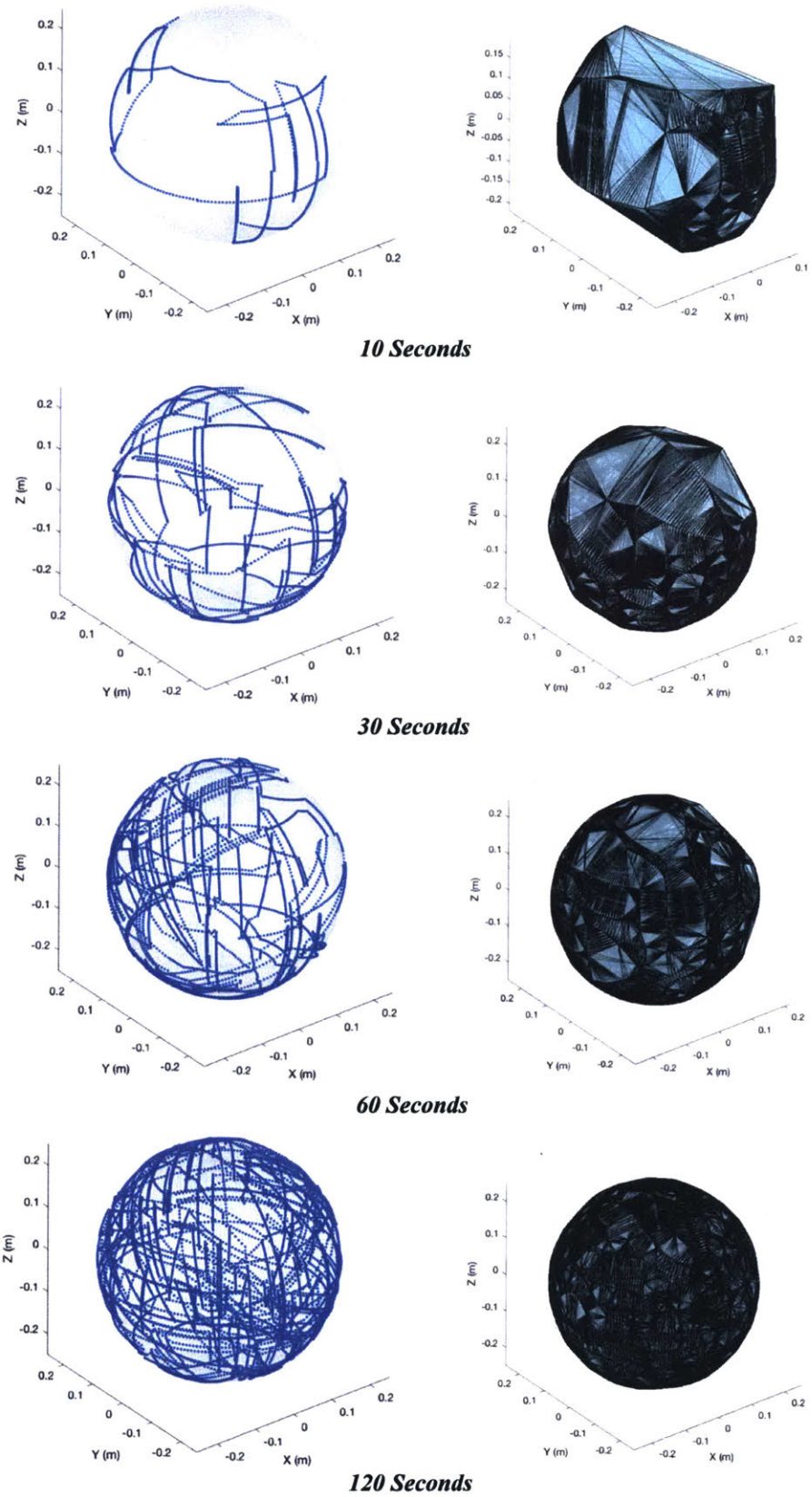


Figure 3-9: Sphere Geometry Ideal Measurement Results for Various Time Lengths

4 Evaluation of Simulation Results

Performance evaluation was conducted in two stages. First, the two motion processing correction methods were compared at a variety of simulation settings using the basic test trajectory to evaluate the functionality and limitations of each, and to determine which was more appropriate for a realistic geometry measurement. Second, the full simulation was used to generate a motion path for a measurement instrument surveying a chosen geometry, generate the instrument and simulated data for all IMUs, and motion processing and correction were applied to calculate the instrument trajectory in order to produce a final shape geometry. The full simulation provided insight on the viability of the overall measurement process.

4.1 Evaluation of Trajectory Correction Methods

In order to evaluate the success of the motion processing methods and trajectory correction, I produced and processed and simulated data across a variety of simulation parameters, and compared the results at each set of parameters using the two error correction methods.

The methods were both tested without the use of calibration in order to dramatically demonstrate the effects of each correction strategy. Please note that as a result, the motion processing results shown in Figures 4-1 through 4-4 are not representative of the capabilities of the motion processing method; the calculated trajectories depicted in these plots are expected to demonstrate significantly lower accuracy than the standard case.

The simulation parameters used in this evaluation are given in Table 4-1. In addition to the specified parameters, the two methods were tested at error levels 1 and 10, and using instruments with 4 and 10 IMUs.

Input Parameter	Value	Units
Simulation Time	5	seconds
Sampling Rate	100	Hz
ADC Length	32	bits
Accelerometer Range	2	g
Gyroscope Range	250	°/s
Instrument Shape	sphere	N/A
Error Type	all	N/A

Table 4-1: Simulation Input Parameters for Motion Processing Correction Evaluation

The results demonstrated in Figure 4-1 through 4-4 demonstrate that motion processing using the designed trajectory correction methods is significantly more accurate than motion processing without trajectory correction. Since the system is entirely uncalibrated for these examples, deviation of the measured trajectory from the actual trajectory is significant; however, in the cases in which correction is applied, the behavior of the calculated trajectory is notably closer to the actual trajectory behavior, even when the system still exhibits large discrepancies due to the lack of calibration.

In general, the instrument shape correction method performed better in systems with many IMUs and which exhibited large error. The averaging method exhibited slightly higher performance in small-error systems with fewer IMUs. For high-accuracy motion processing using a high number of IMUs (in the range of ten or more), I recommend the instrument shape correction method.

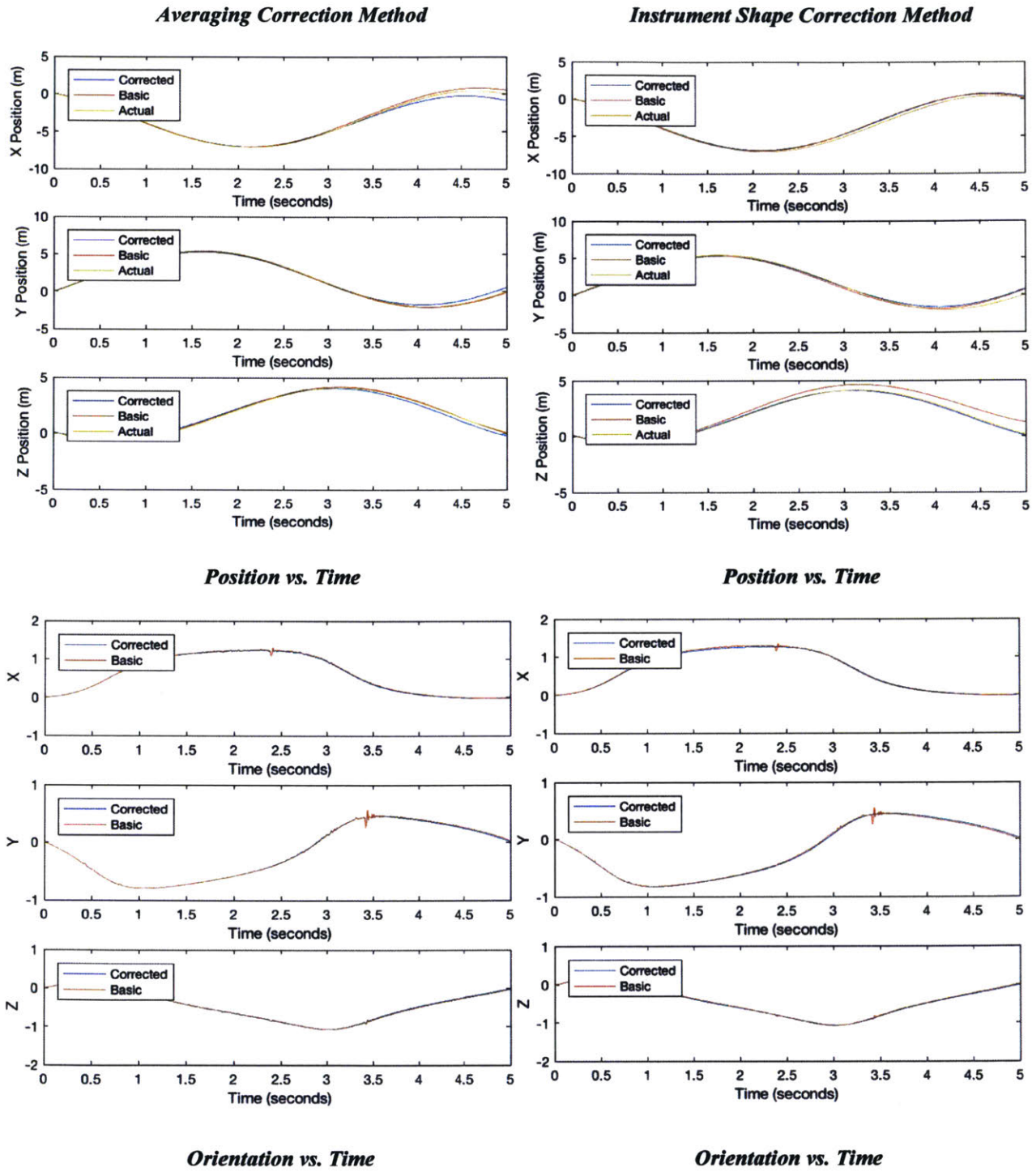


Figure 4-1: Standard Test Path Simulation (10 IMUs, Error Level 1, no calibration)

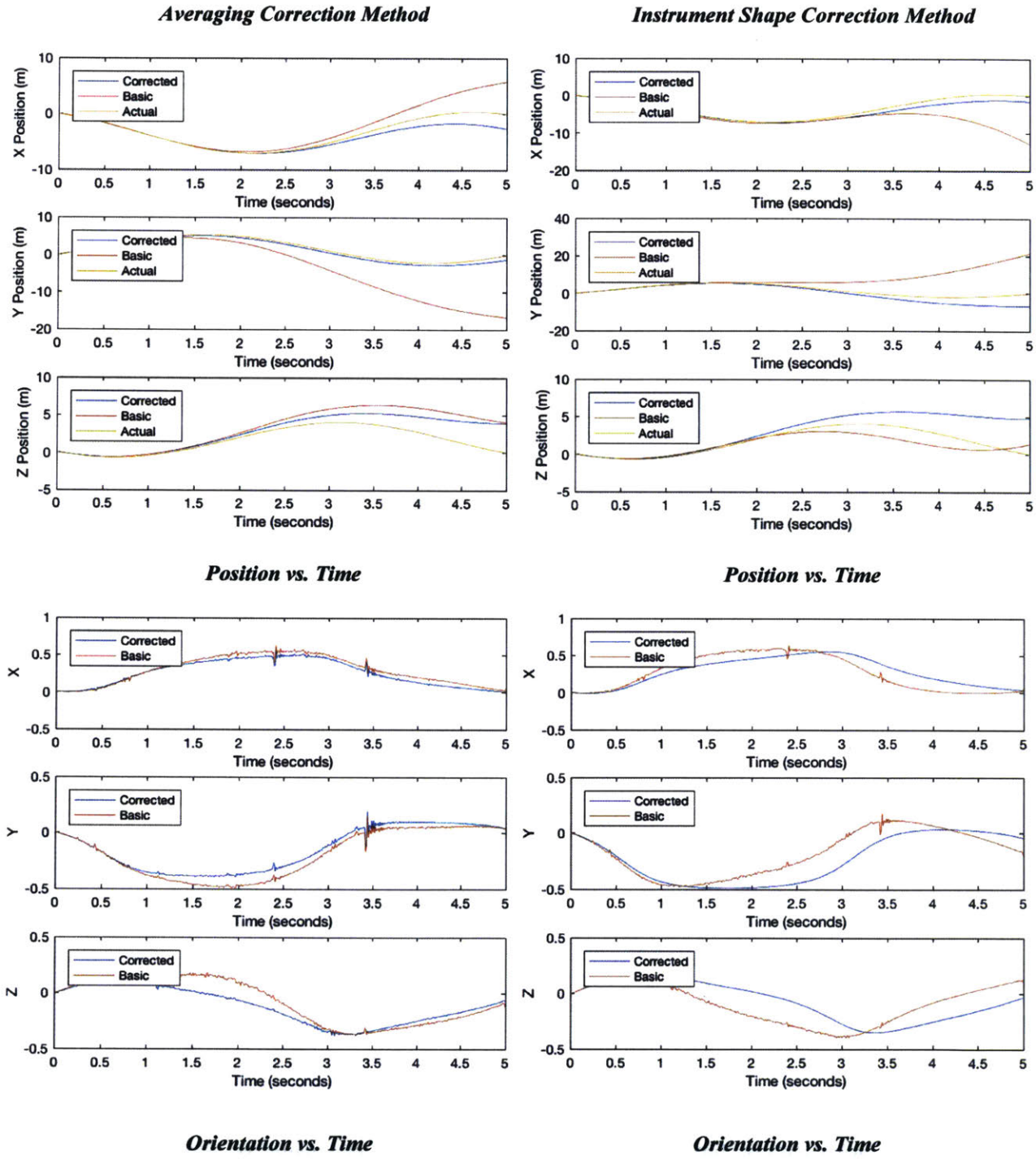


Figure 4-2: Standard Test Path Simulation (10 IMUs, Error Level 10, no calibration)

