A UKF-based orientation estimator for the Atlas platform

by

Jesse Linseman

A Thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfilment of

the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for

Mechanical and Aerospace Engineering

Department of Mechanical and Aerospace Engineering

Carleton University

Ottawa, Ontario, Canada

January 2010

Abstract

This dissertation presents a novel algorithm for real-time estimation of the orientation of a six degree of freedom motion simulation platform using an unscented Kalman filter (UKF). A low cost inertial sensor is more an instrument for higher frequencies but performs poorly over time due to high drift and needs an absolute sensor such as VOS to be corrected. Sensor fusion is used to obtain an improved estimate given measurements from two sensors: an inertial orienting sensor (IOS), and a visual orienting sensor (VOS). In order to overcome representational singularities associated with an unbounded orientation workspace of the platform, quaternions have been utilized within the unscented Kalman filter due to the characteristics of the Atlas simulator motion. IOS stabilized measurements act as direct input to the UKF algorithm and are further improved using statistical information about the gyro scaling factors, misalignments and drift provided by the manufacturer specifications. These errors are estimated and corrected using external absolute orientation measurements provided by a digital camera. The VOS measures the absolute orientation of the platform, processes the images, and obtains an estimated orientation quaternion, but at a slower frequency of approximately 20 Hz, compared to the IOS which operates at 76 Hz. Having concomitant orientation measurements sets up an opportunity to improve the overall accuracy of the orientation measurement. As well, online sensor calibration can be done to eliminate low frequency drift and other errors in the IOS measurements using knowledge of the latency in the data. However, this requires a method to combine the measurements; enter the quaternion based indirect UKF for sensor fusion with sensor error estimation and out of sequence measurement (OOSM) handling developed herein. Simulation of the filter was conducted accounting for the possibility of out of sequence measurements, measurement noise, and varying sensor frequencies. Real IOS data was recorded and passed through the UKF to examine the validity of the filter. Results of the simulation and experiments are detailed and discussed. The UKF is able to estimate the orientation, while at the same time compensate online for misalignments and drift caused by the IOS.

I would like to dedicate this work to my fiancé Katherine Ingrey for her continual belief in me and her loving care.

Acknowledgments

I would like to acknowledge the following individuals for their aid and guidance:

2009 Carleton University M.A.Sc. graduate, Kyle Chisholm, who helped with my understanding of utilizing a quaternion based UKF within Matlab. Particularly, your Matlab coding skills far exceeded my initial understanding. Without this help I would still be struggling.

Carleton mechanics Alex & Kevin for helping to construct the IMU testing bracket for use in my true data testing.

2009 Carleton University M.A.Sc. graduate, Ali Morbi, who provided his time, help and support using the encoded motor for acquiring synchronized recorded test data while running Matlab.

CUSP students, years 2007-2009 for all their documented work. Brian Rasquinha, your SSF code was well documented and easy to follow.

Finally, Carleton University, Professor Mojtaba Ahmadi & Professor M.John.D Hayes, for your knowledge and support leading toward completion of each task.

Table of Contents

Nomenclature

Symbol	Description
$[\phi,\ \theta,\ \psi]$	Roll, pitch, yaw Eulers (phi, theta, psi)
$\sigma_{\phi,\theta,\psi}$	Incremental Euler rotations
Ω	Atlas angular rate vector
$\dot{\Omega}$	Atlas angular acceleration vector
$\omega_{x,y,z}$	IOS angular rates
$\mathbf{q}_{rot}$	Rotation quaternion - used to find equivalent axis-angle
$\Delta\Theta$	Change in orientation
${}^{v}\mathbf{R}_{u}$	Rotation matrix from 'u'-frame to 'v'-frame
${}^{v}\mathbf{T}_{u}$	Translation matrix from 'u'-frame to 'v'-frame
${}^{n}\mathbf{P}_{x,y,z}$	Sphere marker position wrt 'n'-frame
$\mathbf{p}_{x',y'}$	Image plane point
$\mathbf{p}_{u,v}$	Pixel point
$\mathbf{s}_{x,y}$	VOS scale factors
$\mathbf{f}_{x,y}$	VOS focal length factors

Symbol	Description
$\mathbf{M}$	VOS perspective projection matrix
r_s	Atlas sphere radius
$\mathbf{x}_k^a$	'Augmented'(a) state vector
$\bar{x}$	Sample mean
$\mathbf{P}_{xx,k}$	State covariance matrix at time step 'k'
$\mathbf{Q}_{IOS}$	IOS process noise covariance matrix
$\mathbf{P}_{xx,k}^a$	'Augmented'(a) state covariance matrix at time step 'k'
$\mathbf{R}_{VOS}$	VOS Measurement noise covariance matrix
$\mathbf{q}_k$	4-component orientation quaternion
$\mathbf{b}_{g,k}$	Bias gyro error terms
$\mathbf{sf}_{g,k}$	Scale factor gyro error terms
$\mathbf{ma}_{g,k}$	Misalignment factor gyro error terms
$\mathbf{v}_{e,k}$	Rotation vector noise
$\mathbf{v}_{b_{g,k}}$	Gyro bias noise
$\mathbf{v}_{s_{g,k}}$	Gyro scale factor noise
$\mathbf{v}_{ma_{g,k}}$	Gyro misalignment factor noise
$\mathbf{k}_{g,sf}$	Gyro scale factor error as function of true rate
$\mathbf{k}_{g,ma}$	Gyro misalignment factor error as function of true rate

Symbol	Description
$\mathbf{G}_{g,k-\frac{1}{2}}$	Interpolated 'G'-matrix of sensor error factors
Δt_k	IOS measurement timestep
χ_i^a	i^{th} State sigma point
$\mathcal{W}_i$	Weight associated with the i^{th} state sigma point
κ	UKF scaling parameter
$\mathcal{Y}_i^a$	Transformed 'augmented'(a) sigma points
$\mathbf{f}(.)$	IOS state process function
$\mathbf{h}(.)$	VOS observation function
$\bar{\mathbf{x}}_k^{a-}$,$\mathbf{P}_{xx,k}^{-}$	'a-priori' estimates
$\mathbf{\Phi}_i, i = 1, ..., 2n$	axis-angle rotation vector errors
$\tilde{\mathbf{z}}_{k-N}$	Lagged (k-N) innovation vector
$\mathcal{Z}_{i,k-N}$	Latency delayed transformed observation sigma points
$\mathbf{z}_{VOS,k-N}$	VOS measurement state vector
$\bar{\mathbf{z}}_{k-N}^{-}$	'a-priori' mean observation estimate
$\mathbf{P}_{zz,k-N}^{-}$	'a-priori' observation covariance matrix
$\mathbf{P}_{\mathbf{x}_k\mathbf{z}_{k-N}}$	Cross-correlation-covariance-over-time matrix
$\mathbf{P}_{vv,k-N}$	Innovation covariance matrix
$\tilde{\mathbf{K}}_{k,N}$	Kalman gain term

Symbol	Description
$\tilde{\mathbf{q}}_{innov,k-N}$	Innovation quaternion
$\bar{\mathbf{x}}_k^+$	'a-posteriori' state estimate
$\mathbf{P}_{x,k}^+$	'a-posteriori' state covariance matrix

Acronym	Description
CUSP	Carleton University Simulation Project
UKF	Unscented Kalman Filter (with Sensor Error Estimation)
UKF_NoSE	Unscented Kalman Filter (No Sensor Error Estimation)
SSF	Simple Sensor Fusion
EKF	Extended Kalman Filter
UT	Unscented Transform
ADGC	Attitude Determination and Gyro Calibration Filter
MARG	Magnetic Angular Rate Gravity Sensor
MEMS	Micro-Electrical Mechanical System
CMM	Coordinate Measurement Machine
VOS	Vision Orientation Sensor
IOS	Inertial Orientation Sensor
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
OOSM	Out of Sequence Measurement
AWN	Angular White Noise
ARW	Angular Random Walk
RRW	Rate Random Walk

Chapter 1

Introduction

1.1 Motivation

1.1.1 The Atlas Simulation Platform

Estimation of three-dimensional (3D) position and orientation in a real-time simulation application necessitates an efficient means for gathering 'noisy' sensor measurements and 'filtering' out the error. The Atlas simulator, being developed at Carleton University as a fourth year project, is a unique platform that allows for unconstrained rotational freedom as shown in Figure 1.1. It is expected that Atlas will be used as an alternative for a flight simulator platform as compared to the industry standard Stewart platform [1]. However, due to a much greater range of motion, obtaining an optimal estimate of the orientation of the Atlas platform is not a simple matter. Some very unique characteristics of the Atlas platform add to the overall complexity in the design of an effective orientation estimation algorithm. The Atlas sphere is manipulated using three dual row omnidirectional wheels which maintain continuous contact with the sphere, but induce significant undesired vibrations which add to the difficulty of estimating the motion [2]. Moreover, there can be slippage between the contact point of the wheel and the sphere surface creating difficulty in measuring

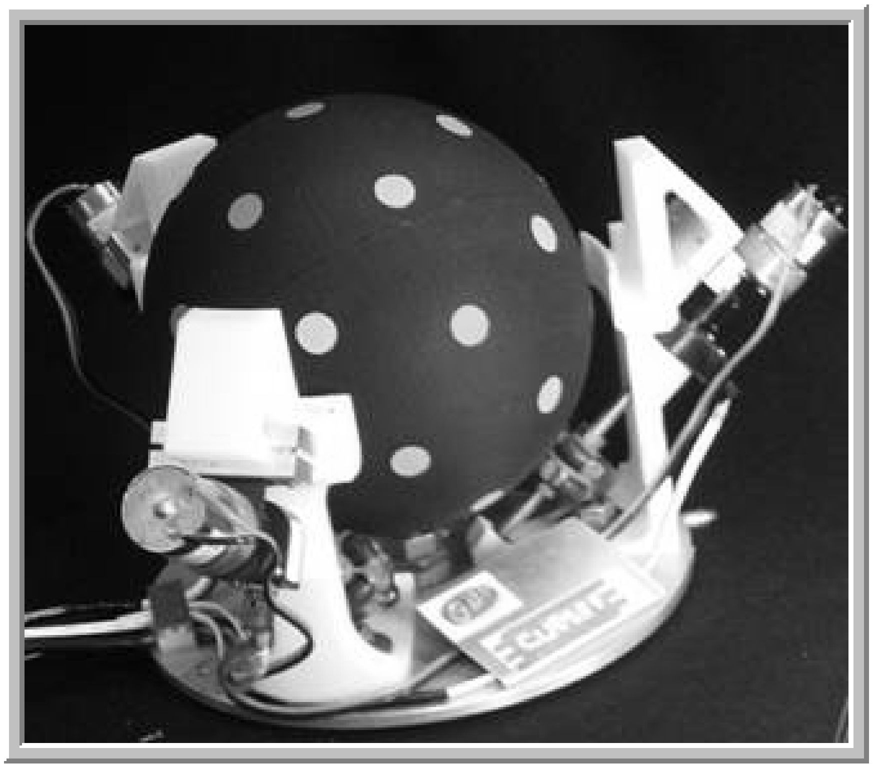


Figure 1.1: The AtlasLite motion platform.

rotational displacement. Attitude estimation of the Atlas platform is a non-linear problem that prevents the use of a classical Kalman filter. Add to this the inherent singularity issues that are posed when using Euler angles and a formidable control problem presents itself.

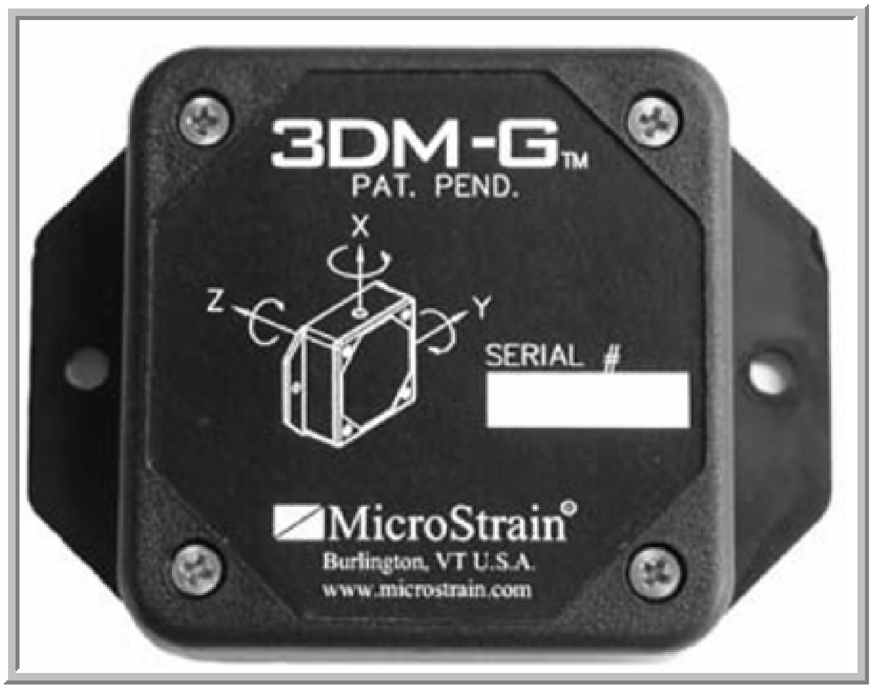
1.2 Background

1.2.1 Atlas Sensors

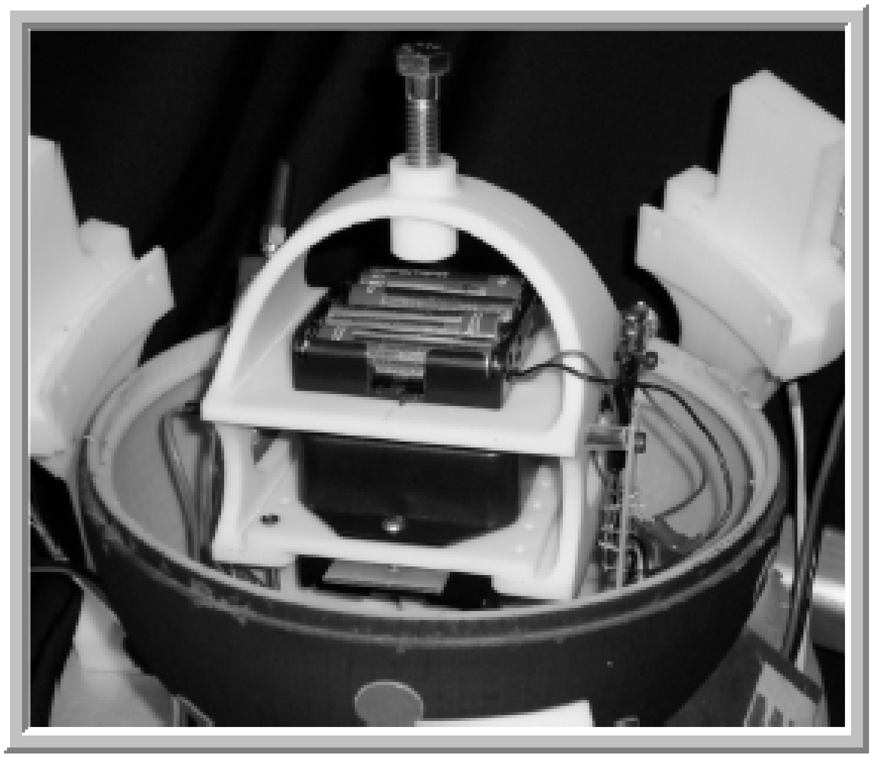
The Atlas simulator currently has a number of different sensors with the capability of determining the attitude of the sphere. Each omnidirectional wheel has an encoder which can directly measure the rotation angle of the wheel. However, the omnidirectional wheels are known to slip and therefore this measurement can not be solely relied upon. Future attempts utilizing a model of the slip may allow for encoder measurements to be used in some manner to determine the sphere's orientation, but at this point they are not used for reliability reasons. During the undertaking of this research, two different sensors were examined and deemed to be better suited for determining the Atlas sphere attitude: the Inertial Orientation Sensor (IOS) and the Vision Orientation Sensor (VOS).

1.2.2 Inertial Orientation Sensor (IOS)

The IOS, as it is referred to in Carleton University's simulator project (CUSP) documents, consists of a Microstrain 3DM-GX1 Inertial Measurement Unit (IMU), as shown in Figure 1.2a. It contains orthogonally mounted gyroscopes, accelerometers and magnetometers, the signals from which are combined in a compensation algorithm to stabilize the outputs. As a result the 3DM-GX1 can provide orientation information, in either Euler angles, rotation matrix or quaternion representation, as well as the angular rate and translational accelerations. The IMU is able to transmit wireless data using a Sena ParaniSD-100 Bluetooth adapter. The IMU is in a strapdown configuration, where all inertial sensors (gyros and accelerometers) are stiff mounted (strapped down) inside the Atlas sphere, as shown in Figure 1.2b. Hence, the IOS senses the Atlas sphere dynamic motions.



(a) The Microstrain 3DM-GX1 IMU.



(b) The 3DM-GX1 strapped-down inside the AtlasLite motion platform.

Figure 1.2: The 3DM-GX1 installed within the AtlasLite motion platform.

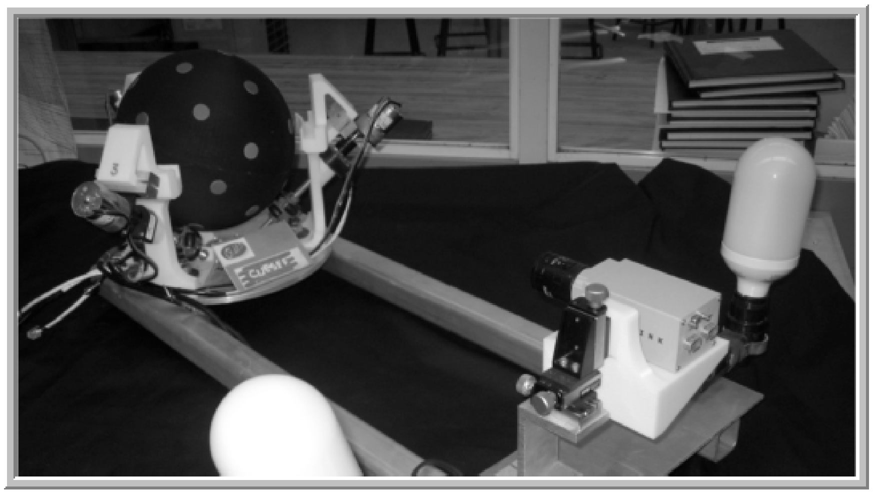


Figure 1.3: The AtlasLite technology demonstrator and digital camera setup.

1.2.3 Vision Orientation Sensor (VOS)

The IOS is able to sense the high frequency dynamics of the Atlas sphere but contains low frequency drift and sensor errors. The VOS is a complementary filter, able to correct the IOS measurement and compensate for this low frequency drift. As of August 2009, the VOS comprised of a digital camera fixed to a mount which focused the camera toward the Atlas sphere as shown in Figure 1.3 [3].

Using Carleton University's coordinate measurement machine (Faro CMM), 32 uniquely coloured markers have been placed at precise known locations on the outside of the AtlasLite sphere. Observing markers in an image, the camera software can distinguish their colours from which to lookup their corresponding local sphere frame coordinates. All 32 local sphere coordinate locations are recorded in a Matlab array file that was created during their CMM placement. Figure 1.4 shows the AtlasLite during the CMM marker placement process.

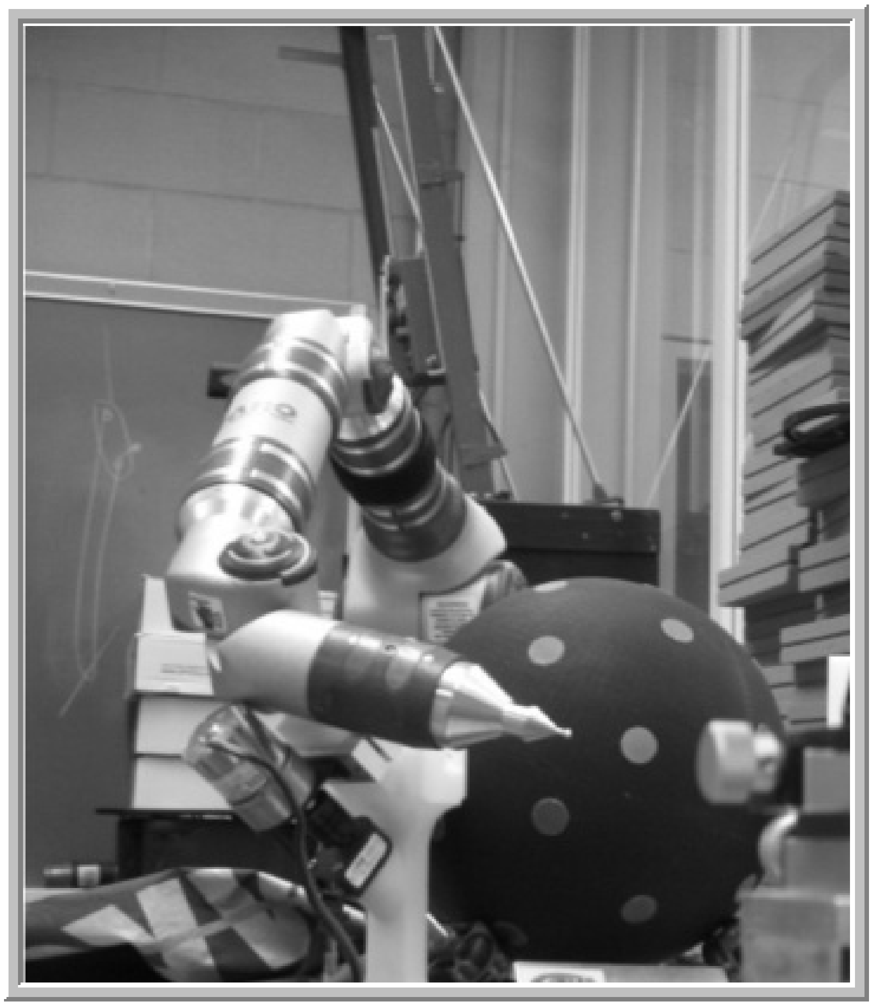


Figure 1.4: The marker placement on the exterior surface of the AtlasLite sphere using CMM.

Using Matlab, Figure 1.5 is a 3D plot that displays the current 32 marker locations surrounding the sphere's centre (local sphere frame origin). These points are what is currently available to test a UKF since at the time of the writing of this thesis, the VOS was undergoing modifications to improve the marker colour recognition capabilities for improved dynamic orientation measurements. Appendix E lists further details of the VOS.

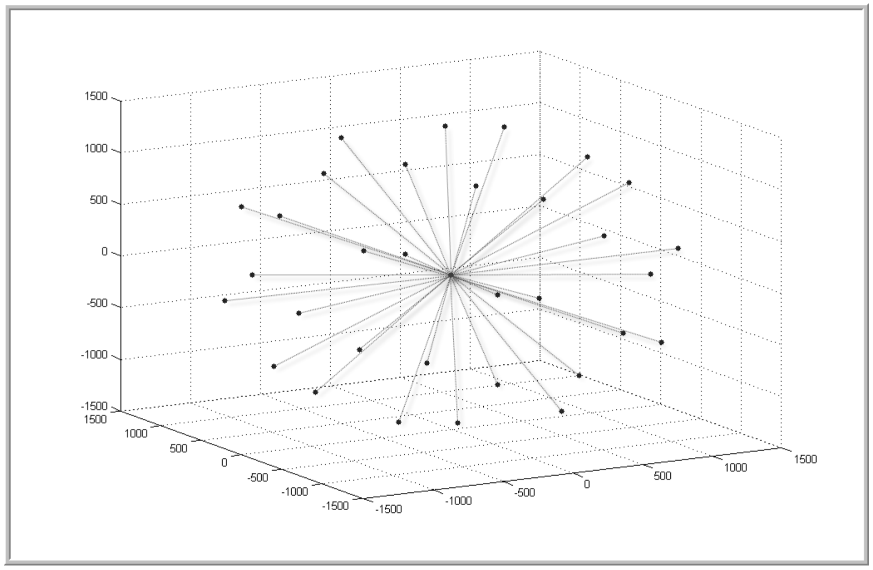


Figure 1.5: A Matlab representation of 32 local sphere marker positions (mm), ${}^{L}\mathbf{P}_{x,y,z}$.

1.2.4 Unscented Kalman Filtering

Unscented Kalman filtering is highly recognized as a useful estimation method when non-linear relationships exist, as is the case for the Atlas simulator rotational motion. A quaternion representation of the orientation is computationally effective and avoids problems of representational singularities [4], however quaternions are not easily visualized owing to their abstract 4 dimensional characteristics. Proper mathematical techniques for composing quaternions within the unscented Kalman filter must be well understood or errors may ensue. Use of quaternions has been investigated and shown to solve issues arising due to representational singularities, however, several extensions and adaptations to any straight forward UKF need to be recognized in order to incorporate an estimator's use within Atlas. Algorithmic procedures to properly convert between quaternions and other useful forms of orientation representations must be performed routinely to handle possible representational rotation singularity issues that may arise due to the Atlas simulator platform's unconstrained rotational freedom.

1.2.5 Developing An Algorithm for Attitude Estimation of the Atlas Platform

There have been many advances in Kalman filtering since it was first developed in 1961 by R.E. Kalman. An introduction to the basic Kalman filter can be found in Maybeck [5]. A Kalman filter is an optimal estimator of linear processes. However, if a process is non-linear, as in the case of the Atlas platform which can rotate about any axis through any number of degrees, another filter is sought; either the extended Kalman filter (EKF) or the unscented Kalman filter (UKF).