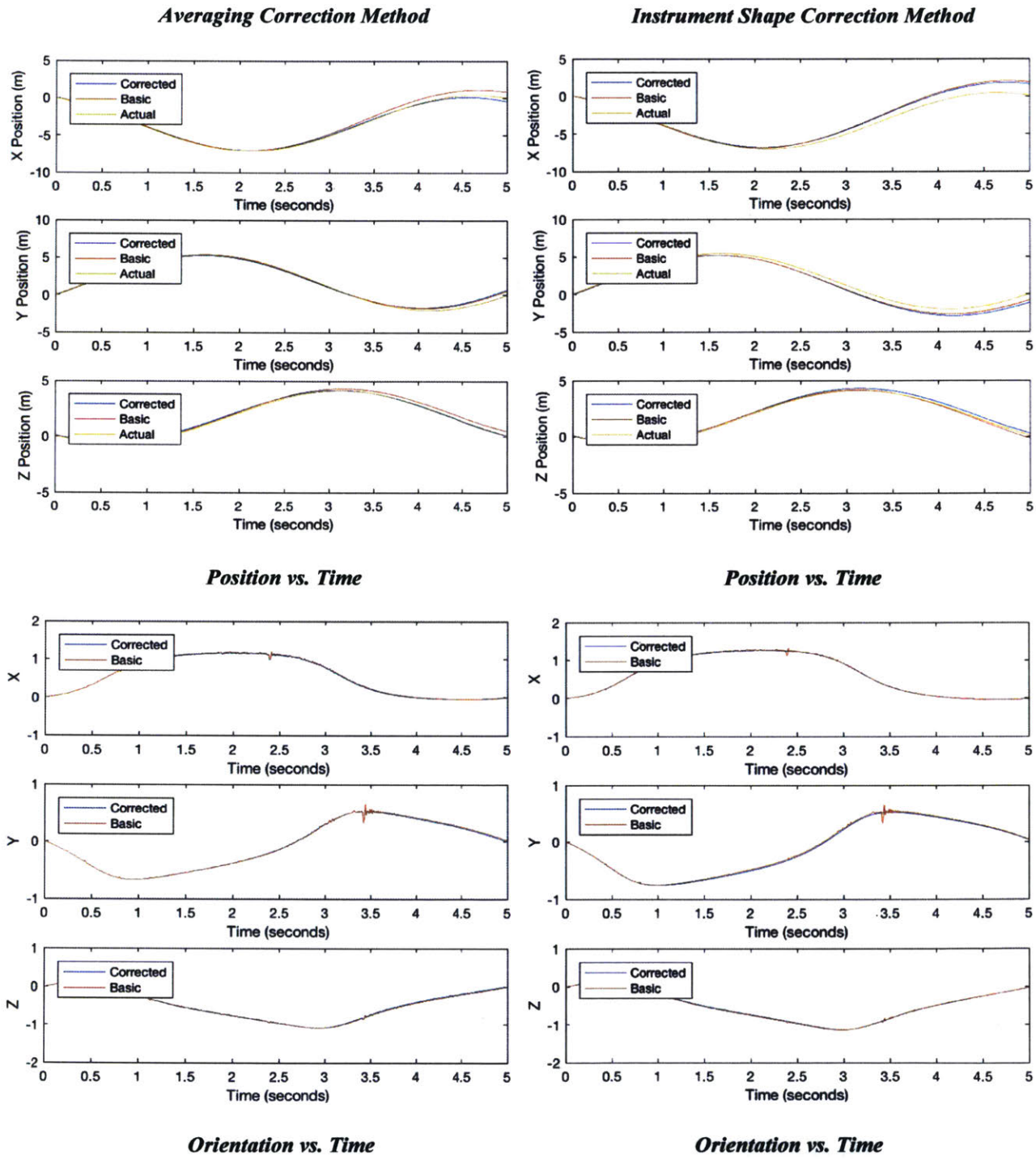


Figure 4-3: Standard Test Path Simulation (4 IMUs, Error Level 1, no calibration)

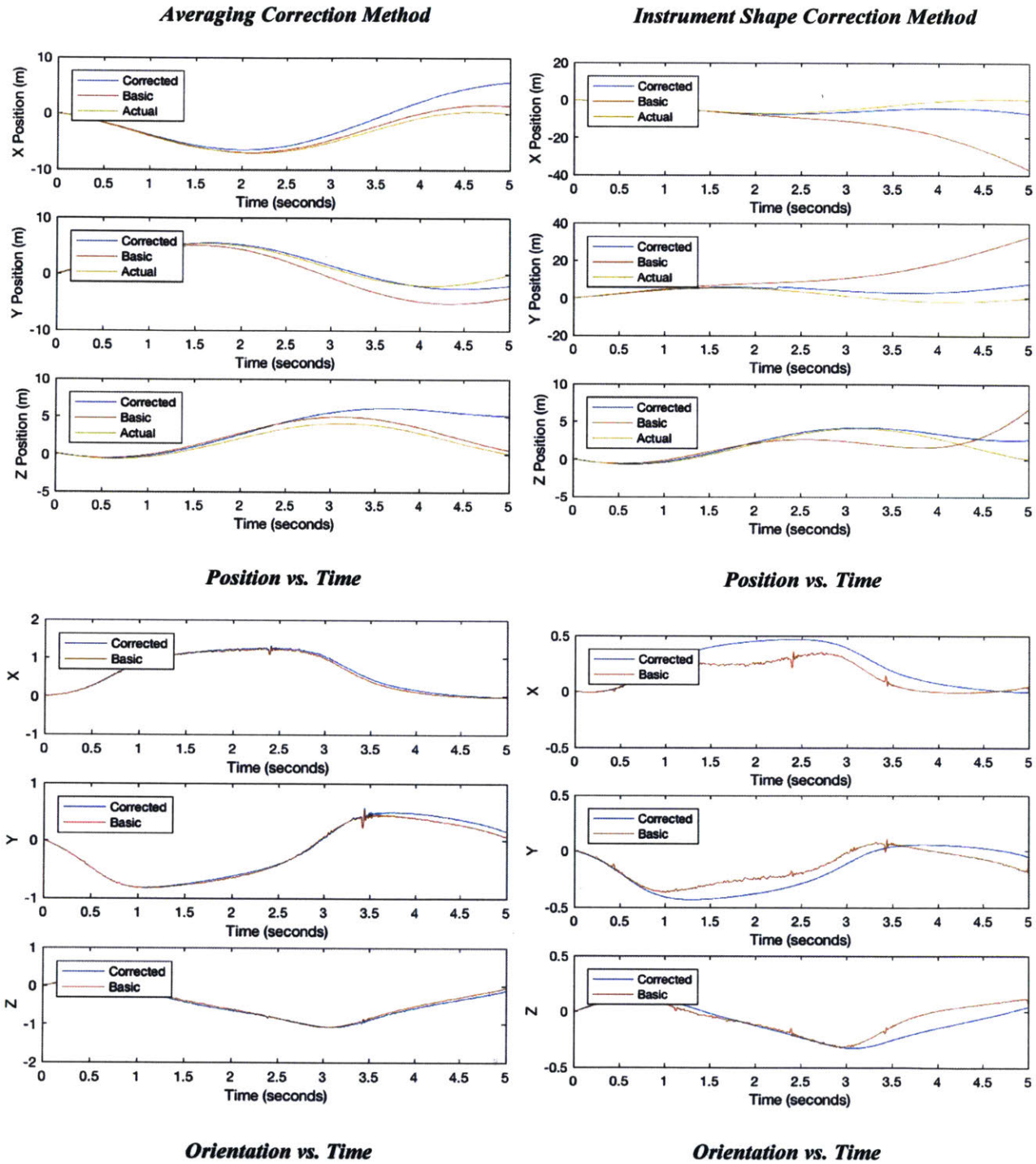


Figure 4-4: Standard Test Path Simulation (4 IMUs, Error Level 10, no calibration)

In order to confirm the viability of the motion processing and correction methods, the following test was conducted, simulating data following the position trajectory shown in Figure 3-3, scaled to a diameter of 0.2 meters. Simulated data was collected at a rate of 1 kHz with an IMU error level of 0.1 (indicating an IMU of very high accuracy), with an instrument consisting of ten calibrated IMUs randomly positioned on the surface of a sphere. Millimeter-level accuracy was achieved for the first 5 seconds of motion, with unacceptable levels of drift starting to accumulate towards 10 seconds; the error over time is shown in Figure 4-5 below. The results of this test demonstrate that in ideal conditions and over short periods of time, this method may be acceptable for imaging.

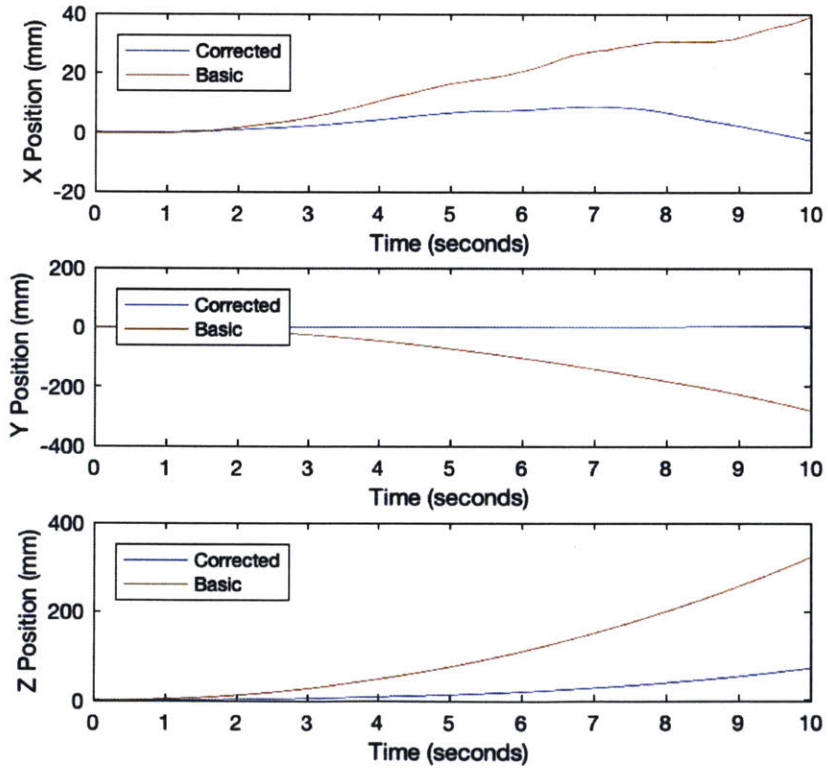


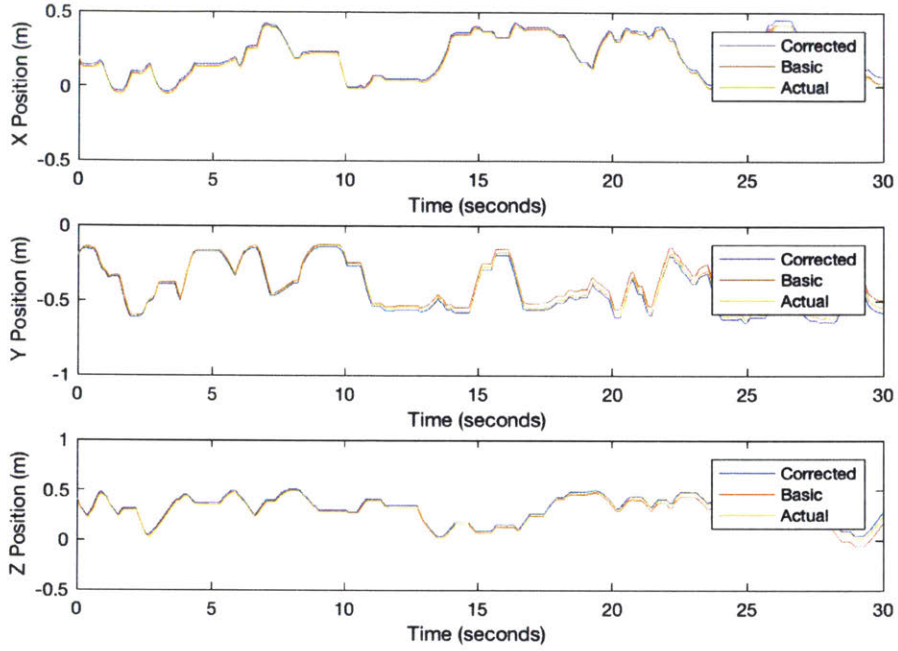
Figure 4-5: Motion Processing and Correction Methods Accuracy Test

4.2 Evaluation of Geometry Reconstruction

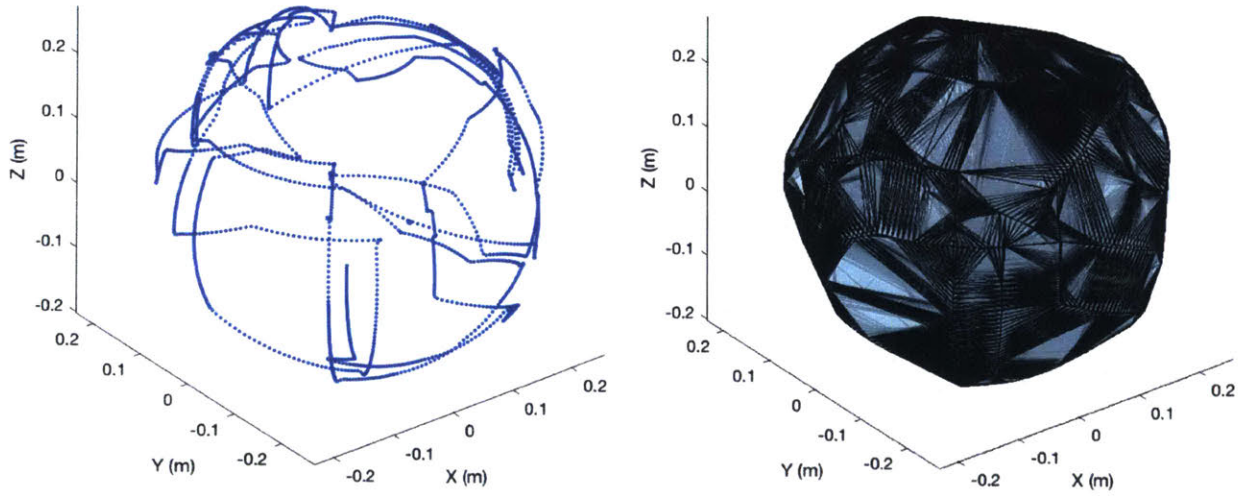
Geometry reconstruction was attempted using the simulated measurement paths described in Section 3.4. The simulation parameters used in the evaluation are outlined in Table 3-2, with a noise only error applied to mimic perfect calibration conditions, and the error level was set to 1 to represent a high-accuracy IMU. The simulation time was set to 30 seconds to allow comparison between the motion processing results and the ideal geometry reproduction (from only the simulated trajectory, without IMU motion processing) shown in Figure 3-9. A simulation time of 30 seconds was also selected because it provides a good measurement of shape progress while not allowing significant error to accumulate. For measurement of a physical system with a necessary error bound on the order of millimeters, as in the case of measurement of residual limbs, higher accuracy may be achieved using a series of short measurements (for example, in the range of five to ten seconds).