In 2007, Ahmadi proposed estimation of orientation of a rigid body using an error-state EKF which utilizes quaternions measured from a magnetic, angular rate, gravity (MARG) sensor [6]. Using a sampling time of 0.001 seconds a simulation was performed for a time interval of 25 seconds using the accelerometer and magnetometer as the aiding sensor measurements to correct the three orthogonal solid-state rate gyros. The simulation results for simultaneous rotations about all three axes indicate that EKF data fusion technique is feasible and better than integrating orientation kinematics.

Seeking a method to formulate more precise estimates for non-linear systems is what led to the unscented Kalman filter. Using the unscented transformation

(UT)(see Julier et al. [7], [8]), a set of sigma points are carefully chosen to encompass the sample mean and sample covariance of a measurement. These sigma points can then be propagated through any non-linear process, and the mean and covariance of the probability density function recovered. This eliminates the cumbersome derivation and evaluation of Jacobian/Hessian matrices making it much easier to implement than an EKF.

In 2003, Laviola performed comparison on unscented and extended Kalman filtering for estimating quaternion motion [9]. He found that both UKF and EKF can be used for virtual reality applications such as for human head motion tracking. Using a UKF, however, with each added state there is added computational demands since a larger covariance matrix is created and must be square rooted. Laviola found that the UKF did not provide any additional benefit in this case due to the simplicity of the Jacobian calculations for the process model. However, it was deemed the motion dynamics would need to have the important characteristic of small angle deviation and be sampled at relatively high rates to make use of the quasi-linear behaviour of quaternion motion estimation. For the case of the Atlas platform, the small angle deviation can not be assured since rotation is unlimited and can occur about any rotational trajectory. It is also speculated that since Atlas may undergo rapid rotations, and since the IOS has a maximum wireless sampling rate of 76 Hz, this may in some cases cause an EKF to become divergent or unfeasible.

In 2003, Kraft proposed a quaternion formulation utilizing the UT [4] to estimate the real-time change in orientation of a rigid body from measurements of its acceleration, angular velocity and magnetic bearing. Kraft outlines several extensions to the original UKF which are necessary to treat the inherent properties of unit quaternions, which makes it a valuable tool for research.

In 2005, Yuanxin Wu et al. [10], outlined the basic difference between an augmented and non-augmented UKF. The difference generally favours the augmented

UKF since sigma points only need to be produced once in a recursion. The general mathematical form of this recursive method is further utilized within the Atlas filter.

Quang M. Lam et al. [11] illustrate the advantages of using an attitude determination and gyro calibration (ADGC) filter under rapidly changing dynamic operating conditions to reduce the performance degradation often associated with low cost IMUs. In this 15 state filter, the effects of scale factors and misalignment factors are estimated online rather than calibrated offline. Using estimates of gyro error sources allows the filter to continue operating for short periods of time without the aiding sensor. Following from this thesis, the formulation of a 'G' matrix was adapted herein to estimate sensor error terms within the Atlas attitude UKF estimator.

In 2004, Eun-Hwan Shin developed a quaternion based UKF for integration of GPS and micro electrical mechanical systems (MEMS) inertial navigation system (INS) to overcome some of the limitations of the EKF [12]. The state vector includes position, velocity, attitude, and sensor biases, as well as scale factors with position information from the differential GPS (DGPS) solution used as the measurement updates in the UKF. Shin further demonstrates that with the UKF, error-prone Jacobian and Hessian computations are avoided and varying error models can be unified under one approach [13]. The added benefit of this approach is that large initial uncertainties in roll, pitch and yaw can also be allowed. Shin details an augmented system process model design with sensor error terms which has been further adapted for use with the Atlas IOS in this thesis [14].

To solve the problem that the weighted mean computation for quaternions does not produce an estimate in unit quaternion space, Yee-Jin Cheon et al. derived a weighted mean computation method for averaging quaternions in rotational vector space [15] for use in the KOMPSAT-1. The method showed that treating process noise as a rotation vector is a more suitable modeling approach than representing it as the vector part of a quaternion.

In order to compensate for OOSM, Julier et al. [7] offer solutions to optimally fuse latent sensor data in both linear and general nonlinear systems. Suggestions from this text have been utilized to modify the Atlas UKF developed herein.

Making full use of unit quaternions, the filter designed for the Atlas platform uses an error state modeling (indirect) formulation that avoids the necessity to model the dynamics of the simulator.

1.2.6 Direct vs. Indirect-Model of the Atlas Platform

It is important to recognize the difference between the total state space vs. error state space formulation of the filtering problem (also known as direct vs. indirect filtering) [5, 6]. In the total state space formulation the model for the filter is a set of dynamic equations governing the state of the system. The states would include orientation and the inputs to the filter would need to include both gyro and vision measurements. Attempting to use a full dynamic (direct) modeling approach poses serious drawbacks since the model would need to account for wheel slip, varying torques, varying inertial masses, buffeting between omni-directional castor wheels and vibration from the dual row omnidirectional wheels. The dynamic model would require a very large number of states and the added complexity would not always yield the expected results. As outlined in Ahmadi [6], dynamic modeling has the following drawbacks in this case:

- Dynamic modeling would need to be redone for any modification made to the simulator platform. i.e. a slightly different platform would require a new estimator.
- Dynamic modeling would require a very large number of states, thus increasing the computational burden.

- Dynamic modeling and the added complexity do not always produce the expected results.
- Precise modeling of the interaction of the platform and the environment is sometimes impossible.

Difficulty arises to continually update items such as mass, and moments of inertia for the ever changing platform. In the error state formulation, the Kalman filter estimates the errors in the gyro information using the difference between gyro and vision data as the measurements.

1.2.7 Sensor Fusion within Atlas

By utilizing an indirect formulation, dynamic modeling of the Atlas sphere is all together avoided. Indirect (also referred to as the error-state) UKF formulation is more commonly used in scenarios in which an external sensor is available to observe a strapdown internal sensor such as a gyroscope. Often, due to inaccuracies of torque models, the indirect filter is found to be more accurate [6, 16]. For this reason, in order to circumvent these issues, most attitude estimation applications in aerospace use gyros in a dynamic model replacement mode. The indirect modeling approach can be used so long as there exists sufficient sensing of the high frequency angular motion [16, 17]. Because the Atlas sphere houses the IMU, the IMU gyros follow the high frequency dynamics very accurately, and there is no need to model the dynamics explicitly. This is commonly known as a ‘strapdown’ configuration with further examples described in Titterton [18]. Changes to the platform mass, center of gravity, possible wheel slip, and torques can all be ignored in a strapdown formulation. Simply stated, with indirect modeling of the Atlas sphere, the dynamics of the system are completely ignored and the system is treated as a black box. The error state formulation is normally used for externally aided inertial navigation systems because of its many benefits including relative simplicity and reduced computation requirements. Treating the IOS in this manner eases future implementation into the full scale platform since the IOS itself can be removed and re-positioned inside an eventual full-scale platform. Sensor fusion is possible when two observations of the same event are available. The indirect UKF algorithm proposed in this thesis uses the IOS along with the VOS data to improve upon orientation quaternion estimates from either sensor alone.

1.3 Objectives & Approach

Investigation into various versions of the unscented Kalman filter detail capabilities which can handle some fundamentally challenging issues common to Atlas, such as latency witnessed in the arrival of measurements from the VOS. In this thesis, the attempt is to adapt each of these capabilities into one algorithm to allow proper fusion of two sensor measurements allowing an accurate orientation estimate to be maintained.

As it stands currently, the Atlas simulator has two sensors capable of accurately estimating the orientation of the sphere in real-time. Understanding each unique sensor is important for understanding how to properly fuse both measurements within an estimation filter. Process models for each sensor are needed to properly characterize the dynamic relationship that exists between each sensor's output and the noise that is associated with each in order to obtain an improved estimate of orientation using both sensors. In this effort, it is important to understand sources of errors/noises in each sensor, which corrupt each measurement, and their possible influence during motion of the sphere.

In this thesis, in order to set one fusion filter apart from other versions, simulation is used to characterize various fusion filters and determine their efficacy. The simulation is versatile enough to handle various sensor frequencies and test varying rotation rate profiles to properly assess each filter's efficacy. By simultaneously following a known rotation rate profile using an encoded motor while recording measurements from the internal IOS, a measured data set is acquired that can also be used for purposes of testing each filter.

Real-time implementation will require a fusion filter to be robust and efficient, in particular immune to representational singularities. In summary, the major objectives of this thesis are:

1. Adapt and prove the design of an unscented Kalman filter algorithm for use within the Atlas simulator platform. The algorithm should be able to fuse information from the IOS and VOS sensors, handle specified dynamic rotational motion typical within the Atlas platform, and ultimately improve upon either sensor measurement by itself, within outlined specifications for the Atlas platform [19]. The algorithm should also be robust during the entire process of estimating any arbitrary 3D orientation with the ability to handle inertial drift terms inherent to the IOS, and finally be able to handle known OOSM from the VOS throughout any dynamic trajectory expected for the Atlas platform.

2. Create a simulation which can test the estimation algorithm for various nonlinear trajectories to show the efficacy of the developed algorithm for eventual use within the Atlas simulator platform.

3. Demonstrate through the use of simulation and experimentation that such an algorithm would be more effective at estimating the Atlas platform's dynamic rotational motion compared to using either sensor by itself.

4. Provide an experimentation setup that would be useful to test the algorithm

for future implementation within the Atlas platform and detail the IOS (3DM-GX1) optimal mode of operation for use within the Atlas platform.

1.4 List of Contributions

The following novel research contributions are presented in this thesis.

1. An adapted version of an unscented Kalman filter is introduced which satisfies all the research objectives for the Atlas simulator. In doing so, a unique estimation algorithm was developed which can now be implemented within the Atlas simulator platform for improved orientation estimation using both the VOS and IOS.

2. A Simulation program is developed in Matlab to verify the adapted unscented Kalman filter for use within the Atlas platform showing that there is no divergence within the operational range.

3. An experimental procedure and test setup is developed with all the required software to communicate and process the IOS data in Matlab which can be used to record a known dynamic rotational trajectory followed by the IOS. Using this experimental setup, measured data is used to test the optimal mode of operation of the IOS for use in the Atlas simulator and to validate the orientation estimation algorithm. The code can be reused in the future real-time control of the Atlas platform.

1.5 Outline

This section provides an overview of the remaining chapters within this thesis.

Chapter 2: Quaternions This chapter outlines the quaternion algebra used throughout the thesis. The equations in this chapter are continually referred to for guidance to the reader.

Chapter 3: The Atlas Quaternion Based Indirect UKF with Sensor Error Estimation This chapter outlines the portions that have been adapted to create the Atlas unscented Kalman filter, highlighting the prediction and update steps.

Chapter 4: Operation of the IOS within Matlab for the Atlas UKF Testing This section details the Matlab code developed for communicating with the IOS and recording measurements. It also details the IOS modes of operation again referencing code listed in appendices.

Chapter 5: Simulation Testing of the Atlas UKF and Measured Data Results This chapter discusses the results from numerous verification experiments with comparison made to two other filter versions. The results validate the estimation algorithm developed as a method for determining the attitude of the Atlas platform by fusing VOS and IOS.

Chapter 6: Conclusions and Recommendations This chapter summarizes the research findings and discusses possible improvements and future steps for implementation.

Chapter 2

Quaternions

The Atlas simulator attitude (orientation), is represented by a four-element quaternion which has the ability to represent any 3-D orientation. First developed by Sir Rowan Hamilton in 1853 [20], with ideas from both vector and matrix algebra, the quaternion $\mathbf{q}$ may be viewed as a linear combination of a scalar q_0 and a spatial vector $\vec{q}$. A quaternion, $\mathbf{q}$, can be defined as a complex number

$$\mathbf{q} \;=\; q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}, \tag{2.1}$$

with $(q_0, q_1, q_2, q_3) \in \mathbb{R}$ and where $\mathbf{i}, \mathbf{j}$,and$\mathbf{k}$ are three orthogonal unit spatial vectors. The quaternion used for representing a rotation in the Atlas simulator is reduced from four to three degrees of freedom, as it satisfies a single normalization constraint given by

$$\mathbf{q}^T\mathbf{q} \;=\; 1, \tag{2.2}$$

where

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1.$$

Quaternions offer a representational singularity-free description (as opposed to

Euler angles). Rotations are computed more effectively as compared to rotation matrices [4]. A few important quaternion operations are outlined in the following sub-sections which will be adhered to for the remainder of this text.

2.1 Quaternion Multiplication and Rotations

Multiplication in quaternion vector space plays a crucial role in orientation kinematics and is defined differently from arithmetic multiplication [4], [21]. For two unit quaternions $\mathbf{q}_a$ and $\mathbf{q}_b$, the quaternion multiplication is defined as

$$\mathbf{q}_a \otimes \mathbf{q}_b = \begin{bmatrix} q_{a0} & -q_{a1} & -q_{a2} & -q_{a3} \\ q_{a1} & q_{a0} & -q_{a3} & q_{a2} \\ q_{a2} & q_{a3} & q_{a0} & -q_{a1} \\ q_{a3} & -q_{a2} & q_{a1} & q_{a0} \end{bmatrix} \begin{bmatrix} q_{b0} \\ q_{b1} \\ q_{b2} \\ q_{b3} \end{bmatrix}, \tag{2.3}$$

where

$$\mathbf{q}_a = \begin{bmatrix} q_{a0} & q_{a1} & q_{a2} & q_{a3} \end{bmatrix}; \mathbf{q}_b = \begin{bmatrix} q_{b0} & q_{b1} & q_{b2} & q_{b3} \end{bmatrix}.$$

Note: Quaternion multiplication is non-commutative: that is $\mathbf{q}_a \otimes \mathbf{q}_b \neq \mathbf{q}_b \otimes \mathbf{q}_a$.

Quaternion multiplication is used to perform successive rotations. The rotation quaternion $\mathbf{q}_{ab}$ which fulfills rotating from quaternion orientation $\mathbf{q}_a$ to quaternion orientation $\mathbf{q}_b$ using

$$\mathbf{q}_b = \mathbf{q}_{ab} \otimes \mathbf{q}_a \tag{2.4}$$

is simply given by

$$\mathbf{q}_{ab} = \mathbf{q}_b \otimes \mathbf{q}_a^{-1}, \tag{2.5}$$

where

$$\mathbf{q}_a^{-1} = \frac{q_0 - q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k}}{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \tag{2.6}$$

Note: Since pure rotations can be represented by unit quaternions, often q_a^{-1} is simply the conjugate since the above denominator is 1. That is, q_a is a normalized unit quaternion.

2.2 Quaternion from Axis-Angle

Axis-angle is used to parameterize a rotation by two values: A unit vector pointing from the origin to a point positioned an absolute distance of 1 unit away as illustrated in Figure 2.1, and an angle describing the magnitude of the rotation about the axis. The rotation occurs in the sense prescribed by the right hand rule.

This representation evolves from Euler's rotation theorem, which implies that any rotation or sequence of rotations of a rigid body in a three-dimensional space is equivalent to a pure rotation about a single fixed axis [23].

The axis-angle representation is equivalent to the more concise rotation vector representation. In this case, both the axis and the angle are represented by a non-normalized 3 component vector codirectional with the axis whose magnitude is the rotation angle. This 3 component rotation vector is used to represent the IMU/Aircraft orientation (phi ϕ, theta θ, psi ψ) corresponding to the current roll, pitch and yaw angular positions of the IMU away from initial rest respectively as illustrated in Figure 2.2.

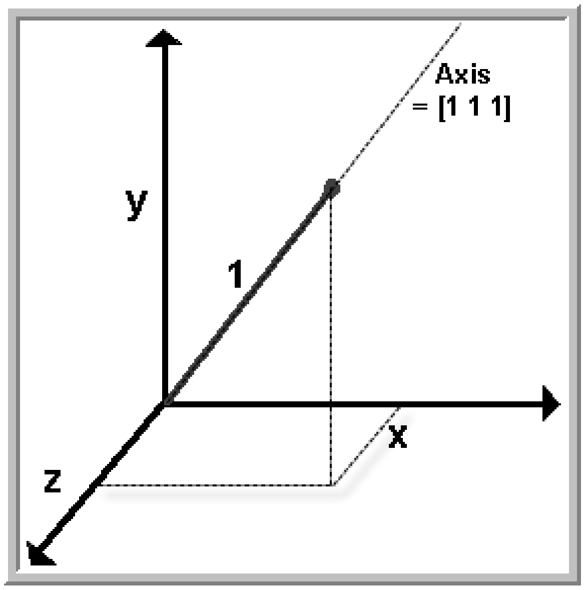


Figure 2.1: A unit vector depiction along the [1,1,1] axis, [22]

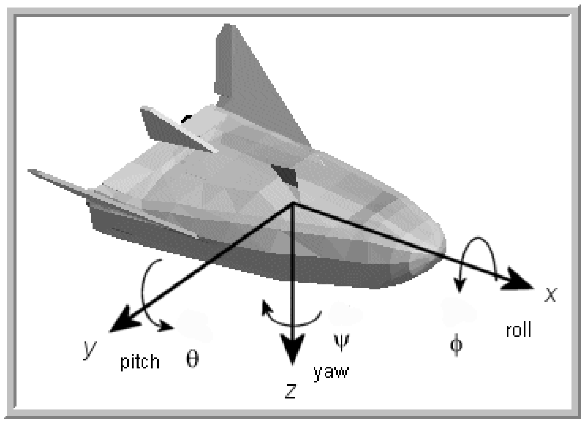


Figure 2.2: Local IMU/Aircraft Coordinate System, [22]

The concise axis-angle rotation vector representation, Θ, can be transformed into a quaternion rotation using

$$\mathbf{q}_{rot} = \begin{bmatrix} \cos\left(\frac{\|\Theta\|}{2}\right) \\ 0.5 * \sin\left(\frac{\|\Theta\|}{2}\right) * \boldsymbol{\Theta} \end{bmatrix}, \tag{2.7}$$

where

$$\|\Theta\| = \sqrt{\sigma_\phi^2 + \sigma_\theta^2 + \sigma_\psi^2},$$

and σ_ϕ,σ_θ, and σ_ψ are the variables which handle integration. They are obtained by integrating the IMU angular rate data from the local frame (2.8) using

$$\boldsymbol{\Theta} = \sigma_{\phi,\theta,\psi} = \int_{t_k}^{t_{k+1}} \omega_{x,y,z} dt, \tag{2.8}$$

which for a given short incremental IMU time interval Δt can be discretized using

$$\widetilde{\boldsymbol{\Theta}} = \sigma_{\phi,\theta,\psi} = \omega_{x,y,z} * \Delta t. \tag{2.9}$$

2.3 Axis-Angle from Quaternion

Axis-angle representation is only one possible way to represent the rotation of a solid 3D object. Simply stated, axis-angle is a rotation represented by a unit vector and an angle of revolution about that vector.

Contrary to quaternions, axis-angle is easy to visualize and very intuitive for 3D rotations of the Atlas sphere attitude. However, two axis-angle representations of rotations can not be directly combined to give an equivalent total rotation. For this we need to use matrices or quaternions. Since a rotation quaternion is related to axis-angle, it is easy to convert between them as explained in the last section.

When converting back to axis-angle representation care must be taken, however, since 3D rotations can be counterintuitive in some ways. There are two singularities at 0° and 180° where the axis can jump suddenly for a small change in input. The axis-angle representation, therefore, has two singularities at angle 0° and angle 180°. It is good practice to check to make sure the formula works in these cases for the Atlas platform implementation. At 0°, the rotation is said to be zero since the axis is arbitrary (any axis will produce the same result). Also at 180° rotation between two subsequent orientation quaternions would only be a concern if the Atlas sphere was able to rotate 180° before the IMU took a measurement. Since the IMU operates at at a frequency of 76 Hz, for logical reasons this will never be the case, thus, the rotation calculation for this case is also treated to be zero.

The process for obtaining the unit axis-angle representation involves using the equivalent rotation quaternion. From Equation 2.6, the rotation angle, α, which turns one orientation quaternion into the other is found between two quaternion orientations using

$$\begin{aligned}\mathbf{q}_{rot} &= \mathbf{q}_b \otimes \mathbf{q}_a^{-1} \\ &= \begin{bmatrix} q_{b0} & q_{b1} & q_{b2} & q_{b3} \\ -q_{b1} & q_{b0} & -q_{b3} & q_{b2} \\ -q_{b2} & q_{b3} & q_{b0} & -q_{b1} \\ -q_{b3} & -q_{b2} & q_{b1} & q_{b0} \end{bmatrix} \begin{bmatrix} q_{a0} \\ q_{a1} \\ q_{a2} \\ q_{a3} \end{bmatrix} .\end{aligned} \tag{2.10}$$

Distinguishing the scalar part, q_0, and vector parts, $q_{1,2,3}$, of the rotation quaternion, $\mathbf{q}_{rot}$, the unit axis-angle representation of the Atlas sphere rotation is found using

$$\mathbf{\Theta}_{\phi,\theta,\psi} = \frac{\alpha \cdot q_{1,2,3}}{s}, \tag{2.11}$$

where

$$\begin{aligned} \alpha &= 2 * \arccos(q_0) \\ s &= \sqrt{1-(q_0)^2}. \end{aligned}$$

Note: When $\alpha = 0$ or s becomes imaginary, axis-angle $\mathbf{\Theta}_{\phi,\theta,\psi} = [0,0,0]$.

2.4 Rotation Matrix from Quaternion

The equivalent axis-angle rotation matrix $\mathbf{R}$ can be constructed from a quaternion $\mathbf{q}$ using

$$\mathbf{R}_{XYZ(\gamma,\beta,\alpha)} = \begin{bmatrix} 1-2(q_2)^2-2(q_3)^2 & 2q_1q_2 - 2q_3q_0 & 2q_1q_3 + 2q_2q_0 \\ 2 * q_1q_2 + 2q_3q_0 & 1-2(q_1)^2-2(q_3)^2 & 2q_2q_3 - 2q_1q_0 \\ 2q_1q_3 - 2q_2q_0 & 2q_2q_3 + 2q_1q_0 & 1-2(q_1)^2-2(q_2)^2 \end{bmatrix} . \tag{2.12}$$

2.5 Quaternion from Rotation Matrix

An orientation quaternion $\mathbf{q}$ is composed from an orthogonal rotation matrix $\mathbf{R}$ using

- Calculate the Trace (the sum of the diagonal terms) of the rotation matrix, $\mathbf{R}$, and add 1.

$$\mathit{Trace}\left\{\mathbf{R}\right\}+1.$$

- If the trace of the rotation matrix plus 1 is greater than zero, ie.($\mathit{Trace}\left\{\mathbf{R}\right\}+1) > 0$, the orientation quaternion is calculated with

$$\begin{aligned}
q_0 &= \frac{\sqrt{1+R_{11}+R_{22}+R_{33}}}{2}, \\
q_1 &= \frac{R_{32}-R_{23}}{4q_0}, \\
q_2 &= \frac{R_{13}-R_{31}}{4q_0}, \\
q_3 &= \frac{R_{21}-R_{12}}{4q_0}.
\end{aligned} \tag{2.13}$$

Otherwise, if the trace of the matrix is less than or equal to zero, ie.($\mathit{Trace}\left\{\mathbf{R}\right\}+1) \leq 0$, then identify which major diagonal element has the greatest value.

- If $\mathbf{R}_{11}$ has the greatest value

$$\begin{aligned}
q_0 &= \frac{R_{32}-R_{23}}{4q_0}, \\
q_1 &= \frac{\sqrt{1+R_{11}+R_{22}+R_{33}}}{2}, \\
q_2 &= \frac{R_{21}+R_{12}}{4q_0}, \\
q_3 &= \frac{R_{13}+R_{31}}{4q_0}.
\end{aligned} \tag{2.14}$$

- If $\mathbf{R}_{22}$ has the greatest value

$$\begin{aligned} q_0 &= \frac{R_{13} - R_{31}}{4q_0}, \\ q_1 &= \frac{R_{21} + R_{12}}{4q_0}, \\ q_2 &= \frac{\sqrt{1 + R_{11} + R_{22} + R_{33}}}{2}, \\ q_3 &= \frac{R_{32} + R_{23}}{4q_0}. \end{aligned} \tag{2.15}$$

- Otherwise

$$\begin{aligned} q_0 &= \frac{R_{21} - R_{12}}{4q_0}, \\ q_1 &= \frac{R_{13} + R_{31}}{4q_0}, \\ q_2 &= \frac{R_{32} + R_{23}}{4q_0}, \\ q_3 &= \frac{\sqrt{1 + R_{11} + R_{22} + R_{33}}}{2}. \end{aligned} \tag{2.16}$$

Chapter 3

A Quaternion Based Indirect Unscented Kalman Filter with Sensor Error Estimation for use in Atlas

3.1 Atlas UKF Preliminaries

Similar to the Kalman filter, the UKF is a recursive process which involves prediction and update steps to continually make available, by estimation, the chosen states.

In general, as per Julier [8], an unscented transform can be used for forming a Gaussian approximation to a joint distribution of random variables x and z of a nonlinear discrete time system defined, with non-linear state equations using

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{f}[\mathbf{x}_k; \mathbf{u}_k; \mathbf{v}_k; k], \\ \mathbf{z}_k &= \mathbf{h}[\mathbf{x}_k; \mathbf{u}_k; k] + \mathbf{w}_k, \end{aligned} \tag{3.1}$$

where $\mathbf{x}_k$ is the n-dimensional state of the system at a dynamic time-step 'k', $\mathbf{u}_k$ is the input vector, $\mathbf{v}_k$ is the q-dimensional state noise process vector due to disturbances and modeling errors, $\mathbf{z}_k$ is the measurement vector and $\mathbf{w}_k$ is the measurement noise.

As per Julier, the n-dimensional random variable $\mathbf{x}_k$ with mean $\bar{\mathbf{x}}_k$ and state covariance $\mathbf{P}_{xx,k}$ is approximated by $2n + 1$ weighted 'sigma' points. These sigma points are chosen to propagate and accurately yield the covariance and mean of the probability distributions, given information about the noises associated to each state. The sigma points are not found at random but instead they are systematically chosen, eliminating the need to obtain the Jacobian or Hessian Matrices as in the extended Kalman filter [7].