Input Parameter	**Value**	**Units**
Simulation Time	30	seconds
Sampling Rate	100	Hz
Number of IMUs	10	N/A
ADC Length	32	bits
Accelerometer Range	2	g
Gyroscope Range	250	°/s
Instrument Shape	sphere	N/A
Correction Method	averaging	N/A
Error Level	1	N/A
Error Type	noise only	N/A

Table 4-2: Simulation Input Parameters for Geometry Reconstruction Evaluation



Corrected IMU Position Calculation



Geometry Reconstruction

Figure 4-6: Sphere Geometry Reconstruction Using Averaging Correction Method

The error demonstrated in the geometry reconstruction shown in Figure 4-6 is too large to be ideal for residual limb geometry measurement; however, this is due in part to the nature of the simulated trajectory (an example of which is shown in Figure 3-8). Please see Section 4.1 and Figure 4-5 for a demonstration of the performance of the motion processing correction methods used in this simulation.

In general, however, the results of the simulation suggest that this is a viable measurement method for residual limb geometry, given very high quality IMUs with excellent calibration.

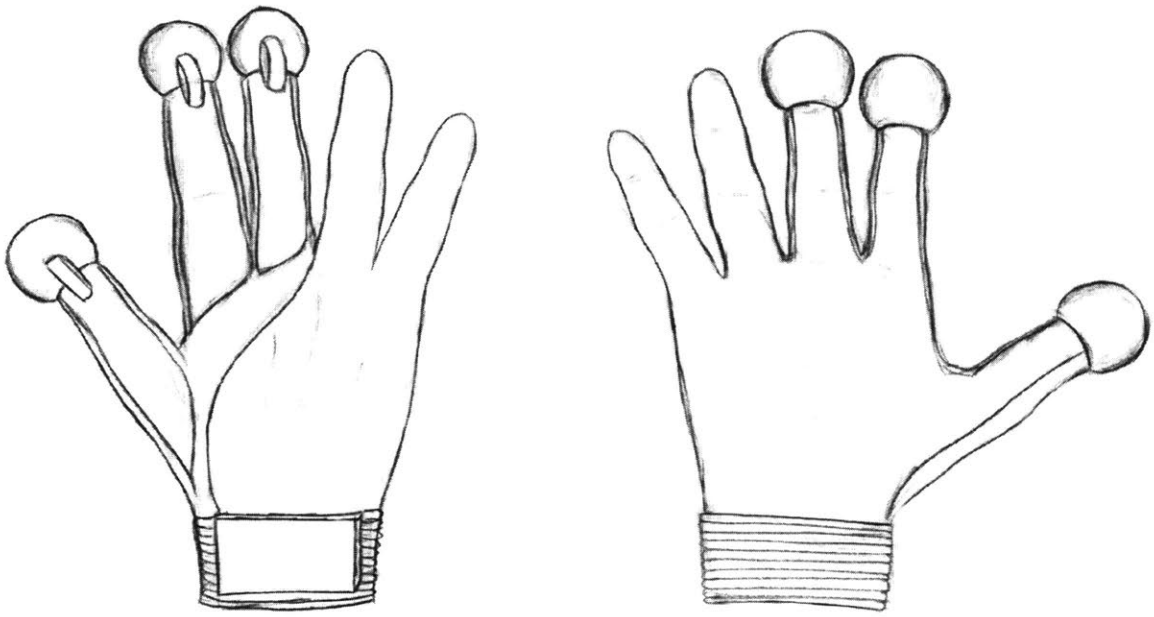
5 Summary and Conclusion

A simulation was designed to test the viability of measuring residual limb geometry using motion processing with inertial measurement unit (IMU) sensors and to provide insight on instrument design. The simulation generated a measurement instrument consisting of randomly spaced IMUs on a three-dimensional shape. Each IMU was generated using a virtual six degree of freedom IMU which incorporated realistic sources of error based on variability in commercial sensors, and used these errors and a given trajectory to produce simulated accelerometer and gyroscope data. Two categories of trajectory were used in this simulation: a regular curve with simple sinusoidal motion in each direction, and a randomized trajectory mimicking the physical measurement process of surveying a three-dimensional shape (in this case, a residual limb) using a light rubbing motion over the shape's surface. The simulation then applied motion processing methods to the simulated data, and used correction routines coordinating the data from multiple

IMUs on the surface of the instrument, using trajectory averaging and correction according to the relative positions and orientation of the IMUs.

The trajectory correction methods in combination with the motion processing routine used in this simulation (which are included in the MATLAB code given in Appendix C, and described in detail in Chapter 2) demonstrated high accuracy over short periods of time with the simple sinusoidal motion path, demonstrating accuracy on the order of millimeters for very high quality IMUs up to five seconds, with unacceptable levels of drift accumulating towards ten seconds. The motion processing method for the randomized shape-measuring simulated trajectory, while demonstrating general performance similar to that seen in the ideal measurement case (using perfect IMUs), did not demonstrate this level of accuracy; however, this is most likely due to the high prevalence of sharp corner turns in the simulated path. As such, during actual measurement, I recommend prioritizing smooth motion while measuring a residual limb.

From the results of this simulation, I have proposed a design for a measurement glove, shown in Figure 5-1. The thumb, index finger, and middle finger are each capped with a sphere containing 10 IMUs, dispersed evenly over an inner sphere. Flexible electronics casing runs down the side of each finger and traces the back of the hand, leading to an inflexible electronics casing at the back of the wrist, on top of the elastic cuff of the glove. Behind each finger is a small ridge which will be used to guide the spheres at each finger to tracks on a known, grounded base to reset every five to ten seconds in order to ensure continued accuracy in measurement.

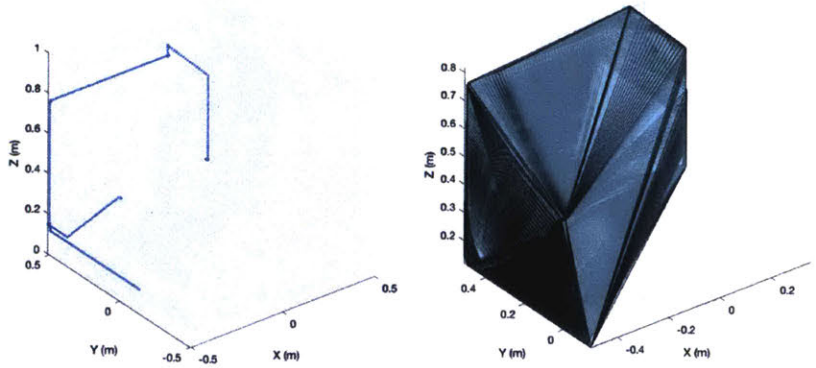


Back ***Front***

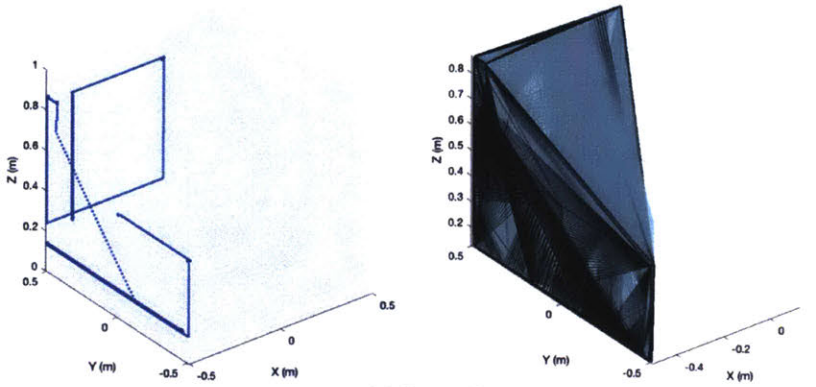
Figure 5-1: Proposed Measurement Instrument Design

A continuation of this project should involve the physical implementation of the proposed coordinated IMU trajectory correction methods in order to ensure measurement accuracy at the desired order, and eventual implementation of these processes into a measurement glove for ease and accuracy in imaging of residual IMUs.

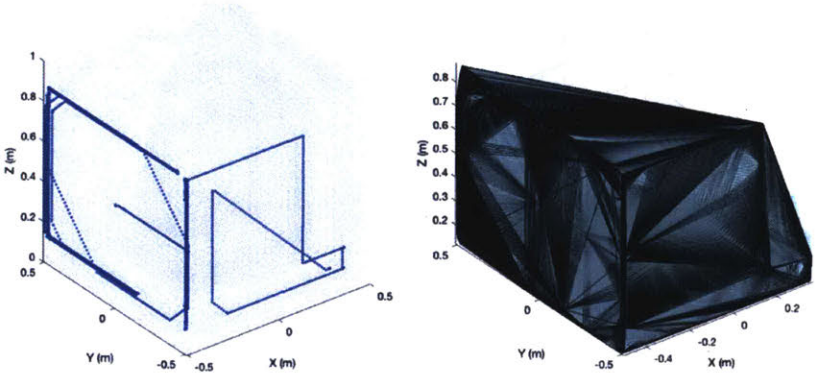
Appendix A: Additional Geometry Ideal Measurement Results



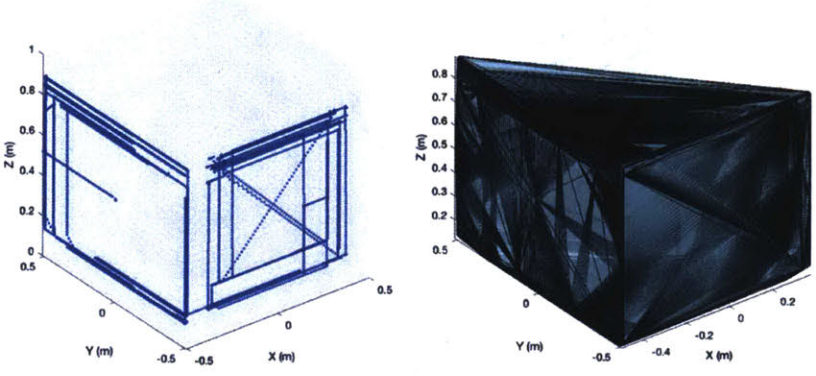
10 Seconds



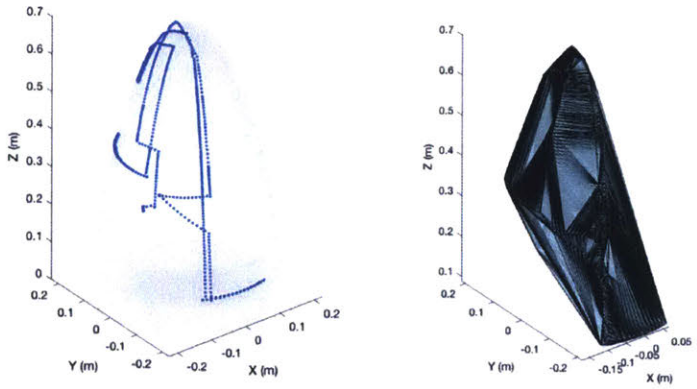
30 Seconds



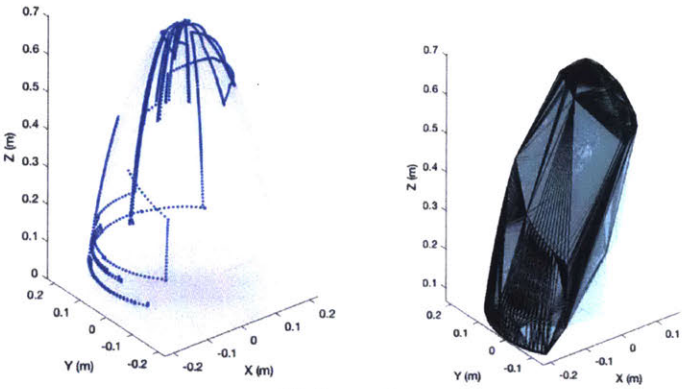
60 Seconds



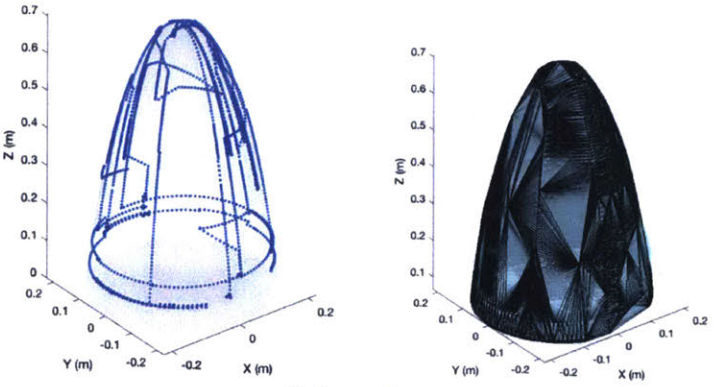
120 Seconds



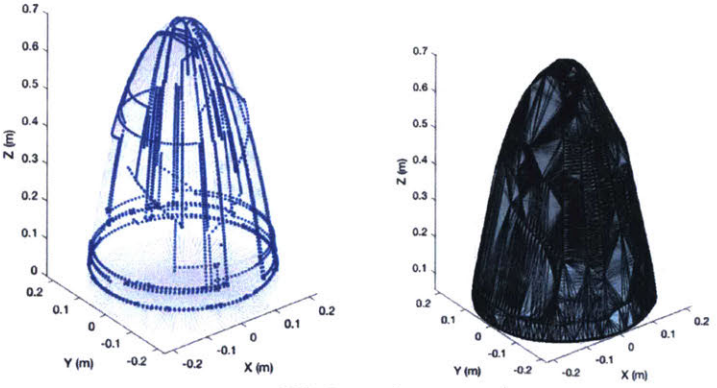
10 Seconds



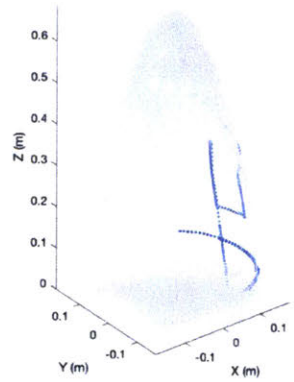
30 Seconds

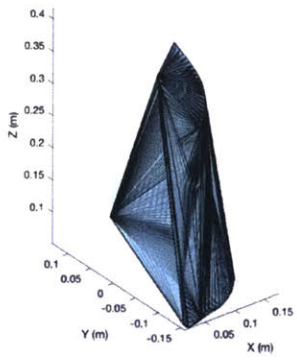


60 Seconds

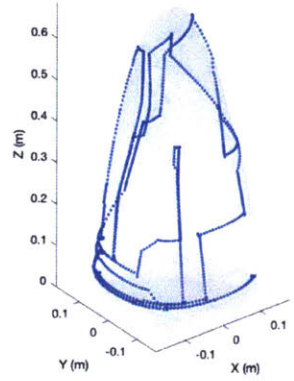


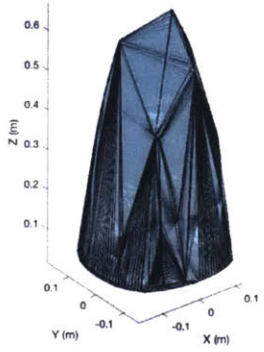
120 Seconds



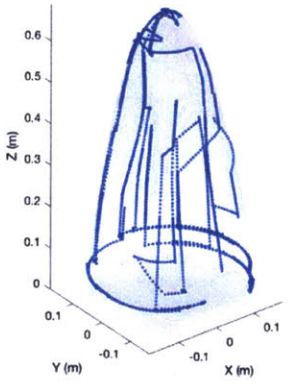


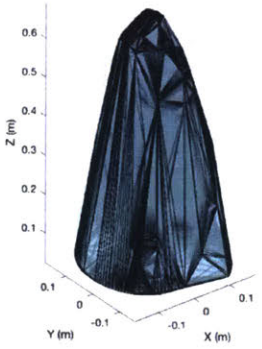
10 Seconds



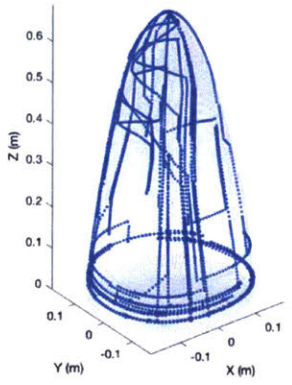


30 Seconds



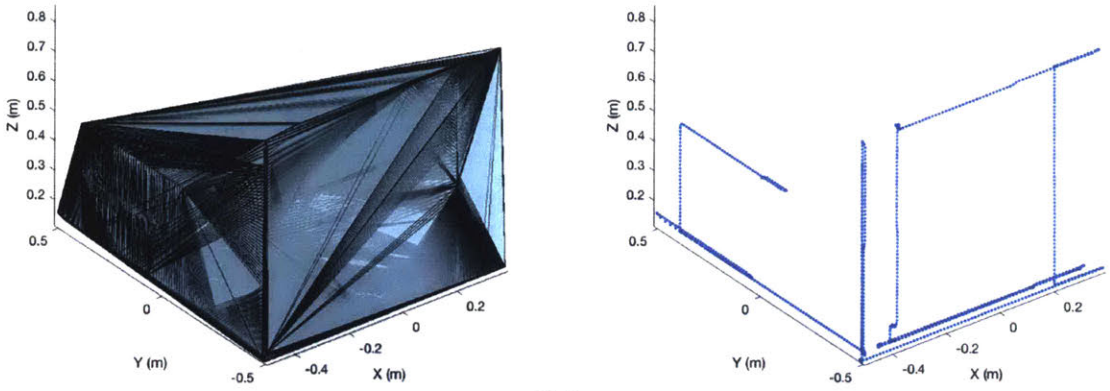


60 Seconds

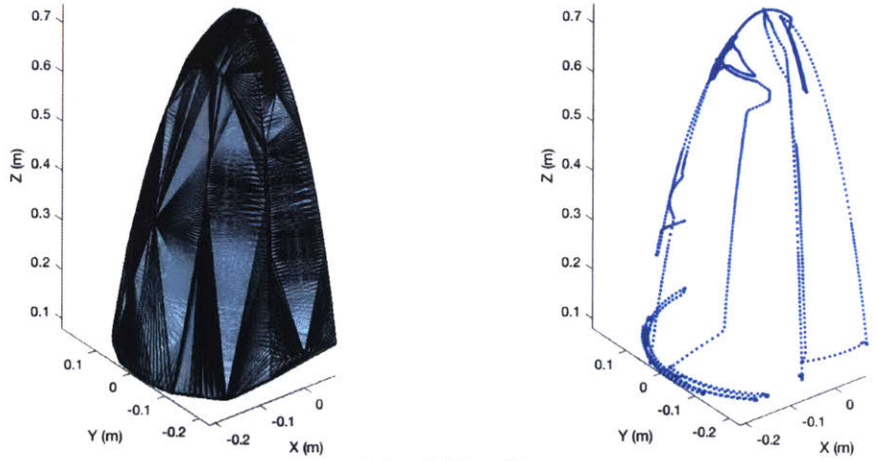


120 Seconds

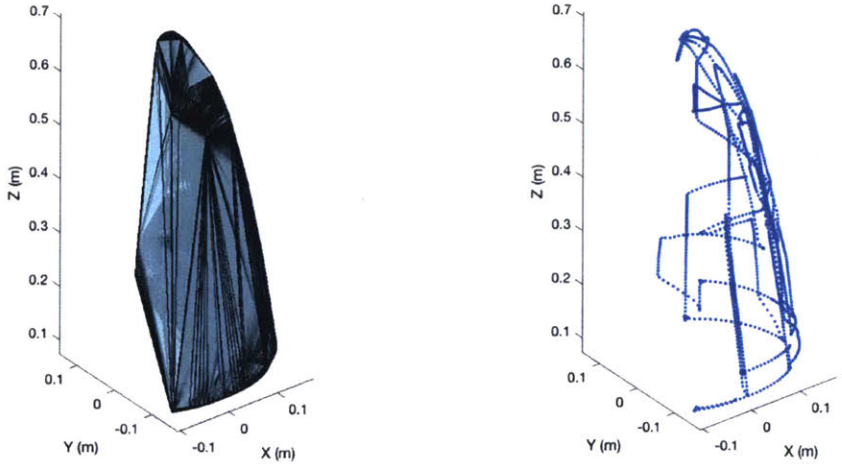
Appendix B: Additional Geometry Reconstruction



Cube



Hemiellipsoid



Biologically-Inspired Geometry

Appendix C: MATLAB Simulation Code

main.m

```
clear; close all;

% Define parameters
sim_time = 10;  % seconds
sampling_rate = 100;    % Hz
imu_num = 10;
ADC_length = 32;    % bits
A_range = 2;
G_range = 250;
geom_shape = 'sphere';   % options: 'sphere', 'cube', 'hemiellipsoid',
'irregular'
instrument_shape = 'sphere';    % options: 'sphere'
correction_method = 'instrument_shape';    % options:
'averaging','instrument_shape','none'
error_level = 0.1;    % eps lowest for instrument_shape correction method
error_type = 'all';  % options:
'gyro_only','accel_only','all','gyro_noise_only','accel_noise_only','noise_on
ly')

addpath('Quaternion');

N = sim_time*sampling_rate;

A_range = A_range*[-1 1];
G_range = G_range*[-1 1];

% % Choose shape to trace
% disp('Generating motion path...');
% [p_path,q_path,ori_i_path,vel_i_path,time] = ...
%                         shape_path(geom_shape,sim_time,sampling_rate);

% Generate x y z path, orientations (baseline)
disp('Generating motion path...');
[p_path,q_path,ori_i_path,vel_i_path,time] = ...
                        circle_path(sim_time,sampling_rate);

% Choose instrument shape
disp('Generating instrument...');
[instrument_xyz,imu_pts,imu_ori] = ...
                        generate_instrument(instrument_shape,imu_num);

% Add instrument "bobbing" over time (mimicking real measurement motion)

% Initialize IMUs
disp('Initializing IMUs...');
[alignment,offset,sensitivity,noise] = imu_setup(A_range,G_range,...
                        ADC_length,error_level,time,imu_num,error_type);

% Move instrument according to baseline path and orientations
    % Collect IMU data along the way
```

```
disp('Generating simulated IMU data...');
imu_pos = repmat(imu_pts,N,1,1) + repmat(p_path,1,1,imu_num);
imu_ori_i = imu_ori + repmat(ori_i_path,1,1,imu_num);
for i = 1:imu_num
    imu_ori_i(:,1,i) = imu_ori_i(:,1,i)/norm(imu_ori_i(:,1,i));
    imu_ori_i(:,2,i) = imu_ori_i(:,2,i)/norm(imu_ori_i(:,2,i));
    imu_ori_i(:,3,i) = imu_ori_i(:,3,i)/norm(imu_ori_i(:,3,i));
end
accelerometer = ones(N,3,imu_num); gyroscope = ones(N,3,imu_num);
for i = 1:imu_num
    [accelerometer(:,:,i),gyroscope(:,:,i)] = ...
        virtualIMU(imu_pos(:,:,i),q_path,time,alignment(:,:,i),...
            offset(:,:,i),sensitivity(:,:,i),noise(:,:,i),...
            ADC_length,imu_ori_i(:,:,i));
end
%
% accelerometer(1,:,:) = accelerometer(2,:,:);
% gyroscope(1,:,:) = gyroscope(2,:,:);

% % Filter raw data
% [B,A] = butter(4,0.9);
% accelerometer(10:end,:,:) = filter(B,A,accelerometer(10:end,:,:));
% gyroscope(10:end,:,:) = filter(B,A,gyroscope(10:end,:,:));

% % Assume no knowledge of IMU error parameters
% alignment = repmat(eye(3),1,2,imu_num);
% offset = zeros(3,2,imu_num);
% raw_max = 2^(ADC_length-1);
% sens = [raw_max/A_range(2) raw_max/G_range(2)];
% sens = repmat(sens,1,1,imu_num);

% Motion processing and correction
disp('Motion processing...');
pos = ones(N,3,imu_num); ori = ones(3,3,N,imu_num);

for i = 1:imu_num
    ori(:,:,:,i) = ori_basic_calc(gyroscope(:,:,i),...
        time,alignment(:,:,i),offset(:,:,i),sensitivity(:,:,i),...
        imu_ori_i(:,:,i));
end
ori_mod = ori_correct(ori,instrument_xyz,imu_ori_i,...
            correction_method,gyroscope,time,alignment,...
            offset,sensitivity);

for i = 1:imu_num
    pos(:,:,i) = pos_basic_calc(ori(:,:,:,i),accelerometer(:,:,i),time,...
                    alignment(:,:,i),offset(:,:,i),sensitivity(:,:,i),...
                    imu_pos(1,:,i),vel_i_path(1,:));
end
pos_mod = pos_correct(pos,instrument_xyz,imu_pts,correction_method,...

ori,accelerometer,time,alignment,offset,sensitivity,imu_pos(1,:,:),vel_i_path
(1,:));
pos_mod(1,:,:) = pos_mod(2,:,:);
pos_mod = pos_mod - pos_mod(1,:,:) + imu_pos(1,:,:);

ori_instr = mean(ori_mod-repmat(permute(imu_ori,[1 2 4 3]),1,1,N,1),4);
```

```
ori_instr = ori_instr + ori_i_path;
for i = 1:3
    ori_instr(:,i,:) = ori_instr(:,i,:) ./ ...
        sqrt(ori_instr(1,i,:).^2+ori_instr(2,i,:).^2+ori_instr(3,i,:).^2);
end

% assume initial positions and orientations known
pos_instr = mean(pos_mod-repmat(imu_pos(1,:,:),N,1,1),3) + p_path(1,:);
pos_instr = mean(pos_mod,3)+p_path(1,:);

% New instrument shapes require design of new trajectory correction methods
if strcmp(instrument_shape,'sphere')
    instrument_radius = ...
                    max(instrument_xyz(:,1))-min(instrument_xyz(:,1))/2;
    for j = 1:length(pos_instr)
        pos_instr(j,:) = pos_instr(j,:) - ...
                    instrument_radius*pos_instr(j,:)/norm(pos_instr(j,:));
    end
end

% Generate plots
disp('Generating plots...');
draw_plots(time,pos,ori,imu_pos,pos_instr,ori_instr,p_path,...
                                ori_mod,pos_mod,accelerometer,gyroscope);
```