Theoretically, taking a full set of sigma points and propagating them through both the IOS system process model and the VOS measurement model, the resulting estimate of the mean and covariance will often be more accurate when compared with an EKF [8, 24].

Realizing the UKF capabilities, a non-linear state equation for the determination of the Atlas orientation is surmised to take on a similar form to Equation (3.1). Since the IOS and VOS are both able to sense the Atlas simulator orientation with some amount of error, by using the faster IOS sensor measurements as the actual state to be estimated, the slower, more accurate, VOS measurements can be used to correct for drift, and in some way can be used to estimate the errors associated to the IOS measurement. A pair of sigma points are created for each of the states to be estimated, by using statistics on the expected noise. In this case, after considering the IOS and literature from [11] and [25], aside from the obvious estimation of the orientation quaternion, in order to correct for the most prominent noise sources, extra state terms were chosen to be estimated based on the Microstrain 3DM-GX1 manufacturer's specification sheet [26]. Appendix B.4 lists the Microstrain specifications. Further details of the choices are found in the next section.

3.2 Atlas Indirect UKF with Sensor Error Estimation

The Atlas UKF involves recursive prediction and update steps to continually make available the estimated states. Indirect (also referred to as the error-state) UKF formulation is more commonly used in scenarios in which an external sensor is available to observe a strapdown internal sensor such as a gyroscope. For this reason, it is opted to utilize an indirect formulation since dynamic modeling of the Atlas sphere is all together avoided.

Often, due to inaccuracies of torque models, the indirect filter is found to be more accurate [6, 16]. It has been shown that the Atlas sphere can be manipulated using dual-row omni-directional wheels which, although maintaining continuous contact with the sphere, will induce significant undesired vibrations which ultimately will add to the difficulty of estimating the motion [2]. Moreover, single row omni-directional wheels have large associated slip making them equally difficult to model. With indirect modeling, the dynamics of the system can be ignored and the system treated as a black box as long as the sensor errors can be predicted. In Atlas, the relative IOS measurements are considered to be the true angular rates of the sphere, along with added sensor error noise, and *angular white noise*(AWN). The IOS process model developed from this is

$$\omega_g = \omega_{true} + \mathbf{b}_g + \mathbf{k}_{g,sf} + \mathbf{k}_{g,ma} + \mathbf{n}_{g,a}, \tag{3.2}$$

where ω_{true} is the true angular rotation rate of the Atlas sphere, $\mathbf{b}_g$ is the gyro drift rate bias ($\frac{rad}{sec}$) driven by *rate random walk*(RRW), $\mathbf{k}_{g,sf}$ and $\mathbf{k}_{g,ma}$ are the rate level gyro scale factor and misalignment errors associate to the IOS, and $\mathbf{n}_{g,a}$ is white noise corrupting the IOS gyro rate measurement but becoming the *angular random walk*

(ARW) ($\frac{rad}{\sqrt{sec}}$) at the gyro angle level [11,25]. For a great description of angle random walk refer to Stockwell from Crossbow Inc. [27]. The rate level scale factor errors are modeled as a function of true rates obtained from the estimated scale factors using

$$\begin{aligned}
\mathbf{k}_{g,sf} &= \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \\
\mathbf{k}_{g,ma} &= \begin{bmatrix} 0 & ma_{xy} & ma_{xz} \\ ma_{yx} & 0 & ma_{yz} \\ ma_{zx} & ma_{zy} & 0 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}.
\end{aligned} \tag{3.3}$$

where $s_{x/y/z}$ represent the individual gyro residual scale factors and $ma_{xy/xz/yx/yz/zx/zy}$ represent the non-orthogonalities of the gyroscope triad. External vision measurements from the VOS are considered absolute attitude measurements with added noise which, presumably on some level, are able to observe the IOS bias drift and IOS sensor error factors, but the VOS operates at a much slower frequency.

3.2.1 Choosing the Augmented State Vector

From literature, there are two main UKF variants; augmented and non-augmented [10]. A non-augmented UKF is used for specific cases when the process and measurement noises are known to be additive. The basic difference is that the augmented UKF creates a set of sigma points once within a filtering recursion, while the non-augmented performs this process twice. The propagated covariance matrix is formed in the non-augmented UKF by adding the noise covariance, Q, to the state covariance. This will only hold for noise that is additive. Realizing comparisons made by

Wu et al. [10], it was opted to use an augmented UKF for Atlas as opposed to a non-augmented UKF for several reasons, namely:

Speed: The augmented UKF draws a set of sigma points only once within a filtering recursion, while the non-augmented UKF must redraw a new set of sigma points to fully incorporate the effect of additive process noise. It is computationally more efficient to create only one set of sigma points and propagate these points once through both the process and measurement models. However, creating an augmented state covariance matrix using the state error covariance, $\mathbf{P}_{xx,k-1}$, and the state process noise covariance, $\mathbf{Q}_k$, has a slight disadvantage for calculating matrix square roots. Since the augmented state covariance matrix is both square and positive definite, use of a Cholesky decomposition aids in speed savings.

Accuracy: Since it is unknown whether the process and measurement noises are additive or non-additive, the augmented UKF is used. Also a benefit, the augmented UKF is not restricted to be Gaussian as long as the state and observation noises are well characterized by mean and covariance information. It has been shown in [10] that the augmented UKF performs more accurately and more consistently mainly owed to its capability in capturing and propagating odd-order moment information, often associated to non-linear processes, throughout one filtering recursion.

Following from this, the generalized non-linear state augmentation equations for the Atlas orientation UKF are defined with

$$\begin{aligned} \mathbf{x}^a_{k+1} &= \mathbf{f}[\mathbf{x}^a_k; \mathbf{u}_k; k], \\ \mathbf{z}_k &= \mathbf{h}[\mathbf{x}^a_k; \mathbf{u}_k; k] + \mathbf{w}_k \end{aligned} \tag{3.4}$$

where $\mathbf{x}_k^a$ is the 31-dimensional state of the system at a dynamic time-step. This can further be defined with

$$\mathbf{x}_k^a = \begin{bmatrix} \mathbf{x}_k \\ \\ \mathbf{v}_k \end{bmatrix}$$

where $\mathbf{u}_k$ is the input vector, and $\mathbf{z}_k$ is the observation vector with the state noise $\mathbf{v}_k$ and observation noise $\mathbf{w}_k$ which can be assumed zero mean white Gaussian $N(0, \mathbf{Q}_{IOS})$, $N(0, \mathbf{R}_{VOS})$ respectively. Further details of the make-up of the Atlas UKF state vector are in the following section.

3.2.2 A Breakdown of the Atlas IOS Process Design

The Atlas/IOS process design can be broken down into two sub-routines; IOS mechanization and IOS sensor error compensation. Given that a low-cost IMU is used in a strapdown scenario, large biases, scale factors and misalignment factors are typically the most prominent errors to consider [13]. The augmented state vector is therefore designed to include these. Table 3.1 displays the augmented state vector for the UKF design.

State ($\mathbf{x}$)	Description	Size
$\mathbf{q}$	Attitude quaternion	4×1
$\mathbf{b}_g$	Gyroscope biases	3×1
$\mathbf{s}_g$	Gyroscope scale factors	3×1
$\mathbf{ma}_g$	Gyroscope misalignment factors	6×1
Noise ($\mathbf{v}$)		
$\mathbf{v}_e$	Rotation vector noises	3×1
$\mathbf{v}_{b_g}$	Gyro bias noises	3×1
$\mathbf{v}_{s_g}$	Gyro scale factor noises	3×1
$\mathbf{v}_{ma_g}$	Gyro misalignment factor noises	6×1

Table 3.1: The Alas UKF Augmented State Vector

IOS Mechanization

The Microstrain 3DM-GX1 can output multiple forms of data. Using a Sena ParaniSD-100 Bluetooth adapter, the 3DM-GX1 IMU has been found to transmit stabilized angular rates every 13.1072 milliseconds. This translates into a frequency of $\approx$ 76.2939 Hz. As previously outlined in Equation (2.9), IOS mechanization involves integrating the Euler angular rates into incremental rotations performed with $\Delta T = 0.0131072$ seconds reproduced as

$$\widetilde{\mathbf{\Theta}}_k = \sigma_{\phi,\theta,\psi} \quad = \quad \omega_{x,y,z} * \Delta t. \tag{3.5}$$

where $\widetilde{\mathbf{\Theta}}$ denotes the measured 3DM-G IMU incremental Euler angular rotations, and tilde $\widetilde{}$ denotes a measured sensor value as opposed to the actual value. The Atlas IOS sensor error terms are modeled as slowly varying random variables with added white Gaussian noise and are obtained using

$$\mathbf{b}_{g,k} \quad = \quad \mathbf{b}_{g,k-1} + \mathbf{v}_{g_b,k}, \tag{3.6}$$

$$\mathbf{s}_{g,k} \quad = \quad \mathbf{s}_{g,k-1} + \mathbf{v}_{g_s,k}, \tag{3.7}$$

$$\mathbf{ma}_{g,k} \quad = \quad \mathbf{ma}_{g,k-1} + \mathbf{v}_{g_{ma},k}, \tag{3.8}$$

with white Gaussian noise terms, $\mathbf{v}$. This noise is incorporated into the sensor models here to appropriately encompass the random walk nature of these typically non-stationary, non-zero mean error terms [12].

IOS Sensor Error Compensation

Before re-composing the incremental angular rotations to obtain the corresponding rotation quaternion, the IMU sensor measurement error estimates are first used to obtain compensated incremental angular rotations about each local IMU axis with

$$\boldsymbol{\Theta}_k = (\mathbf{I} + \mathbf{G}_{g,k-\frac{1}{2}})^{-1}[\widetilde{\boldsymbol{\Theta}}_k - \mathbf{b}_{g,k-\frac{1}{2}}\Delta T - \mathbf{v}_{e,k}], \tag{3.9}$$

where $\mathbf{I}$ is a 3×3 identity matrix and $\mathbf{G}_{g,k-\frac{1}{2}}$ is called the '**Gmatrix**', composed of the interpolated gyro scale factor and gyro misalignment sensor error terms where

$$\mathbf{G}_{g,k-\frac{1}{2}} = \begin{bmatrix} s_x & ma_{xy} & ma_{xz} \\ ma_{yx} & s_y & ma_{yz} \\ ma_{zx} & ma_{zy} & s_z \end{bmatrix}. \tag{3.10}$$

with *sandma* representing interpolated scale factors, $\mathbf{s}_{g,k-\frac{1}{2}}$, and misalignment factors, $\mathbf{ma}_{g,k-\frac{1}{2}}$, respectively. Note: interpolated sensor error terms are denoted with subscript '$k-\frac{1}{2}$'. Linear interpolation is used to obtain the sensor error values midway through the incremental rotation which are used for compensating the angular rate during each filter recursion.

As per Equation (2.7), the *rotation quaternion* is composed using the compensated incremental angular rotations calculated in Equation (3.9). The *rotation quaternion* is then applied to obtain the new orientation quaternion as follows:

$$\mathbf{q}_k = \mathbf{q}_{k-1} \otimes \mathbf{q}_{rot}. \tag{3.11}$$

Note: this use assumes the angular velocity vector remains constant during the short (constant) IOS time interval, Δt_k.

3.3 Prediction

Once again, the prediction portion of the Atlas UKF is performed for each IOS incremental measurement, regardless of whether a VOS measurement has been made available or not at this point. The following prediction process involves creating disturbances around the previous estimated 'augmented' state, $\bar{\mathbf{x}}^a_{k-1}$, based on the previous 'augmented' state covariance, $\mathbf{P}^a_{xx|k-1}$.

1. In general, with n representing the number of states estimated, an initial $\mathbf{n} \times 1$ dimensional random 'augmented' state vector $\mathbf{x}^a$ is created, with estimated mean $\bar{\mathbf{x}}^a_{k-1}$ and 'augmented' covariance $\mathbf{P}^a_{xx,k-1}$. This vector has a set of $2n+1$ sigma points created using

$$\begin{aligned}
\chi^a_0 &= \bar{\mathbf{x}}^a_{k-1}, \\
\chi^a_i &= \bar{\mathbf{x}}^a_{k-1} + \sqrt{(n+\kappa)\mathbf{P}^a_{xx,k-1}}_{\,i}, \\
\chi^a_{i+n} &= \bar{\mathbf{x}}^a_{k-1} - \sqrt{(n+\kappa)\mathbf{P}^a_{xx,k-1}}_{\,i}, \qquad (3.12) \\
\mathcal{W}_0 &= \kappa/(n+\kappa), \\
\mathcal{W}_i &= 1/2(n+\kappa), \\
\mathcal{W}_{i+n} &= 1/2(n+\kappa). \qquad (3.13)
\end{aligned}$$

where $i \in \{1, \cdots, n\}$, $\sqrt{\mathbf{P}^a_{xx,k-1}}_{\,i}$ is the i^{th} column of the 'Cholesky' matrix square root of $\begin{bmatrix} \mathbf{P}_{xx,k-1,15\times 15} & \mathbf{0}_{15\times 15} \\ \mathbf{0}_{15\times 15} & \mathbf{Q}_{IOS,15\times 15} \end{bmatrix}$ and $\mathcal{W}_i$ is the weight associated with the i^{th} sigma point, χ^a_i.