---

shape_path.m

```
function [p_path,q_path,ori_i_path,vel_i_path,time] = ...
                        shape_path(geom_shape,sim_time,sampling_rate)

% INPUTS: geometry shape ('cube', 'sphere', 'hemiellipsoid' or 'irregular')
%         sim_time (seconds), sampling rate (Hz)
% OUTPUTS: p_path (xyz coordinates over time, m, Nx3)
%          q_path (iterative quaternion over time, Nx4)
%          ori_i_path (initial orientation, 3x3)
%          vel_i_path (initial velocity, m/s, 3x1)
%          time (Nx1)

NN = sim_time*sampling_rate;
dt = 1/sampling_rate;

% Set reference geometry
% Options: 'cube','sphere','hemiellipsoid','irregular'
def_shape = geom_shape;
XYZ_scale = 0.125;

[X,Y,Z] = test_surfaces(def_shape,XYZ_scale);
X = reshape(X,[],1); Y = reshape(Y,[],1); Z = reshape(Z,[],1);
XYZ = [X Y Z];

% Set starting point
start_loc = 3*round(length(XYZ)/4);
xyz_start = XYZ(start_loc,:);
x = xyz_start(1); y = xyz_start(2); z = xyz_start(3);
```

```
% Set starting velocity coefficient
V = sampling_rate/10;
% V = 200;    % ~ m/s

XYZ_center = [mean(X) mean(Y) mean(Z)];
dir = (XYZ_center-xyz_start)/norm(XYZ_center-xyz_start);

t = 0;
i = 1;
reached_object = 0;
enough_data = 0;
xyz(1,:) = xyz_start;
vel_init = ones(1,3);

% vector to object center
v_cen = (xyz(i,:)-XYZ_center)/norm(xyz(i,:)-XYZ_center);
% Calculate orthogonal vector to v_cen
dir_z = 2; dir_x = 2;
dir_y = -(dir_z*v_cen(3)+dir_x*v_cen(1))/v_cen(2);
dir = [dir_x dir_y dir_z];

while (enough_data == 0)

    t_path_init = t;
    t_path = (V*sim_time*dt)/4;
    t_path = rand;

    % vector to object center
    v_cen = (xyz(i,:)-XYZ_center)/norm(xyz(i,:)-XYZ_center);

    N = 13; spacing = 4; % N = odd; spacing = dist between pts for interp
    while t-t_path_init < t_path
        curve_XYZ = ones(N,3);
        for n = -(N-1)/2:(N-1)/2
            curve_XYZ(n+(N+1)/2,:) = ...
                closest_pt((xyz(i,:) - spacing*n*V*dt*dir), XYZ);
        end
        t_eff = (-(N-1)/2:(N-1)/2) * dt*spacing*V; % effective time for
interp pts
        exp_mat = (N:-1:0);
        Vand = t_eff'.^exp_mat; % Vandermonde matrix
        Ax = Vand\curve_XYZ(:,1);
        Ay = Vand\curve_XYZ(:,2);
        Az = Vand\curve_XYZ(:,3);
        x_step = sum(((dt*V).^exp_mat').*Ax)-xyz(i,1);
        y_step = sum(((dt*V).^exp_mat').*Ay)-xyz(i,2);
        z_step = sum(((dt*V).^exp_mat').*Az)-xyz(i,3);
        xyz(i+1,:) = closest_pt(xyz(i,:),XYZ) + V*dt*[x_step y_step z_step];

        if t == 0
            vel_init = V*[x_step y_step z_step];
        end
        i = i+1; t = t+dt;

    end
```

```
    % Calculate orthogonal vector to v_cen
    dir_z = 2*(rand-0.5); dir_x = 2*(rand-0.5);
    dir_y = -(dir_z*v_cen(3)+dir_x*v_cen(1))/v_cen(2);
    dir = [dir_x dir_y dir_z];

    if t > sim_time-dt
        enough_data=1;
        break
    end
end

if strcmp(def_shape,'cube')
    vel_init = vel_init + [0 0.166 0.166];
elseif strcmp(def_shape,'sphere')
    vel_init = vel_init + [-0.048 0.082 -.001];
elseif strcmp(def_shape,'hemiellipsoid')
    vel_init = vel_init + [-0.051 0.078 0.071];
elseif strcmp(def_shape,'irregular')
    vel_init = vel_init + [0.038 -0.054 0.002];
end

p_path = xyz(1:NN,:);
vel_i_path = vel_init;
time = 0:dt:dt*(length(p_path)-1);
ori_i_path = eye(3);
q_path = repmat([1 0 0 0],length(p_path)-1,1);

end
```

---

test_surfaces.m

```
function [x,y,z] = test_surfaces(shape,XYZ_scale)
% Defines reference geometry (goal shape)
% Possible paths: cube, sphere, hemiellipsoid, irregular
% Returns set of x,y,z points

%% CUBE
% NxNxN cube centered at 0 in xy plane, with max z at N

if strcmp(shape,'cube')
    N = 4;
    n = 0.05;
%     n = 0.2;
    N = N*XYZ_scale; n = n*XYZ_scale;
    [a,b] = meshgrid(-(N/2):n:(N/2),-(N/2):n:(N/2));
    a = reshape(a,[numel(a) 1]); b = reshape(b,[numel(b) 1]);
    x = [a a b b -N/2*ones(size(a)) N/2*ones(size(a))];
    y = [b b -N/2*ones(size(a)) N/2*ones(size(a)) a a];
    z = [zeros(size(a)) N*ones(size(a)) a+N/2 a+N/2 b+N/2 b+N/2];
%     x = x*XYZ_scale; y = y*XYZ_scale; z = z*XYZ_scale;
    A1 = [min(a) max(a) max(a) min(a)];
    A2 = [min(a) min(a) max(a) max(a)];
    A3 = min(a)*ones(1,4); A4 = max(a)*ones(1,4);
    xsurf = [A1;         A1;         A1;   A1;   A3;   A4];
    ysurf = [A2;         A2;         A3;   A4;   A2;   A2];
```

```
    zsurf = [zeros(1,4); N*ones(1,4); A2+N/2; A2+N/2; A1+N/2; A1+N/2];
    figure()
    subplot(1,2,1)
    for i = 1:6
        patch('XData',xsurf(i,:),'YData',ysurf(i,:),'ZData',zsurf(i,:), ...
            'FaceColor',[0 0.8 0.7],'FaceAlpha',0.2,'LineStyle',':')
        hold on
        view(3)
        axis([-N/2-XYZ_scale N/2+XYZ_scale -N/2-XYZ_scale ...
                    N/2+XYZ_scale -XYZ_scale N+XYZ_scale])
%         axis([-N/2-1 N/2+1 -N/2-1 N/2+1 -1 N+1])
        pbaspect([1 1 1])
    end
    title('Reference Geometry')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    subplot(1,2,2)
    plot3(x,y,z,'.');
%     plot3(x,y,z,'k.');
    axis([-N/2-XYZ_scale N/2+XYZ_scale -N/2-XYZ_scale ...
                N/2+XYZ_scale -XYZ_scale N+XYZ_scale])
%     axis([-N/2-1 N/2+1 -N/2-1 N/2+1 -1 N+1])
    pbaspect([1 1 1])
    title('Reference Points')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    suptitle('Generated Shape: Cube')
end

%% SPHERE

if strcmp(shape,'sphere')
    sphere_N = 200;
%     sphere_N = 50;
    theta = 0:0.2:2*pi;
    [x,y,z] = sphere(sphere_N);
    x = x*XYZ_scale; y = y*XYZ_scale; z = z*XYZ_scale;
    figure()
    subplot(1,2,1)
    surf(x,y,z,'EdgeColor','none','FaceColor',[0 0.8 0.7],'FaceAlpha',0.2)
    hold on
    plot3(cos(theta)*XYZ_scale,sin(theta)*XYZ_scale,...
        zeros(size(theta)),'k:')
    hold on
    plot3(zeros(size(theta)),cos(theta)*XYZ_scale,...
        sin(theta)*XYZ_scale,'k:')
    hold on
    plot3(sin(theta)*XYZ_scale,zeros(size(theta)),...
        cos(theta)*XYZ_scale,'k:')
    axis equal
    title('Reference Geometry')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    subplot(1,2,2)
    plot3(x,y,z,'.');
%     plot3(x,y,z,'k.');
    axis equal
    title('Reference Points')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    suptitle('Generated Shape: Sphere')
end
```

```
%% HEMIELLIPSOID

if strcmp(shape,'hemiellipsoid')
    circle_N = 200;
%     circle_N = 50;
    dtheta = 0.05;
%     dtheta = 0.1;
    theta = (0:dtheta:(2*pi+dtheta))';
    theta = repmat(theta,1,circle_N+1);
    T = length(theta(:,1));
    xy_scale = 0:1/circle_N:1;
    xy_scale = repmat(xy_scale,length(theta(:,1)),1);
    x = xy_scale.*cos(theta); y = xy_scale.*sin(theta);
    z_scale = 5;
    z = sqrt(1-x.^2-y.^2)*z_scale;
    z = real(z);
    x = x(:,1:length(x(1,:))-round(1/10*circle_N));
    y = y(:,1:length(y(1,:))-round(1/10*circle_N));
    z = z(:,1:length(z(1,:))-round(1/10*circle_N));
    z = z-min(z(:,end));
    x = x*XYZ_scale; y = y*XYZ_scale; z = z*XYZ_scale;
    figure()
    subplot(1,2,1)
    surf(x,y,z,'EdgeColor','none','FaceColor',[0 0.8 0.7],'FaceAlpha',0.2)
    hold on
    plot3(x(end,:),y(end,:),z(end,:),'k:')
    hold on
    plot3(x(1,:),y(1,:),z(1,:),'k:')
    hold on
    plot3(x(round(T/4),:),y(round(T/4),:),z(round(T/4),:),'k:')
    hold on
    plot3(x(round(T/2),:),y(round(T/2),:),z(round(T/2),:),'k:')
    hold on
    plot3(x(round(3*T/4),:),y(round(3*T/4),:),z(round(3*T/4),:),'k:')
    axis equal
    title('Reference Geometry')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    subplot(1,2,2)
    plot3(x,y,z,'.');
%     plot3(x,y,z,'k.');
    axis equal
    title('Reference Points')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    suptitle('Generated Shape: Hemiellipsoid')
    x = repmat(x,2,1); y = repmat(y,2,1); z = [zeros(size(z)); z];
end

%% IRREGULAR BIOLOGICALLY-INSPIRED SHAPE

if strcmp(shape,'irregular')
    N = 300;
%     N = 50;
    dtheta = 0.05;
%     dtheta = 0.1;
    theta = (0:dtheta:(2*pi+dtheta))';
    theta = repmat(theta,1,N+1);
```

```
T = length(theta(:,1));
xy_scale = 0:1/N:1;
xy_scale = repmat(xy_scale,length(theta(:,1)),1);
x = xy_scale.*cos(theta); y = xy_scale.*sin(theta);
z_scale = 4;
z = sqrt(1-x.^2-y.^2)*z_scale;
z = real(z);
for i = round(T/6):round(T/3)
    for j = round(N/2):round(7*N/8)
        x(i,j) = x(i,j)*...
            (1-0.3*(1-(abs(11*N/16-j)/abs(11*N/16-N/2))^1.5)*...
                (1-(abs(T/4-i)/abs(T/4-T/6))^1.5));
        y(i,j) = y(i,j)*...
            (1-0.3*(1-(abs(11*N/16-j)/abs(11*N/16-N/2))^1.5)*...
                (1-(abs(T/4-i)/abs(T/4-T/6))^1.5));
    end
end
for i = round(9*T/16):round(15*T/16)
    for j = round(N/2):N
        y(i,j) = y(i,j)*...
            (1-0.3*(1-(abs(3*N/4-j)/abs(N/4))^1.5)*...
                (1-(abs(3*T/4-i)/abs(3*T/4-9*T/16))^1.5));
    end
end
for i = round(T/3):round(2*T/3)
    for j = round(N/2):round(19*N/20)
        x(i,j) = x(i,j)*...
            (1-0.4*(1-(abs(29*N/40-j)/abs(29*N/40-N/2))^1.5)*...
                (1-(abs(T/2-i)/abs(T/2-T/3))^1.5));
    end
end
for i = round(T/3):round(2*T/3)
    for j = 1:round(N/2)
        x(i,j) = x(i,j)*...
            (1+0.3*(1-(abs(N/4-j)/abs(N/4))^1.5)*...
                (1-(abs(T/2-i)/abs(T/2-T/3))^1.5));
    end
end
for i = [1:round(T/6) round(5*T/6):T]
    for j = 1:round(5*N/6)
        x(i,j) = x(i,j)*...
            (1-0.3*(1-(abs(5*N/12-j)/abs(5*N/12))^1.5)*...
                (1-(abs(1-i+T*(i>T/2))/abs(T/6))^1.5));
        y(i,j) = y(i,j)*...
            (1-0.3*(1-(abs(5*N/12-j)/abs(5*N/12))^1.5)*...
                (1-(abs(1-i+T*(i>T/2))/abs(T/6))^1.5));
    end
end
for i = 1:round(T/4)
    for j = round(N/4):round(3*N/4)
        y(i,j) = y(i,j)*...
            (1-0.2*(1-(abs(N/2-j)/abs(N/4))^1.5)*...
                (1-(abs(T/8-i)/abs(T/8))^1.5));
        x(i,j) = x(i,j)*...
            (1-0.2*(1-(abs(N/2-j)/abs(N/4))^1.5)*...
                (1-(abs(T/8-i)/abs(T/8))^1.5));
    end
end
```

```
    x = x(:,1:length(x(1,:))-round(1/20*N)) * 3/4;
    y = y(:,1:length(y(1,:))-round(1/20*N)) * 3/4;
    z = z(:,1:length(z(1,:))-round(1/20*N));
    z = z-min(z(:,end));
    x = x*XYZ_scale; y = y*XYZ_scale; z = z*XYZ_scale;
    figure()
    subplot(1,2,1)
    surf(x,y,z,'EdgeColor','none','FaceColor',[0 0.8 0.7],'FaceAlpha',0.2)
    hold on
    plot3(x(end,:),y(end,:),z(end,:),'k:')
    hold on
    plot3(x(1,:),y(1,:),z(1,:),'k:')
    hold on
    plot3(x(round(T/4),:),y(round(T/4),:),z(round(T/4),:),'k:')
    hold on
    plot3(x(round(T/2),:),y(round(T/2),:),z(round(T/2),:),'k:')
    hold on
    plot3(x(round(3*T/4),:),y(round(3*T/4),:),z(round(3*T/4),:),'k:')
    axis equal
    view(-15,28)
    title('Reference Shape')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    subplot(1,2,2)
    plot3(x,y,z,'.')
%     plot3(x,y,z,'k.');
    axis equal
    view(-15,28)
    title('Reference Points')
    xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
    suptitle('Generated Shape: Irregular')
    x = repmat(x,2,1); y = repmat(y,2,1); z = [z; zeros(size(z))];
end

end
```