The scalar κ is a scaling parameter which is usually set to 0 or $3-n$. This is done so that fourth-order moment information is mostly captured in the true Gaussian case [7, 10].

Following from Kraft [4], using a quaternion formulation adds a level of complexity which must be taken into consideration for the Atlas UKF. The correct number of 'augmented' state sigma points need to be created. For the Atlas UKF, the scaling parameter is maintained in this case; $\kappa = 0$. The quaternion has four components in the Atlas UKF formulation, however, since a quaternion has only 3 degrees of freedom, as mentioned earlier, in this case $n = 30$. However, one cannot simply add the axis-angle disturbance to the previous quaternion. In order to add the current disturbance angles, a necessary step for this involves transforming the current *axis-angle disturbances* into an equivalent *disturbance quaternion* as per Equation (2.7) and using *quaternion multiplication* as per Equation (2.3) to apply the attitude disturbance. **Note:** Distinguishing the difference between an *orientation* quaternion and a *disturbance* or *rotation* quaternion which is used to rotate between two orientations is extremely important and a lot of times misunderstood from the literature.

Following from this, the remaining Atlas UKF state vector components are simply disturbed using standard vector addition as per a usual UKF formulation.

2. The Atlas 'augmented' sigma points are then instantiated through the Atlas process model to yield a set of transformed sigma points (3.14).

$$\mathcal{Y}_i^a = \mathbf{f}(\chi_k^a; \mathbf{u}_k^a; k) \tag{3.14}$$

where $\mathcal{Y}_i^a$ is used to clearly distinguish it as an estimate based on the IOS probability distribution and separate from an actual IOS or VOS measurement.

3. Next, the Atlas transformed sigma points must be properly averaged to obtain the 'augmented' 'a-priori' mean, $\bar{\mathbf{x}}_k^{a-}$, and 'a-priori' state covariance, $\mathbf{P}_{xx,k}^{-}$.

Note: Unlike the vector portions of the state, simple barycentric averaging does not yield correct results for the computation of the mean quaternion. The *orientation* portion of $\mathcal{Y}_i^a$, ie. the *quaternion* portion of the transformed sigma points, is averaged keeping in mind that quaternions are members of a homogeneous Riemannian manifold (the four dimensional unit sphere) and not of a vector space (see Kraft [4]). The approach uses the intrinsic gradient descent algorithm outlined in Appendix A.1.

4. The $[31 \times 1]$ 'augmented' 'a-priori' mean state estimate, $\bar{\mathbf{x}}_k^-$, is formed from the mean quaternion portion along with the average sensor error vector portions, and state noise portions.

5. Next, the Atlas 'a-priori' state covariance, namely $\mathbf{P}_{xx,k}^-$, is obtained as follows (3.15).

$$\mathbf{P}_{xx,k}^{a-} = \frac{1}{2n}\sum_{i=1}^{2n}[\mathcal{Y}_i^a - \bar{\mathbf{x}}_k^{a-}] \cdot [\mathcal{Y}_i^a - \bar{\mathbf{x}}_k^{a-}]^T \tag{3.15}$$

Note: In reference to $[\mathcal{Y}_i^a - \bar{\mathbf{x}}_k^{a-}]$ in Equation (3.15), the sigma point orientation axis-angle differences, $\mathbf{\Phi}_i, i = 1, ..., 2n$, between the propagated sigma points and the mean of the distribution are calculated from the last iteration of the mean finding algorithm (step(c) from Appendix A.1). It is these axis-angle differences along with the calculated differences pertaining to the sensor errors and noise vector portions which get used to create sigma point variance arrays $[30 \times 1]$ in size.

3.4 Update

Typically in a normal UKF, a measurement update step requires both the 'a-priori' state estimate, $\bar{\mathbf{x}}_k^{a-}$, and an estimate of the measurement, $\bar{\mathbf{z}}_k^-$ to both be available. For this reason, due to varying sensor frequencies in Atlas, the update portion of the Atlas UKF is instead performed only when the slower VOS measurement becomes available. Further complicating the matter, there is suggestion that each VOS measurement may be lagged due to image processing time [28]. For simplification, it is assumed that this latency-delay is well characterized and constant based on a known VOS frequency.

3.4.1 Out of Sequence Measurement Handling in Atlas UKF

In this section, an approach (developed by Julier et al. [7]) for optimally fusing latency delayed sensor data in *nonlinear systems* has been adapted for use in the Atlas UKF. With the VOS frequency known, (i.e. $\approx$ 20 Hz), the lag time can be calculated as $\Delta T_l = \frac{1}{VOS_{freq}}(sec)$. Next, this lag time is utilized to indicate when the first and all subsequent VOS measurements become available to the UKF. It is assumed the first lagged VOS measurement from timestep $l = k - N$ in the past, only becomes available to the Atlas UKF ΔT_l seconds later. The first IOS measurement, also from timestep $l = k - N$, is improved at timestep k, after ΔT_l seconds have passed.

Figure 3.1 depicts accurate fusion of a *lagged* innovation vector that takes place at time k to utilize the Kalman correction term that corresponds to time $l = k - N$.

Sensor Latency Compensation

In the Atlas UKF, optimally fusing an N-sample *lagged innovation vector*, $\tilde{\mathbf{z}}_{k-N}$ is accomplished using

$$\tilde{\mathbf{z}}_{k-N} = \mathbf{z}_{VOS,k-N} - \bar{\mathbf{z}}_{k-N}^-, \tag{3.16}$$

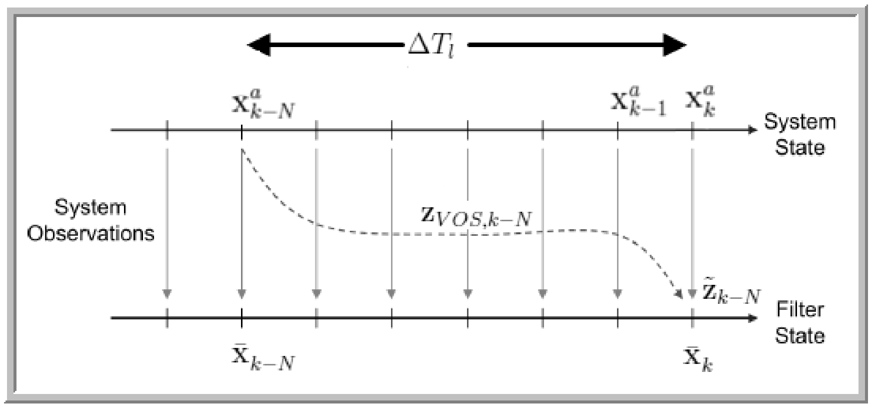


Figure 3.1: Out of Sequence Measurement from VOS Sensor Latency

which requires accurately obtaining the correct Kalman gain term, $\mathbf{K}_{k,N}$, where the IOS 'a-priori' state estimate and state covariance, corresponding to when the lagged VOS measurement was initially captured, are held in memory, $\bar{\mathbf{x}}^{a-}_{k-N}$ and $\mathbf{P}^{-}_{x|k-N}$ respectively.

1. Generally in an UKF, in order to characterize the influence of measurement noise, an observation function (or measurement model) is typically used to further propagate the transformed set of sigma points, $\mathcal{Y}^a_i$ (see [4]) with

$$\mathcal{Z}_{i,k-N} = \mathbf{h}(\mathcal{Y}^a_{i,k-N}, 0), \tag{3.17}$$

$$\bar{\mathbf{z}}^{-}_{k-N} = mean(\mathcal{Z}_{i,k-N}), \tag{3.18}$$

where $i \in \{1, \cdots, 2n\}$, and $\mathbf{h}$ is the observation VOS process model function (see Appendix E.0.2). Note: There is no added noise in Equation (3.17) since the sigma points encompass this.

A very interesting issue arises in the Atlas UKF at this point with regards to the observation function. Since the VOS process will ultimately measure the orientation quaternion directly, the *lagged* observation function $\mathbf{h}(.)$ can simply be treated as the identity matrix, thereby always using the corresponding 'k-N' 'a-priori' IOS state estimate as the expected VOS measurement. This allows for speed savings with minimal loss to the end estimate. The prediction, can quite clearly be the 4×1 'a-priori' *orientation quaternion* portion, $\bar{\mathbf{q}}^{-}_{k-N}$, of the 'k-N' 'a-priori' state estimate, $\bar{\mathbf{x}}^{a-}_{k-N}$, using

$$\bar{\mathbf{z}}^{-}_{k-N} \quad = \quad \bar{\mathbf{q}}^{-}_{k-N}. \tag{3.19}$$

2. Similarly, the 3×3 *lagged* VOS measurement state covariance is simply the 'k-N' IOS 'a priori' state process covariance, (*attitude parts only*):

$$\mathbf{P}^{-}_{zz,k-N} \quad = \quad \mathbf{P}^{-}_{xx,k-N,(1:3,1:3)}, \tag{3.20}$$

   where $\mathbf{P}^{-}_{zz,k-N}$ is the uncertainty in the measurement caused by the uncertainty in the state vector prediction which correspond to 'k-N' timesteps in the past.

3. Again for speed savings, the 'k-N' propagated sigma point orientation axis-angle differences, $\mathbf{\Phi}_{k-N,i}, i \in \{1, \cdots, 2n\}$, can also be held in memory. These can be re-used to create the cross-correlation-covariance-over-time matrix, $\mathbf{P}_{\mathbf{x}_k \mathbf{z}_{k-N}}$, calculation between the current system state at timestep 'k', and the *lagged* 'a-priori' observation estimate corresponding to timestep 'k-N', using

$$\mathbf{P}_{\mathbf{x}_k \mathbf{z}_{k-N}} \quad = \quad \frac{1}{2n} \sum_{i=1}^{2n} [\mathcal{Y}^{a}_{i} - \bar{\mathbf{x}}^{a-}_{k}] \cdot [\mathbf{\Phi}_{k-N,i}]^{T}. \tag{3.21}$$

   Note: The equivalent axis-angle rotation vector differences are used here. Also $[\mathcal{Y}^{a}_{i} - \bar{\mathbf{x}}^{a-}_{k}]$ is $[30 \times 1]$ in size whereas, $[\mathbf{\Phi}_{k-N,i}]$ is $[3 \times 1]$ in size. This creates a $[30 \times 3]$ cross-covariance matrix.

4. Next, The *innovation covariance*, $\mathbf{P}_{vv,k-N}$, is calculated as the sum of the projected measurement state vector covariance $\mathbf{P}^{-}_{zz,k-N}$ and the measurement noise covariance $\mathbf{R}_{VOS,k-N}$ using

$$\mathbf{P}_{vv,k-N} = \mathbf{P}^{-}_{zz,k-N} + \mathbf{R}_{VOS,k-N}. \tag{3.22}$$

5. The Kalman Gain can now be expressed in terms of the correct covariance terms using

$$\mathbf{K}_{k,N} = \mathbf{P}_{\mathbf{x}_k \mathbf{z}_{k-N}} \cdot (\mathbf{P}_{vv,k-N})^{-1}. \tag{3.23}$$

6. Typically, the 'a posteriori' state estimate, $\bar{\mathbf{x}}^{+}_{k}$, can be calculated at this point from the sum of the current 'a-priori' predicted state plus a correction factor formed from the *lagged innovation*, $\tilde{\mathbf{z}}_{k-N}$, and the Kalman gain term:

$$\bar{\mathbf{x}}^{+}_{k} = \bar{\mathbf{x}}^{-}_{k} + \mathbf{K}_{k,N}(\tilde{\mathbf{z}}_{k-N}). \tag{3.24}$$

   However, in this case, since a quaternion is used, contrary to Equation (3.16), the *lagged innovation* must be formed instead using the corresponding quaternion rotation that rotates $\bar{\mathbf{q}}^{-}_{vos,k-N}$ into $\mathbf{q}_{vos,k-N}$ using formulation described in Equation (2.10). Reproduced here, the multiplication from the measured *orientation quaternion* is

$$\tilde{\mathbf{q}}_{innov,k-N} = \mathbf{q}_{vos,k-N} \otimes (\bar{\mathbf{q}}^{-}_{vos,k-N})^{-1}. \tag{3.25}$$

7. Next, the axis-angle form $\mathbf{\Theta}_{innov,k-N}$ is obtained from $\tilde{\mathbf{q}}_{innov,k-N}$ using Equation (2.11) outlined in Section 2.3. This is followed by multiplying by the Kalman

gain term to obtain a *Kalman correction vector*, **Corr**, of size 30×1 using

$$\mathbf{Corr} = \mathbf{K}_{k,N} * \mathbf{\Theta}_{innov,k-N}, \tag{3.26}$$

where the first 3 components of the *Kalman correction vector*, namely $\Delta\mathbf{\Theta}_{corr}$, become re-transformed back into a correcting rotation quaternion, $\mathbf{q}_{corr}$, as follows:

$$\bar{\mathbf{q}}_k^+ = \mathbf{q}_{corr} \otimes \bar{\mathbf{q}}_k^-. \tag{3.27}$$

Note: The remaining 27 *Kalman correction factors* are used as per usual vector addition to update the sensor error states, 12 total, and state noise terms, 15 total.