---

closest_pt.m

```
function [XYZ_closest] = closest_pt(xyz,XYZ)
% Returns point within XYZ which minimizes squared distance to point xyz

    dist_squared = sum((xyz-XYZ).^2,2);
    closest_pt = find(dist_squared == min(dist_squared));
    if numel(closest_pt)>1
        closest_pt = min(closest_pt);
    end
    XYZ_closest = XYZ(closest_pt,:);

end
```

---

circle_path.m

```
function [p_path,q_path,ori_i_path,vel_i_path,time] = ...
                                circle_path(sim_time,sampling_rate)
% INPUTS:   sim_time -- simulation run time (seconds, scalar)
%           sampling_rate -- simulation sampling rate (Hz, scalar)
% OUTPUTS:  p_path -- xyz points through time (m, Nx3)
%           q_path -- iterative quaternion through time (Nx4)
%           ori_i_path -- initial normal vector to path (3x3)
%           vel_i_path -- initial velocity following path (m/s, 3x1)
%           time -- corresponding time matrix (seconds, Nx1)

    t = sim_time; % seconds
    dt = 1/sampling_rate;
    N = t/dt;

    % Define translational path
    maxtheta = 2*pi;    % rotation
    dtheta = maxtheta/N;
    theta = (0:dtheta:maxtheta-dtheta)';
    p = [cos(theta) sin(theta) zeros(N,1)] - ...
            repmat(cos(theta)+sin(theta),1,3)/3;    % position, m
    p = p-p(1,:);
    v = [-sin(theta) cos(theta) zeros(N,1)] - ...
            repmat(-sin(theta)+cos(theta),1,3)/3;
    xyz_scale = 0.1;
    p = p*xyz_scale;  v = v*xyz_scale;
    v = v*dtheta/dt;    % velocity, m/s
    path_center = mean(p,1);

    % Define basis vectors
    z_bas = (p-path_center);
    z_bas_norm = sqrt(z_bas(:,1).^2+z_bas(:,2).^2+z_bas(:,3).^2);
    z_bas = z_bas./z_bas_norm;
    z_const = (z_bas(:,2)./z_bas(:,1)).^2;
    y_bas = [(z_const./(1+z_const)).^(1/2) ...
                    (1./(1+z_const)).^(1/2) zeros(N,1)];
    x_bas = [y_bas(:,2).*z_bas(:,3)-y_bas(:,3).*z_bas(:,2) ...
                    y_bas(:,3).*z_bas(:,1)-y_bas(:,1).*z_bas(:,3) ...
                    y_bas(:,1).*z_bas(:,2)-y_bas(:,2).*z_bas(:,1)];
    x_bas_norm = sqrt(x_bas(:,1).^2+x_bas(:,2).^2+x_bas(:,3).^2);
    x_bas = x_bas./x_bas_norm;

    % Define rotation (reverse solve for quaternion)
%     q = ones(length(z_bas)-1,4);
%     for i = 1:length(z_bas)-1
%         q(i,:) = Q_backsolve(z_bas(i,:),z_bas(i+1,:));
%     end
%     qtheta = 0.05; % rad/s
%     dqtheta = qtheta*dt;
%     dqtheta = dqtheta*ones(N,1);
%     qv = [1 2 3];
%     qv = qv/norm(qv);
%     q = [cos(dqtheta/2) qv(1)*sin(dqtheta/2) ...
%                         qv(2)*sin(dqtheta/2) qv(3)*sin(dqtheta/2)];
    q = repmat([1 0 0 0],N,1);
```

```
    % OUTPUTS
    p_path = p;
    q_path = q;
    ori_i_path = [x_bas(1,:)' y_bas(1,:)' z_bas(1,:)']';
    vel_i_path = v(1,:);
    vel_i_path = vel_i_path;
    time = dt:dt:t;
end
```

---

generate_instrument.m

```
function [instrument_xyz,imu_pts,imu_ori] = ...
                                generate_instrument(instrument_shape,imu_num)
% Defines x,y,z points for measurement instrument and places IMUs in random
% locations on instrument outer surface
% x, y centered at 0, z starts at 0
% Options: sphere
% OUTPUTS: instrument_xyz -- instrument shape (Nx3, meters)
%          imu_pts        -- imu locations on instrument (Mx3, meters)
%          imu_ori        -- imu orientations (3x3xM)

if instrument_shape == 'sphere'
    N = 40;
    xyz_scale = 0.5;
    [x,y,z] = sphere(N);
    x = x*xyz_scale; y = y*xyz_scale; z = z*xyz_scale;
    z = z-min(reshape(z,[],1));
    instrument_xyz = [reshape(x,[],1) reshape(y,[],1) reshape(z,[],1)];
    locs = randi(N-1,[1 imu_num])+1;
    imu_pts = ones(imu_num,3);
    imu_ori = ones(3,3,imu_num);

    for i = 1:imu_num
        imu_pts(i,:) = [x(locs(i),locs(i)),y(locs(i),locs(i)),...
                                                z(locs(i),locs(i))];
        z_bas = (imu_pts(i,:)-mean(instrument_xyz(:,3)))' / ...
                    norm(imu_pts(i,:)-mean(instrument_xyz(:,3)));
        z_const = (z_bas(2)/z_bas(1))^2;
        y_bas = [(z_const/(1+z_const))^(1/2) (1/(1+z_const))^(1/2) 0]';
        x_bas = [y_bas(2)*z_bas(3)-y_bas(3)*z_bas(2) ...
                 y_bas(3)*z_bas(1)-y_bas(1)*z_bas(3) ...
                 y_bas(1)*z_bas(2)-y_bas(2)*z_bas(1)]';
        x_bas = x_bas/norm(x_bas);
        imu_ori(:,:,i) = [x_bas y_bas z_bas];
    end

    imu_pts = permute(imu_pts,[3 2 1]);
end

end
```

---

imu_setup.m

```
function [align,off,sens,noise] = imu_setup(A_range,G_range,...
                                    ADC_length,error_lvl,t,imu_num,error_type)
% Calculates error parameters (axis alignment, offset, sensitivity, noise)
% based on IMU parameters

raw_max = 2^(ADC_length-1); % LSB

align = [eye(3) eye(3)];     % accelerometer, gyroscope (unitless)
align = repmat(align,1,1,imu_num);
for i = 1:6
    for j = 1:imu_num
%         align(:,i,j) = align(:,i,j).*...
%                        (1+0.1*error_lvl*(2*rand(size(align(:,i,j)))-1));
        align(:,i,j) = align(:,i,j) .* ...
                       (1+error_lvl/5*normrnd(0,1e-2,[3 1 1]));
        align(:,i,j) = align(:,i,j)/norm(align(:,i,j));
    end
end

% off = 0.1*error_lvl * [A_range(2)*(2*rand(3,1,imu_num)-1) ...
%                    G_range(2)*(2*rand(3,1,imu_num)-1)];  % g, rad/s

off = error_lvl/5 * [A_range(2)*normrnd(0,1e-2,[3 1 imu_num]) ...
                     G_range(2)*normrnd(0,1e-2,[3 1 imu_num])]; % g, rad/s

sens = [raw_max/A_range(2) raw_max/G_range(2)];  % LSB/g, LSB/(deg/s)
sens = repmat(sens,1,1,imu_num);
% sens = sens.*(1+0.1*error_lvl*(2*rand(size(sens))-1));
sens = sens.*(1+error_lvl/5*normrnd(0,1e-2,[1 2 imu_num]));
sens = [1 1]./sens; % g/LSB, (deg/s)/LSB

noise = error_lvl/5 * [normrnd(0,1e-2,[length(t) 3 imu_num]) ...
                       10*normrnd(0,1e-2,[length(t) 3 imu_num])];
noise = noise + eps*ones(size(noise));

if strcmp(error_type,'accel_only') || strcmp(error_type,'accel_noise_only')
    align(:,4:6,:) = repmat(eye(3),1,1,imu_num);
    off(:,2,:) = repmat(zeros(3,1),1,1,imu_num);
    sens(:,2,:) = repmat(raw_max/G_range(2),1,1,imu_num);
    noise(:,4:6,:) = eps*ones(size(noise(:,4:6,:)));
elseif strcmp(error_type,'gyro_only') || ...
                                    strcmp(error_type,'gyro_noise_only')
    align(:,1:3,:) = repmat(eye(3),1,1,imu_num);
    off(:,1,:) = repmat(zeros(3,1),1,1,imu_num);
    sens(:,1,:) = repmat(raw_max/A_range(2),1,1,imu_num);
    noise(:,1:3,:) = eps*ones(size(noise(:,1:3,:)));
end

if strcmp(error_type,'accel_noise_only') || strcmp(error_type,'noise_only')
    align(:,1:3,:) = repmat(eye(3),1,1,imu_num);
    off(:,1,:) = repmat(zeros(3,1),1,1,imu_num);
    sens(:,1,:) = repmat(raw_max/A_range(2),1,1,imu_num);
end
```

```
if strcmp(error_type,'gyro_noise_only') || strcmp(error_type,'noise_only')
    align(:,4:6,:) = repmat(eye(3),1,1,imu_num);
    off(:,2,:) = repmat(zeros(3,1),1,1,imu_num);
    sens(:,2,:) = repmat(raw_max/G_range(2),1,1,imu_num);
end

end
```

---

virtualIMU.m

```
function [accelerometer,gyroscope] = virtualIMU(pos,q,time,...

alignment,offset,sensitivity,noise,bits,init_ori)

% Calculates accelerometer and gyroscope output given position,
% orientation, linear acceleration, angular velocity, direction of gravity
% INPUTS: position (3xN, GRF), rotation quaternion (3xN, iterative),
% initial gyroscope and accelerometer orientations (3x3, GRF)
% initial position 3x1, initial orientation 3x3
% OUTPUTS: simulated accelerometer, gyroscope data

% Separate error parameters into accelerometer, gyroscope values
align_A = alignment(:,1:3); align_G = alignment(:,4:6);
offset_A = offset(:,1); offset_G = offset(:,2);
sens_A = sensitivity(1); sens_G = sensitivity(2);
noise_A = noise(:,1:3); noise_G = noise(:,4:6);

% Add gravity to acceleration values
N = length(pos);
dt = max(time)/(N);
gravity = 9.81*[zeros(N,2) -1*ones(N,1)];

% Convert offset to Nx3 matrices in LSB (from g, rad/s)
offset_A = offset_A/sens_A;
offset_G = offset_G/sens_G;
offset_A = [offset_A(1)*ones(N,1) offset_A(2)*ones(N,1)
offset_A(3)*ones(N,1)];
offset_G = [offset_G(1)*ones(N,1) offset_G(2)*ones(N,1)
offset_G(3)*ones(N,1)];

% Convert noise to LSB (from g, rad/s)
noise_A = noise_A/sens_A;
noise_G = noise_G/sens_G;

% Change sensitivities to (m/s^2)/LSB, (rad/s)/LSB
sens_A = sens_A*9.81;
sens_G = sens_G*pi/180;

% Calculate accelerometer, gyroscope ranges
% bits = 32;      % 24 max; raise for ideal situation; aim for 16
range_A = sens_A*2^(bits-1);
range_G = sens_G*2^(bits-1);
```

```
% Define orientation matrices (3x3)
Oa = zeros(3,3,N); Og = zeros(3,3,N);
Oa(:,:,1) = init_ori*align_A;
Og(:,:,1) = init_ori*align_G;

% Rotate accel, gyro orientations iteratively using quaternion array q
for i = 1:N-1
    Oa(:,:,i+1) = [Q_rotate(Oa(:,1,i),q(i,:)) ...
                   Q_rotate(Oa(:,2,i),q(i,:)) ...
                   Q_rotate(Oa(:,3,i),q(i,:))];
    Og(:,:,i+1) = [Q_rotate(Og(:,1,i),q(i,:)) ...
                   Q_rotate(Og(:,2,i),q(i,:)) ...
                   Q_rotate(Og(:,3,i),q(i,:))];
end

acc = zeros(N,3);
for i = 1:N-2
    acc(i,:) = (pos(i,:)-2*pos(i+1,:)+pos(i+2,:))/dt^2;
end
acc = acc+gravity;

% Calculate GRF gyro from orientation quaternion
theta = 2*acos(q(:,1))+eps;
omega = [q(:,2).*theta./sin(theta/2) ...
         q(:,3).*theta./sin(theta/2) ...
         q(:,4).*theta./sin(theta/2)];
% omega(isnan(omega))=0;              % new line
omega = omega/dt;

accelerometer = zeros(N,3); gyroscope = zeros(N,3);

% Calculate rotated accel, gyro from Oa/Og and acc/omega
for i = 1:N-1
    accelerometer(i,:) = Oa(:,:,i)\acc(i,:)';
%     gyroscope(i,:) = Og(:,:,1)\omega(i,:)';    % Note: Og(:,:,1) = align_G
    gyroscope(i,:) = align_G\omega(i,:)';
end

% accelerometer = round(accelerometer/sens_A)*sens_A;
% gyroscope = round(gyroscope/sens_A)*sens_A;
%
% accelerometer(abs(accelerometer)>range_A) = range_A;
% gyroscope(abs(gyroscope)>range_G) = range_G;

% Convert accelerometer and gyroscope data to LSB
accelerometer = accelerometer/sens_A;
gyroscope = gyroscope/sens_G;

% Add noise, offset
accelerometer = accelerometer + offset_A + noise_A;
gyroscope = gyroscope + offset_G + noise_G;

end
```