8. Finally, the updated 'a-posteriori' covariance can be formed using the 'a-priori' predicted state covariance minus the innovation covariance, weighted by the Kalman gain using

$$\mathbf{P}_{x,k}^+ = \mathbf{P}_{x,k}^- - \mathbf{K}_{k,N}\mathbf{P}_{vv,k-N}\mathbf{K}_k{}^T. \tag{3.28}$$

The entire Atlas UKF outlined above was formulated into a Matlab function '*UKF.m*' and is provided in Appendix C.2. Figure 3.2 visually depicts the Atlas UKF estimation of the Atlas orientation. **Note:** for clarity, all covariance terms, iterative sigma point propagation terms, quaternion and vector averaging, and IOS sensor error estimation terms have been left out of the diagram.

Note: A second version of the Atlas UKF allows instantaneous IOS quaternion measurements to be used namely *UKF2.m* (see Appendix C.4).

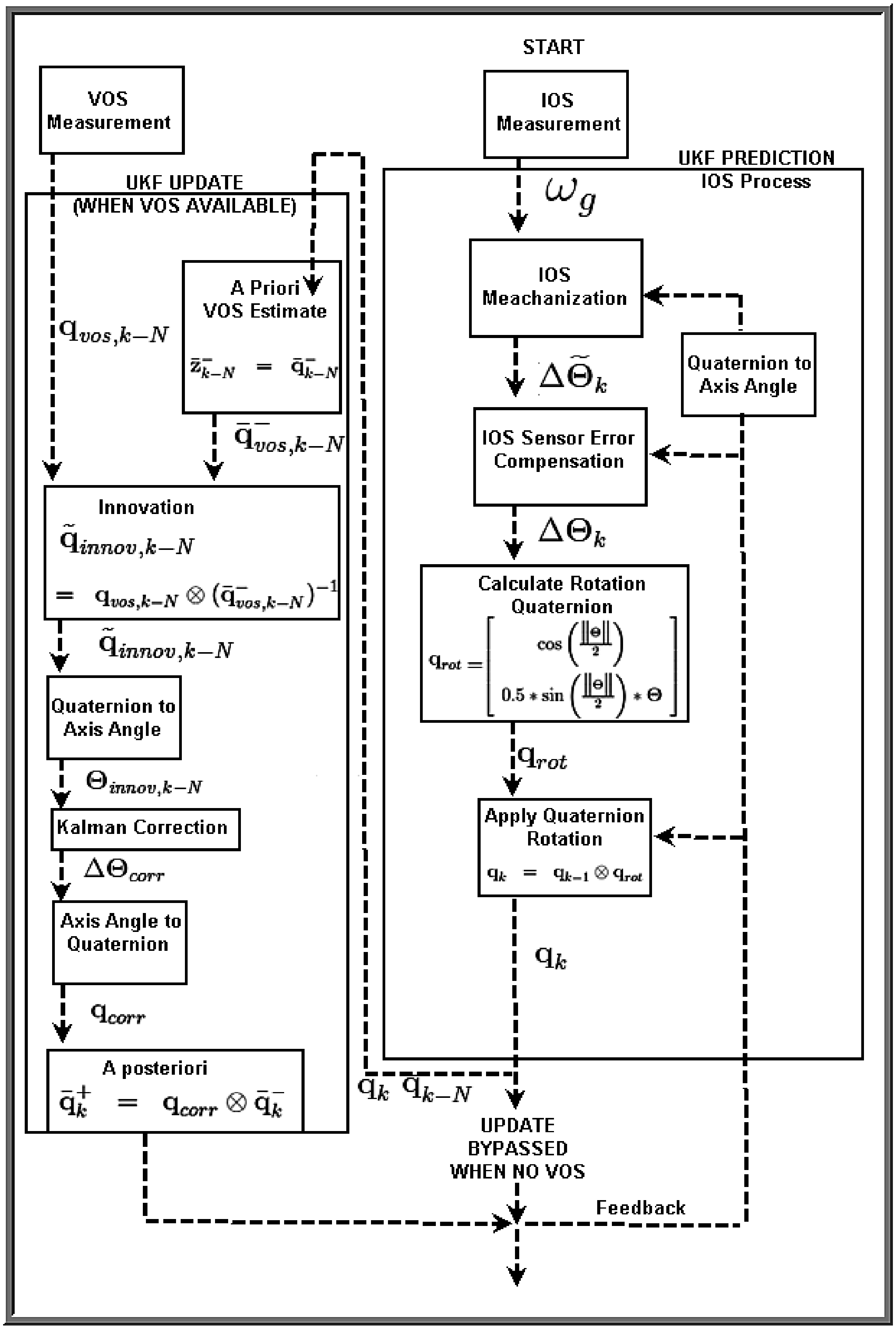

START
VOS Measurement
IOS Measurement
UKF UPDATE (WHEN VOS AVAILABLE)
UKF PREDICTION IOS Process
ω_g
IOS Meachanization
A Priori VOS Estimate
$\bar{z}^-_{k-N} = \bar{q}^-_{k-N}$
$q_{vos,k-N}$
$\Delta\widetilde{\Theta}_k$
Quaternion to Axis Angle
$\bar{q}^-_{vos,k-N}$
IOS Sensor Error Compensation
Innovation
$\widetilde{q}_{innov,k-N} = q_{vos,k-N} \otimes (\bar{q}^-_{vos,k-N})^{-1}$
$\Delta\Theta_k$
Calculate Rotation Quaternion
$q_{rot} = \begin{bmatrix} \cos\left(\frac{\|\Theta\|}{2}\right) \\ 0.5 * \sin\left(\frac{\|\Theta\|}{2}\right) * \Theta \end{bmatrix}$
$\widetilde{q}_{innov,k-N}$
Quaternion to Axis Angle
q_{rot}
$\Theta_{innov,k-N}$
Kalman Correction
Apply Quaternion Rotation
$q_k = q_{k-1} \otimes q_{rot}$
$\Delta\Theta_{corr}$
Axis Angle to Quaternion
q_k
q_{corr}
A posteriori
$\bar{q}^+_k = q_{corr} \otimes \bar{q}^-_k$
q_k q_{k-N}
UPDATE BYPASSED WHEN NO VOS
Feedback

Figure 3.2: The Atlas UKF flowchart of orientation estimation.

Chapter 4

Using the IOS within Matlab for Atlas UKF testing

For the purpose of testing the Atlas UKF, only attitude sensing is considered by using the 3DM-GX1 IMU measurements. The 3DM-GX1 includes three orthogonal MEMs accelerometers, three vector magnetometers, three angular rate gyros, and a thermocouple. It is capable of measuring static and dynamic rates with $360°$ range over all three axes (pitch, roll, & yaw).

The algorithms required to compute orientation over $360°$ on all three axes are embedded within the micro controller of each 3DM-GX1. Gyro-stabilized rates are obtained after being corrected using Microstrain's proprietary sensor fusion filter embedded within the 3DM-GX1. This compensates for gravitational and temperature issues. Details of this filter fusion algorithm are not available to the general public. For this reason, herein this thesis, the gyro-stabilized rates are treated as the measured gyro rates. Detailed specifications for the 3DM-GX1 are provided in Appendix B.4.

The 3DM-GX1 arrives from the manufacturer calibrated over the temperature range of $-20°C \rightarrow +70°C$. Correction coefficients needed for each and every sensor on the 3DM-GX1 are burned into the modules non-volatile memory. The 3DM-GX1 houses a small thermocouple with each gyro to compensate for temperature changes [29]. The 3DM-GX1 has its own internal clock which tracks time. All of the data acquisition functions, whether in the continuous mode or the polled mode, provide elapsed timer ticks in the data return. This capability provides the user with accurate time stamping independent of the computer operating system. All the constant manufacture 3DM-GX1 IMU specifications used within the Atlas UKF are written to a Matlab file, '*IMUspecs.m*', and can be found in Appendix B.5.

For the testing of the Atlas UKF, a Matlab program has been written to acquire data from the 3DM-GX1. As outlined in the 3DM-GX1 communications protocol [30], the 3DM-GX1 has the capability to transmit numerous quantities of interest. The 3DM-GX1's on-board processor operates with a 0.0065536 second clock tick interval. Wireless transmission differs from a direct wired communication link. The 3DM-GX1 will not transmit unsolicited data. In order to conduct experiments, a Matlab function is adapted and developed based on example code provided by Prime [31]. This adapted code, re-named '*3DMG_operate.m*', enables R2-232 communication with the IOS within Matlab. The written function enables Matlab to record IOS data for a chosen amount of time and plots and displays the results. This code is provided in Appendix C.8. Using this function, varying commands can be issued to have the 3DM-GX1 transmit any or all of its sensor measurements in any form. It is found, that while a direct RS-232 cable connection offers faster transmission rates, up to $\approx$ 153 Hz, wireless data is only transmitted at a maximum frequency half of this (76.2939 Hz). Due to this fact, presumably it is less suitable to use instantaneous IOS measurements since the maximum transmission rate is the same as receiving the stabilized IOS measurements. This fact is not detailed in any other Microstrain 3DM-GX1 literature. It may be possible for the full scale future Atlas simulator to utilize faster direct RS-232 cable instantaneous measurement quantities from the 3DM-GX1 by transmitting VOS measurements to an onboard computer internal to the sphere for data fusion. Due to a direct cable connection, this would allow for faster IOS measurements. Currently, having IOS transmitted using a paraniSD-100 Bluetooth adapter, the maximum rate that any quantity can be transmitted is two IOS clock ticks, which translates into a constant frequency of 76.2939 Hz. This is maintained throughout all tests conducted herein. It was also observed that no 'bad' or dropped packets were handled. Details for the use of the '*3DMG_operate.m*' function which is used to operate the IOS within Matlab are outlined here. The Matlab code can be found in Appendix C.8.

4.0.2 Operating the IOS within Matlab

In order to operate the IOS, the '*3DMG_operate.m*' function is written for Matlab to acquire IOS measurements for a set length of time. It may be possible to create a 'real-time' version of this function in the future, however, there exist functions in Labview for this purpose, although fidelity of these Labview functions is unknown. Instead, the Matlab function is written to easily allow the Atlas UKF performance to be analyzed offline. Key features to this developed Matlab function are as follows:

Communication The construction of a serial port object is formed for writing and receiving bytes from the IOS, an IOS command can be chosen by the user and will set the IOS into a continuous polling mode.

Translate Data Packets The data quantities obtained from the 3DM-GX1 are un-signed 16bit integers that get transformed into a 2's complement representation.

Error Handling There is no opportunity for bad or dropped packets since this will cause the function to return an error message. The last two un-signed data bits are used as checksums. Each data packet is analyzed for error and passed on.

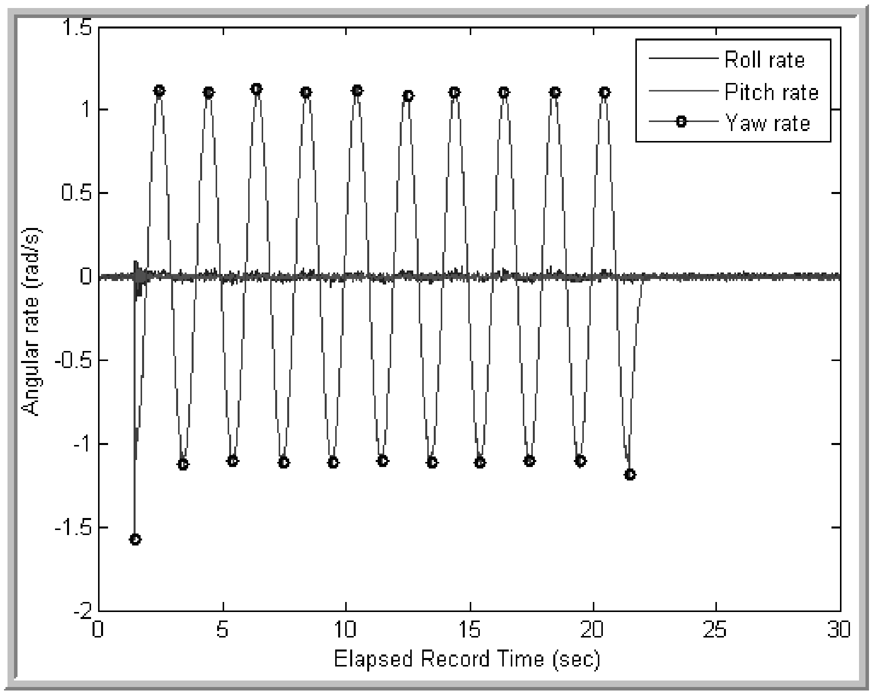
Scaling The data packets are scaled into the proper units corresponding with '*3DMG protocol v.3.1.01*' [30].