ori_basic_calc.m

```
function [ori] = ori_basic_calc(gyro,time,alignment,offset,...
                                                    sensitivity,init_ori)

N = length(time);
tstep = max(time)/N;

align_G = alignment(:,4:6);
sens_G = sensitivity(2)*pi/180; % deg/s / LSB to rad/s / LSB
offset_G = offset(:,2)/sens_G;
offset_G = [offset_G(1)*ones(N,1) ...
                offset_G(2)*ones(N,1) offset_G(3)*ones(N,1)];

offset_G = offset_G * pi/180;
gyro = gyro-offset_G;
gyro = gyro*sens_G;

Og = zeros(3,3,N);
Og(:,:,1) = init_ori*align_G;
% Og(:,:,1) = align_G;%%

theta = 0;
tsave = zeros(N,1);

r = ones(N,4);
for i = 1:N
    % Rotate angular velocity to GRF
% %     w_g = Og(:,:,1)*gyro(i,:)';
    w_g = align_G*gyro(i,:)';%%%
    % Rotation quaternion, based on angular velocity
    theta = theta+norm(w_g)*tstep;
    tsave(i) = theta;
    u = w_g/norm(w_g);
    u(find(isnan(u)))=0;
    r(i,:) = [cos(theta/2); u*sin(theta/2)]';
end

Og = repmat(Og(:,:,1),1,1,N);
Og1 = Q_rotateN(permute(Og(:,1,:),[3 1 2]),r);
Og2 = Q_rotateN(permute(Og(:,2,:),[3 1 2]),r);
Og3 = Q_rotateN(permute(Og(:,3,:),[3 1 2]),r);
Og = [permute(Og1,[2 3 1]) permute(Og2,[2 3 1]) permute(Og3,[2 3 1])];

ori = Og;

for i = 1:3
    ori(:,i,:) = ori(:,i,:) ./ ...
                        sqrt(ori(1,i,:).^2+ori(2,i,:).^2+ori(3,i,:).^2);
end

end
```

ori_correct.m

```
function [ori_mod] = ori_correct(ori,instrument_xyz,imu_ori_i,...
            correction_method,gyroscope,time,alignment,offset,sensitivity)

N = length(time);
tstep = max(time)/N;
imu_num = length(ori(1,1,1,:));

if strcmp(correction_method,'averaging')
    ori_mod = ones(size(ori));
    for i = 1:imu_num
        ori_mod(:,:,:,i) = ori_basic_calc(mean(gyroscope,3),...
                    time,alignment,offset,sensitivity,imu_ori_i(:,:,i));
    end

elseif strcmp(correction_method,'instrument_shape')
    %MODIFY METHOD TO CALCULATE INITIAL ORIENTATIONS WITH ACCELEROMETER
    % also rotate instrument through time to check accuracy

    norm_diff_init = ones(nchoosek(imu_num,2),3);
    i = 1;
    for n = 1:(imu_num-1)
        for m = (n+1):imu_num
            diff_init = imu_ori_i(:,:,m) - imu_ori_i(:,:,n);
            norm_diff_init(i,1) = n; norm_diff_init(i,2) = m;
            norm_diff_init(i,3) = norm(diff_init(:,1));
            norm_diff_init(i,4) = norm(diff_init(:,2));
            norm_diff_init(i,5) = norm(diff_init(:,3));
            i = i+1;
        end
    end

    ori_mod = ones(size(ori));
    ori_mod(:,:,1,:) = imu_ori_i;
    for j = 2:N
        % update orientation
        for k = 1:imu_num
            ori_mod(:,:,j,k) = ori_step_update(gyroscope(j,:,k),tstep,...
                alignment(:,:,k),offset(:,:,k),sensitivity(:,:,k),...
                ori_mod(:,:,j-1,k));
        end
        ori_mod_shift = ori_shift_solve(norm_diff_init(:,3:5),imu_num,...
                                                    ori_mod(:,:,j,:));
        ori_mod(:,:,j,:) = ori_mod(:,:,j,:) + ori_mod_shift;
        j
    end

elseif strcmp(correction_method,'none')
    ori_mod = ori;

end

end
```

ori_step_update.m

```
function [ori_step] = ori_step_update(gyro,tstep,alignment,offset,...
                                                    sensitivity,init_ori)
% gyro = 3x1, alignment = 3x6; offset = 3x2; sensitivity = 1x2
% init_ori = 3x3

align_G = alignment(:,4:6);
sens_G = sensitivity(2)*pi/180;
offset_G = offset(:,2)/sens_G;

offset_G = offset_G * pi/180;

gyro = gyro-offset_G';
gyro = gyro*sens_G;

Og = init_ori*align_G;

w_g = align_G*gyro';
theta = norm(w_g)*tstep;
u = w_g/norm(w_g);
q = [cos(theta/2); u*sin(theta/2)]';

ori_x = Q_rotate(Og(:,1),q);
ori_y = Q_rotate(Og(:,2),q);
ori_z = Q_rotate(Og(:,3),q);

ori_x = ori_x/norm(ori_x);
ori_y = ori_y/norm(ori_y);
ori_z = ori_z/norm(ori_z);

ori_step = [ori_x ori_y ori_z];

end
```

---

ori_shift_solve.m

```
function [ori_shift] = ori_shift_solve(norm_diff,imu_num,ori)

global n_IMU ND o1

n_IMU = imu_num;
ND = norm_diff;
o1 = ori;

options = optimset('Display','off','Algorithm','Levenberg-Marquardt');
ori_shift = fsolve(@shiftcalc,zeros(size(o1)),options);

end

function y = shiftcalc(o_shift)

global n_IMU ND o1
```

```
i = 1;
nd = ones(nchoosek(n_IMU,2),3);
for n = 1:n_IMU-1
    for m = n+1:n_IMU
        o_diff = (o1(:,:,m)+o_shift(:,:,m))-(o1(:,:,n)+o_shift(:,:,n));
        nd(i,1) = norm(o_diff(:,1));
        nd(i,2) = norm(o_diff(:,2));
        nd(i,3) = norm(o_diff(:,3));
        i = i+1;
    end
end

y = nd - ND;

end
```

---

pos_basic_calc.m

```
function [pos] = pos_basic_calc(ori,accel,time,alignment,offset,...
                                        sensitivity,init_pos,init_vel)
N = length(time);
tstep = max(time)/N;
gravity = 9.81*[zeros(N,2) -1*ones(N,1)];

align_A = alignment(:,1:3); %% new
sens_A = sensitivity(1)*9.81;
offset_A = offset(:,1)/sens_A;
offset_A = [offset_A(1)*ones(N,1) ...
                offset_A(2)*ones(N,1) offset_A(3)*ones(N,1)];

accel = accel-offset_A;
accel = accel*sens_A;
for i = 1:length(time)
accel(i,:) = accel(i,:)*align_A;     %% new
end

Oa = ori;

accel = permute(accel,[3 2 1]);
acc = bsxfun(@times,Oa,accel);
acc = permute(sum(acc,2),[3 1 2]);  % Nx3

acc = acc-gravity;

vel = cumsum((acc(1:end-1,:)+acc(2:end,:))/2*tstep);
vel = [vel; vel(end,:)];
vel = vel+init_vel;
pos = cumsum((vel(1:end-1,:)+vel(2:end,:))/2*tstep);
pos = [pos; pos(end,:)];
pos = pos+init_pos;

end
```

pos_correct.m

```
function [pos_mod] = pos_correct(pos,instrument_xyz,imu_pts,...
    correction_method,ori,accelerometer,time,offset,sensitivity,...
    init_pos,init_vel)

imu_num = length(ori(1,1,1,:));
N = length(time);
tstep = max(time)/N;

if strcmp(correction_method,'averaging')
    % calculate acceleration AFTER gravity removed
    % average this
    % apply to each imu to find position
    accel_gnd = ones(size(accelerometer));
    for i = 1:imu_num
        accel_gnd(:,:,i) = accel_earthframe(ori(:,:,:,i),...
            accelerometer(:,:,i),time,offset(:,:,i),sensitivity(:,:,i));
    end
    accel_avg = mean(accel_gnd,3);
    vel_mod = ones(size(pos));
    pos_mod = ones(size(pos));
    for i = 1:imu_num
        vel_mod(1:end-1,:,i) = ...
            cumsum((accel_avg(1:end-1,:)+accel_avg(2:end,:))/2*tstep);
        vel_mod(:,:,i) = [vel_mod(1:end-1,:,i); vel_mod(end-1,:,i)];
        vel_mod(:,:,i) = vel_mod(:,:,i)+init_vel;
        pos_mod(1:end-1,:,i) = ...
            cumsum((vel_mod(1:end-1,:,i)+vel_mod(2:end,:,i))/2*tstep);
        pos_mod(:,:,i) = [pos_mod(1:end-1,:,i); pos_mod(end-1,:,i)];
        pos_mod(:,:,i) = pos_mod(:,:,i)+init_pos(:,:,i);

    end

elseif strcmp(correction_method,'instrument_shape')
    % rotate instrument according to orientation (assumed correct)
    % known: associated closest point (exact, in simulation) for shape)
    % rotated pts to GROUND frame should keep exact distance apart
        % (not distance between relative orientations)
        % so using same method as above to solve for accel_gnd, solve for
        % difference in earth frame, and THAT should stay constant
    diff_init = ones(nchoosek(imu_num,2),3);
    i = 1;
    for n = 1:(imu_num-1)
        for m = (n+1):imu_num
            diff_init(i,:) = init_pos(:,:,m)-init_pos(:,:,n);
            norm_diff_init(i,1) = n; norm_diff_init(i,2) = m;%%
            norm_diff_init(i,3) = norm(diff_init(:,1));%%
            norm_diff_init(i,4) = norm(diff_init(:,2));%%
            norm_diff_init(i,5) = norm(diff_init(:,3));%%
            i = i+1;
        end
    end
    accel_gnd = ones(size(accelerometer));
```

```
    for i = 1:imu_num
        accel_gnd(:,:,i) = accel_earthframe(ori(:,:,:,i),...
            accelerometer(:,:,i),time,offset(:,:,i),sensitivity(:,:,i));
    end

    vel = ones(size(accel_gnd)); pos_mod = ones(size(accel_gnd));
    vel(1,:,:) = repmat(init_vel,1,1,imu_num);
    pos_mod(1,:,:) = init_pos;
    for j = 2:length(pos)
        vel(j,:,:) = vel(j-1,:,:) + accel_gnd(j,:,:)*tstep;
        pos_mod(j,:,:) = pos_mod(j-1,:,:) + vel(j,:,:)*tstep;
        pos_mod_shift = pos_shift_solve(norm_diff_init(:,3:5),imu_num,...
                                                    pos_mod(j,:,:));
        pos_mod(j,:,:) = pos_mod(j,:,:) + pos_mod_shift;
        j

    end

elseif strcmp(correction_method,'none')
    pos_mod = pos;

end

end
```

---

accel_earthframe.m

```
function [accel_gnd] = accel_earthframe(ori,accel,time,alignment,...
                                                    offset,sensitivity)

N = length(time);
gravity = 9.81*[zeros(N,2) -1*ones(N,1)];

align_A = alignment(:,1:3);
sens_A = sensitivity(1)*9.81;
offset_A = offset(:,1)/sens_A;
offset_A = [offset_A(1)*ones(N,1) ...
                offset_A(2)*ones(N,1) offset_A(3)*ones(N,1)];

accel = accel-offset_A;
accel = accel*sens_A;
for i = 1:length(time)
    accel(i,:) = accel(i,:)*align_A;    %% new
end

Oa = ori;

accel = permute(accel,[3 2 1]);
acc = bsxfun(@times,Oa,accel);
acc = permute(sum(acc,2),[3 1 2]);  % Nx3

accel_gnd = acc-gravity;

end
```

pos_shift_solve.m

```
% function [pos_shift] = pos_shift_solve(diff_init,imu_num,pos)
function [pos_shift] = pos_shift_solve(norm_diff,imu_num,pos)

% global n_IMU diff_i p1
global n_IMU ND p1

n_IMU = imu_num;
% diff_i = diff_init;
ND = norm_diff;
p1 = pos;

options = optimset('Display','off','Algorithm','Levenberg-Marquardt');
pos_shift = fsolve(@shiftcalc,zeros(size(p1)),options);

end

function y = shiftcalc(p_shift)

% global n_IMU diff_i p1
global n_IMU ND p1

i = 1;
nd = ones(nchoosek(n_IMU,2),3);%%
% p_diff = ones(nchoosek(n_IMU,2),3);
for n = 1:(n_IMU-1)
    for m = (n+1):n_IMU
        p_diff = (p1(:,:,m)+p_shift(:,:,m))-(p1(:,:,n)+p_shift(:,:,n));
        nd(i,1) = norm(p_diff(:,1));%%
        nd(i,2) = norm(p_diff(:,2));
        nd(i,3) = norm(p_diff(:,3));
        i = i+1;
    end
end

% y = p_diff - diff_i;
% y = sqrt(y(:,1).^2+y(:,2).^2+y(:,3).^2)
y = nd - ND;

end
```

draw_plots.m

```
function [] = draw_plots(time,pos,ori,imu_pos,pos_instr,ori_instr,...
                         p_path,ori_mod,pos_mod,accelerometer,gyroscope)

figure()
subplot(2,1,1)
plot(time,accelerometer(:,1,1));
hold on
```

```
plot(time,accelerometer(:,2,1));
hold on
plot(time,accelerometer(:,3,1));
xlabel('Time (seconds)'); ylabel('Accelerometer Raw Data (LSB)');
legend('X','Y','Z');
subplot(2,1,2)
plot(time,gyroscope(:,1,1));
hold on
plot(time,gyroscope(:,2,1));
hold on
plot(time,gyroscope(:,3,1));
xlabel('Time (seconds)'); ylabel('Gyroscope Raw Data (LSB)');
legend('X','Y','Z');
suptitle('Raw Generated IMU Data');

figure()
subplot(3,1,1)
plot(time,squeeze(ori(1,3,:,1)-ori(1,3,1,1)));
xlabel('Time (seconds)'); ylabel('X');
subplot(3,1,2)
plot(time,squeeze(ori(2,3,:,1)-ori(2,3,1,1)));
xlabel('Time (seconds)'); ylabel('Y');
subplot(3,1,3)
plot(time,squeeze(ori(3,3,:,1)-ori(3,3,1,1)));
xlabel('Time (seconds)'); ylabel('Z');
suptitle('Basic Orientation Calculation, IMU 1 (basis vector orthogonal to
surface)');

figure()
subplot(3,1,1)
plot(time,pos(:,1,1)); hold on; plot(time,imu_pos(:,1,1));
xlabel('Time (seconds)'); ylabel('X Position (m)')
legend('Calculated','Actual');
subplot(3,1,2)
plot(time,pos(:,2,1)); hold on; plot(time,imu_pos(:,2,1));
xlabel('Time (seconds)'); ylabel('Y Position (m)')
legend('Calculated','Actual');
subplot(3,1,3)
plot(time,pos(:,3,1)); hold on; plot(time,imu_pos(:,3,1));
xlabel('Time (seconds)'); ylabel('Z Position (m)')
legend('Calculated','Actual');
suptitle('Basic Position Calculation, IMU 1');

figure()
subplot(3,1,1)
% plot(time,squeeze(ori_instr(1,3,:)-ori_instr(1,3,1)));
plot(time,squeeze(ori_instr(1,3,:)));
xlabel('Time (seconds)'); ylabel('X');
subplot(3,1,2)
% plot(time,squeeze(ori_instr(2,3,:)-ori_instr(2,3,1)));
plot(time,squeeze(ori_instr(2,3,:)));
xlabel('Time (seconds)'); ylabel('Y');
subplot(3,1,3)
% plot(time,squeeze(ori_instr(3,3,:)-ori_instr(3,3,1)));
plot(time,squeeze(ori_instr(3,3,:)));
xlabel('Time (seconds)'); ylabel('Z');
suptitle('Instrument Orientation (basis vector orthogonal to surface)');
```

```
figure()
subplot(3,1,1)
plot(time,pos_instr(:,1)); hold on; plot(time,p_path(:,1));
xlabel('Time (seconds)'); ylabel('X Position (m)');
legend('Calculated','Actual');
subplot(3,1,2)
plot(time,pos_instr(:,2)); hold on; plot(time,p_path(:,2));
xlabel('Time (seconds)'); ylabel('Y Position (m)');
legend('Calculated','Actual');
subplot(3,1,3)
plot(time,pos_instr(:,3)); hold on; plot(time,p_path(:,3));
xlabel('Time (seconds)'); ylabel('Z Position (m)');
legend('Calculated','Actual');
suptitle('Instrument Position');

figure('Units','inches','Position',[4 4 6 6]);
subplot(3,1,1)
plot(time,squeeze(ori_mod(1,3,:,1)-ori_mod(1,3,1,1)));
hold on
plot(time,squeeze(ori(1,3,:,1)-ori(1,3,1,1)));
xlabel('Time (seconds)'); ylabel('X');
legend('Corrected','Basic','Location','northwest');
subplot(3,1,2)
plot(time,squeeze(ori_mod(2,3,:,1)-ori_mod(2,3,1,1)));
hold on
plot(time,squeeze(ori(2,3,:,1)-ori(2,3,1,1)));
xlabel('Time (seconds)'); ylabel('Y');
legend('Corrected','Basic','Location','northwest');
subplot(3,1,3)
plot(time,squeeze(ori_mod(3,3,:,1)-ori_mod(3,3,1,1)));
hold on
plot(time,squeeze(ori(3,3,:,1)-ori(3,3,1,1)));
xlabel('Time (seconds)'); ylabel('Z');
legend('Corrected','Basic','Location','northwest');
suptitle('Corrected Orientation, IMU 1 (basis vector orthogonal to
surface)');

figure('Units','inches','Position',[4 4 6 6]);
subplot(3,1,1)
plot(time,pos_mod(:,1,1));
hold on
plot(time,pos(:,1,1));
hold on
plot(time,imu_pos(:,1,1));
xlabel('Time (seconds)'); ylabel('X Position (m)');
legend('Corrected','Basic','Actual','Location','northwest');
subplot(3,1,2)
plot(time,pos_mod(:,2,1));
hold on
plot(time,pos(:,2,1));
hold on
plot(time,imu_pos(:,2,1));
xlabel('Time (seconds)'); ylabel('Y Position (m)');
legend('Corrected','Basic','Actual','Location','northwest');
subplot(3,1,3)
plot(time,pos_mod(:,3,1));
hold on
```

```
plot(time,pos(:,3,1));
hold on
plot(time,imu_pos(:,3,1));
xlabel('Time (seconds)'); ylabel('Z Position (m)');
legend('Corrected','Basic','Actual','Location','northwest');
suptitle('Corrected Position, IMU 1');

figure()
plot3(pos_instr(:,1),pos_instr(:,2),pos_instr(:,3),'.')
xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
axis equal

% comment out if using circle_path.m
Tri = delaunay(pos_instr(:,1),pos_instr(:,2),pos_instr(:,3));
figure()
trisurf(Tri,pos_instr(:,1),pos_instr(:,2),pos_instr(:,3),'FaceColor',...
    [0.6875 0.8750 0.8984],'FaceAlpha',0.3)
xlabel('X (m)'); ylabel('Y (m)'); zlabel('Z (m)');
axis equal

figure('Units','inches','Position',[4 4 6 6]);
subplot(3,1,1)
plot(time,(pos_mod(:,1,1)-imu_pos(:,1,1))*1e3);
hold on
plot(time,(pos(:,1,1)-imu_pos(:,1,1))*1e3);
xlabel('Time (seconds)'); ylabel('X Position (mm)');
legend('Corrected','Basic','Location','northwest');
subplot(3,1,2)
plot(time,(pos_mod(:,2,1)-imu_pos(:,2,1))*1e3);
hold on
plot(time,(pos(:,2,1)-imu_pos(:,2,1))*1e3);
xlabel('Time (seconds)'); ylabel('Y Position (mm)');
legend('Corrected','Basic','Actual','Location','northwest');
subplot(3,1,3)
plot(time,(pos_mod(:,3,1)-imu_pos(:,3,1))*1e3);
hold on
plot(time,(pos(:,3,1)-imu_pos(:,3,1))*1e3);
xlabel('Time (seconds)'); ylabel('Z Position (mm)');
legend('Corrected','Basic','Actual','Location','northwest');
suptitle('Position Error, IMU 1');

end
```

QUATERNION METHODS

Q_rotate.m

```
function [v_rotate] = Q_rotate(v,q)
% Rotates v (3x1) about q (1x4)

v_quatern = [0 v'];
qconj = Q_conjugate(q);

temp = Q_crossproduct(q,v_quatern);
v_temp = Q_crossproduct(temp,qconj);
v_rotate = v_temp(:,2:4);
v_rotate = v_rotate';

end
```

Q_rotateN.m

```
function [v_rotate] = Q_rotateN(v,q)
% Rotates Nx3 matrix iteratively using Nx4 matrix of quaternions

N = length(v);

v_quatern = [zeros(N,1) v];
qconj = Q_conjugateN(q);

v_temp = Q_multiplyN(Q_multiplyN(q,v_quatern),qconj);
v_rotate = v_temp(:,2:4);

end
```

Q_crossproduct.m

```
function [q_cross] = Q_crossproduct(q,p)
% Calculates quaternion cross product, q x p (both Nx4)

q_cross = [-q(:,3).*p(:,3)-q(:,2).*p(:,2)-q(:,4).*p(:,4)+q(:,1).*p(:,1) ...
            q(:,2).*p(:,1)+q(:,3).*p(:,4)+q(:,1).*p(:,2)-q(:,4).*p(:,3) ...
            q(:,4).*p(:,2)+q(:,1).*p(:,3)+q(:,3).*p(:,1)-q(:,2).*p(:,4) ...
            q(:,1).*p(:,4)+q(:,4).*p(:,1)+q(:,2).*p(:,3)-q(:,3).*p(:,2)];

end
```

Q_multiplyN.m

```
function [qC] = Q_multiplyN(qA,qB)
% INPUTS: Nx4 quaternion arrays qA, qB

a = permute(qA,[2 3 1]);
b = permute(qB,[3 2 1]);

c = bsxfun(@times,a,b);

% c = 4x4xN, where a = rows, b = columns --> c(a,b)

C = [c(1,1,:)-c(2,2,:)-c(3,3,:)-c(4,4,:);
     c(1,2,:)+c(2,1,:)+c(3,4,:)-c(4,3,:);
     c(1,3,:)-c(2,4,:)+c(3,1,:)+c(4,2,:);
     c(1,4,:)+c(2,3,:)-c(3,2,:)+c(4,1,:)];

qC = permute(C,[3 1 2]);

end
```

Q_conjugate.m

```
function [qconj] = Q_conjugate(q)
% Calculates conjugate of quaternion q (1x4)

qconj = [q(1) -q(2:4)];

end
```

Q_conjugateN.m

```
function [qconj] = Q_conjugateN(q)
% Quaternion conjugate of Nx4 quaternion array

qconj = [q(:,1) -q(:,2:4)];

end
```

Q_backsolve.m

```
function [q] = Q_backsolve(a,b)
% Given two 3x1 unit vectors A and B, solve for quaternion q to describe
% rotation from A to B
```

```
    global A B

    A = a; B = b;

    options = optimset('Display','off');
    q = fsolve(@qbacksolve,[1 0 0 0],options);

end

function y = qbacksolve(Q)

    global A B

    Q_cross = @(p,q) ...
    [-q(3).*p(3)-q(2).*p(2)-q(4).*p(4)+q(1).*p(1) ...
      q(2).*p(1)+q(3).*p(4)+q(1).*p(2)-q(4).*p(3) ...
      q(4).*p(2)+q(1).*p(3)+q(3).*p(1)-q(2).*p(4) ...
      q(1).*p(4)+q(4).*p(1)+q(2).*p(3)-q(3).*p(2)];

    Q_conj = @(q) [q(1) -q(2) -q(3) -q(4)];

    temp = Q_cross(Q,[0 B]);
    Qc = Q_conj(Q);

    y = Q_cross(temp,Qc) - [0 A];

end
```

Bibliography

[1] Ziegler-Graham, Kathryn, et al. "Estimating the prevalence of limb loss in the United States: 2005 to 2050." *Archives of physical medicine and rehabilitation* 89.3 (2008): 422-429.

[2] Nicholas, John J., et al. "Problems Experienced and Perceived by Prosthetic Patients." *JPO: Journal of Prosthetics and Orthotics* 5.1 (1993): 36-39.

[3] Nielsen, Caroline C. "A Survey of Amputees: Functional Level and Life Satisfaction, Information Needs, and the Prosthetist's Role." *JPO: Journal of Prosthetics and Orthotics* 3.3 (1991): 125-129.

[4] Zheng, Yong-Ping, Arthur FT Mak, and Aaron KL Leung. "State-of-the-art methods for geometric and biomechanical assessments of residual limbs: A review." *Journal of Rehabilitation Research and Development* 38.5 (2001): 487.

[5] Geil, Mark D. "Consistency, precision, and accuracy of optical and electromagnetic shape-capturing systems for digital measurement of residual-limb anthropometrics of persons with transtibial amputation." Journal of rehabilitation research and development 44.4 (2007): 515.

[6] Commean, Paul K., et al. "Precision of surface measurements for below-knee residua." Archives of physical medicine and rehabilitation 77.5 (1996): 477-486.

[7] Colombo, Giorgio, et. al. "Automatic 3D reconstruction of transfemoral residual limb from MRI images." *International Conference on Digital Human Modeling and Applications in Health, Safety, Ergonomics and Risk Management.* Springer Berlin Heidelberg, 2013.

[8] Dickinson, Alexander S., et al. "Registering a methodology for imaging and analysis of residual-limb shape after transtibial amputation." *Journal of Rehabilitation Research and Development* 53.2 (2016): 207-218.

[9] Chahande, Aunshumali, Sampath Billakanti, and Nicolas Walsh. "Lower limb shape characterization using feature extraction techniques [noncontact laser scanning]." Engineering in Medicine and Biology Society, 1994. Engineering Advances: New Opportunities for Biomedical Engineers. Proceedings of the 16th Annual International Conference of the IEEE. Vol. 1. IEEE, 1994.

[10] He, Ping, Kefu Xue, and Paul Murka. "3-D imaging of residual limbs using ultrasound." Journal of rehabilitation research and development 34.3 (1997): 269.

[11] Madgwick, Sebastian OH, Andrew JL Harrison, and Ravi Vaidyanathan. "Estimation of IMU and MARG orientation using a gradient descent algorithm." Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on. IEEE, 2011.

[12] Flenniken, W., J. Wall, and D. Bevley. "Characterization of Various IMU Error Sources and the Effect on Navigation Performance." *ION GNSS*. 2005.

[13] Syed, Z. F., et al. "A new multi-position calibration method for MEMS inertial navigation systems." Measurement Science and Technology 18.7 (2007): 1897.

[14] Fong, W.T., S. K. Ong, and A. Y. C. Nee. "Methods for in-field user calibration of an inertial measurement unit without external equipment." *Measurement Science and technology* 19.8 (2008): 085202

[15] Petovello, Mark G. Real-time integration of a tactical-grade IMU and GPS for high-accuracy positioning and navigation. *National Library of Canada (Bibliothèque nationale du Canada),* 2004.

[16] Guerrier, Stéphane. "Improving accuracy with multiple sensors: Study of redundant MEMS-IMU/GPS configurations." Proceedings of the 22nd international technical meeting of the Satellite Division of the Institute of Navigation (ION GNSS 2009). 2009.

[17] Colomina, I., et al. "Redundant IMUs for precise trajectory determination." Proceedings of the 20th ISPRS Congress, Istanbul, Turkey. Vol. 1223. 2004.

[18] InvenSense, "MPU-6000 and MPU-6050 Product Specification: Revision 3.4," MPU-6000/MPU-6050 Product Specification, 19 Aug. 2013.