Closing connection The correct method for deleting a serial port object is performed.

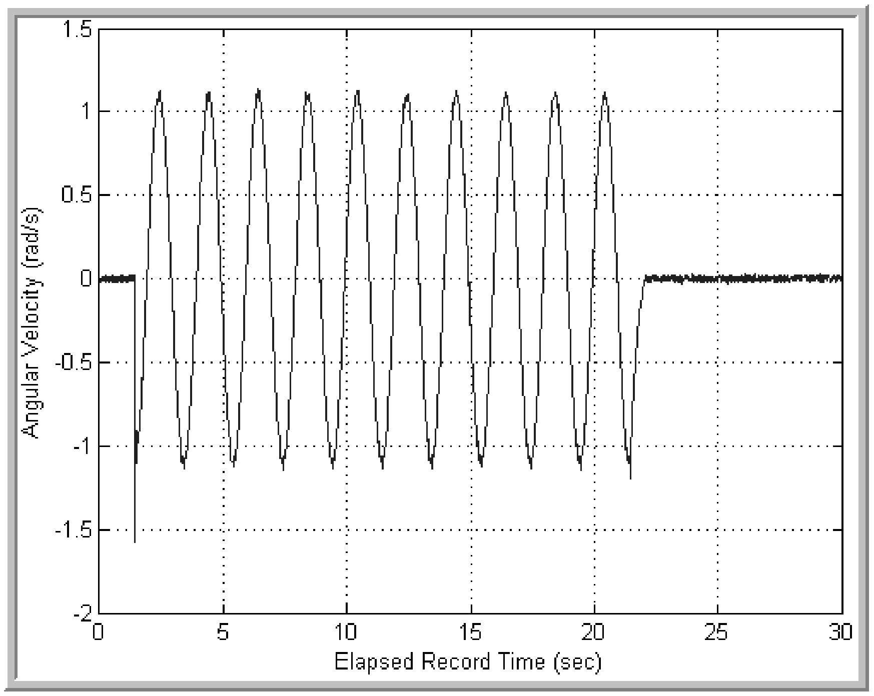
Visualization Once the connection is closed the recording is complete and the data is visualized in Matlab.

For now, only commands 2,4,5 and 12 have been fully implemented within '*3DMG_operate.m*'. These commands correspond to quantities of interest for the purposes of testing the Atlas UKF and are described further in Section 5.2.

Using command 2, Figure 4.1(a) shows the un-compensated individual IOS gyro stabilized angular rate measurements from the 3DM-GX1 for 30 seconds of back and forth sinusoidal rotation about the local IOS 'yaw' z-axis. Figure 4.1(b) shows these individual Euler angular rate measurements formed together into one corresponding angular rate measurement about an axis of rotation. The angular rate magnitude measurement includes errors from all three individual gyros. It should be noted that obtaining the magnitude measurement requires that care be taken to include the direction of rotation from each gyro rate which involves retaining the sign. The Matlab function '*sign_mag.m*'(see Appendix D.8) is created to calculate this magnitude for the three angular rates while maintaining the sign for the direction of rotation. This function is used for visualization of the un-compensated IOS angular rate magnitude only. During the actual Atlas UKF operation, the *sign_mag.m* function is not required since the IOS process handles individual IOS Euler rates as inputs for performing integration and creates the subsequent quaternion from the individual Euler angles.



(a) Individual IOS Euler angular rates.



(b) Angular rate magnitude.

Figure 4.1: 3DM-GX1 gyro stabilized angular rates for 30 seconds of pure yaw

Chapter 5

Simulation Testing of the Atlas UKF and Measured Data Results

For testing the Atlas UKF, a simulation is developed that creates simulated measurements for both the IOS and VOS based on expected noise on each state. The simulated measurements with noise stay within reasonable limits of the Atlas platform operating conditions. The current angular velocity and acceleration performance specifications for the Atlas sphere are listed in Table (5.1). These specifications are maintained when testing simulations of varying rotational trajectories.

Parameter	Description	Value	Units
Ω	Maximum Angular Velocity	1	$\frac{rad}{s}$
$\dot{\Omega}$	Maximum Angular Acceleration	8.8	$\frac{rad}{s^2}$

Table 5.1: Atlas full scale performance specifications.

5.1 Simulation Tests

Using MATLAB, a simulation has been created to test the quaternion based Atlas UKF with sensor error estimation against two other forms of fusion filter: the *Simple Sensor Fusion*(SSF) algorithm and the *non-augmented quaternion based UKF without* *sensor error estimation* (UKF_NoSE) [28]. Unlike the SSF algorithm developed by Brian Rasquinha [32], the SSF algorithm used in this thesis utilizes quaternions and incorporates OOSM handling (see Appendix C.9). Along with the ability to test various sensor frequencies, any non-linear rate profile can be tested within this Matlab simulation program (see Appendix C.14). The simulation is used to contaminate a proposed non-linear angular rate profile by adding random Gaussian noise based on sensor specifications for each simulated IOS and VOS sensor measurement over the operating time designated.

5.1.1 Simulated Noise

During simulation, both IOS and VOS measurements are created by purposely contaminating an actual rotation trajectory with noise based on the covariance of each measured state. Initial covariance of each state is based on manufactured specifications using standard deviations where available. During simulation, the covariance is used to create contaminated measurements. This is accomplished with a computer generated random number between 0 and 1 which is used to create scaled noise based on the standard deviations of each state. Next, the simulated noises are added within the IOS process model and VOS process model onto each state. This creates the simulated noisy measurements for each sensor. The simulated measurements subsequently get passed on to the UKF to estimate the Atlas orientation.

IOS Process Noises

Using the angular random walk (ARW) value of $3.5\frac{\circ}{\sqrt{hr}}$ based on a 15 second Allan variance method which is provided by Microstrain [26], the IOS angular position standard deviation for a short duration timestep '*dt*' is calculated as

$$\begin{aligned}
ARW &= \frac{3.5 * \pi}{180 * \sqrt{3600}} (\frac{rad}{\sqrt{s}}), \\
\sigma_\omega &= 60 * ARW * \sqrt{BW_{ins}} (\frac{rad}{s}), \\
&= \frac{\frac{\frac{3.5*\pi}{180}}{\sqrt{\frac{dt}{3600}}}}{3600} (\frac{rad}{s}) \\
and & \qquad\qquad\qquad\qquad\qquad\qquad (5.1) \\
\sigma_\theta &= ARW * \sqrt{dt}\,(rad),
\end{aligned}$$

$$(5.2)$$

where '*dt*' is the discrete timestep based on the current IOS selection (i.e. typically 0.0131 *sec*).

The simulated Euler angle measurement standard deviations and the sensor error terms standard deviations can be used to create the white gaussian noise (WGN) variables which get added to the actual states to be estimated to simulate an IOS angular velocity measurement as outlined in Section 3.2, Equation (3.2). As such, the standard deviations are

$$\begin{aligned}
\mathbf{v}_e &= \sigma_\theta * randn[3 \times 1], \\
\mathbf{v}_{b_g} &= \sigma_{b_g} * randn[3 \times 1], \\
\mathbf{v}_{s_g} &= \sigma_{s_g} * randn[3 \times 1], \\
\mathbf{v}_{ma_g} &= \sigma_{ma_g} * randn[6 \times 1],
\end{aligned} \tag{5.3}$$

with

$$\widetilde{\mathbf{\Theta}}_k = (\mathbf{I} + \mathbf{G}_{g,k-\frac{1}{2}}) * \mathbf{\Theta}_k + \mathbf{b}_{g,k-\frac{1}{2}} \Delta T + \mathbf{v}_{e,k}. \tag{5.4}$$

Similarly, the IOS process noise covariance matrix is initially created using the standard deviations

$$\mathbf{Q}_{IOS,[15\times15]} = \begin{bmatrix} \sigma^2_{\theta_x} 0 0 0 0 0 0 0 0 0 0 0 0 0 0 \\ 0 \sigma^2_{\theta_y} 0 0 0 0 0 0 0 0 0 0 0 0 0 \\ 0 0 \sigma^2_{\theta_z} 0 0 0 0 0 0 0 0 0 0 0 0 \\ 0 0 0 \sigma^2_{b_{g,x}} 0 0 0 0 0 0 0 0 0 0 0 \\ 0 0 0 0 \sigma^2_{b_{g,y}} 0 0 0 0 0 0 0 0 0 0 \\ 0 0 0 0 0 \sigma^2_{b_{g,z}} 0 0 0 0 0 0 0 0 0 \\ 0 0 0 0 0 0 \sigma^2_{s_{g,x}} 0 0 0 0 0 0 0 0 \\ 0 0 0 0 0 0 0 \sigma^2_{s_{g,y}} 0 0 0 0 0 0 0 \\ 0 0 0 0 0 0 0 0 \sigma^2_{s_{g,z}} 0 0 0 0 0 0 \\ 0 0 0 0 0 0 0 0 0 \sigma^2_{ma_{g,xy}} 0 0 0 0 0 \\ 0 0 0 0 0 0 0 0 0 0 \sigma^2_{ma_{g,xz}} 0 0 0 0 \\ 0 0 0 0 0 0 0 0 0 0 0 \sigma^2_{ma_{g,yx}} 0 0 0 \\ 0 0 0 0 0 0 0 0 0 0 0 0 \sigma^2_{ma_{g,yz}} 0 0 \\ 0 0 0 0 0 0 0 0 0 0 0 0 0 \sigma^2_{ma_{g,zx}} 0 \\ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 \sigma^2_{ma_{g,zy}} \end{bmatrix}. \tag{5.5}$$

VOS Process Noises

Within the Matlab function *VOSparameters.m*, standard deviations for pixel location in image coordinates, as well as the Atlas sphere marker placement standard deviations based on the CMM are used to create the VOS measurement covariance matrix. In this matrix, both CMM calibration inaccuracies as well as image processing inaccuracies are represented, to complete the current overall model of the VOS [33]. During simulation, using the standard deviations, simulated noise terms are created to purposefully contaminate the Atlas orientation camera images. The VOS process ('*VOSprocess.m*') is re-written in a condensed form for clarity in this thesis and is available in Appendix C.7.

As outlined in the '*VOSparameters.m*' and the '*MakeRvos.m*' function, a general $[3 \times 3]$ axis-angle VOS measurement noise covariance, $\mathbf{R}_{VOS}$, is created by encompassing both the image pixel error and the sphere marker location error. Details for creation of this matrix are slightly more complex and outside the overall focus of this thesis, however, they are available in Appendix C.7.

The results from simulation reveal good feasibility of the UKF for attitude estimation through a non-linear trajectory and lead to further testing using actual measured data from the IOS.

5.2 Dynamic Test of the UKF using measured IOS Data

The simulation program is modified and made into a Matlab program to take actual measured IOS data and fuse with simulated VOS measurements. In order to test the true dynamic performance of the UKF, a highly precise Apex Dynamics AD047 encoded gearbox is used to rotate the 3DM-GX1 through known rate profiles.

Measurements from the 3DM-GX1 and gearbox encoder measurements of the global angular position are recorded. Suitability of the gearbox for use in testing the IOS and UKF is investigated in this section. Manufacture specifications for the gearbox are provided in Appendix B.6.

The objective of the true data tests is to manipulate and record the IOS motion through a known, non-linear, angular rate profile, in an attempt to excite possible sensor errors and cause drift in the IOS measurements. Using the gearbox/IOS setup, two different types of tests are conducted.

5.2.1 Description of Tests

A series of tests, short in duration are run to gain a general sense if the UKF will work and also to determine the best IOS command to use. After this initial testing, a second series of tests, longer in duration are run to increase the possibility of misalignment errors in the IOS gyro-stabilized measurements and demonstrate the filter remains convergent for extended periods of time while undergoing dynamic rotational motion.

30 second test: Matlab is used to simultaneously record (motor and IOS) back and forth pure yaw rotation following a sine function rate profile about the local 3DM-GX1 z-axis, $[0, 0, -1]^T$. This test is performed 3 different times for each IMU command output for 30 seconds.

- Using a command of 4 in '*3DMG_operate.m*', the instantaneous orientation quaternion is transmitted.
- Using a command of 5 in '*3DMG_operate.m*', the instantaneous gyro stabilized quaternion is transmitted.
- Using a command of 2 in '*3DMG_operate.m*', the gyro stabilized angular rate vector is transmitted.

120 second test: The second test conducted is a two minute non-linear rate profile as listed in Table 5.2. The true gearbox encoder measurement rotation angle is shown in Figure 5.1 for this rate profile. Since the encoder measures absolute rotation from the initial gearbox starting orientation, the data is modified to stay within $360°(2\pi$ radians), see Figure 5.2. The 3DM-GX1 is attached initially rolled $45°$ about the local 3DM-GX1 x-axis while maintaining that the motor's axis of rotation is pointed through the 3DM-GX1's origin. This configuration assumes a constant rotation axis pointed along the IMU's $[0, 1, -1]^T$ axis. A manual attempt is made to minimize the initial 3DM-GX1 mounting configuration misalignment. Manual mounting corresponds well to what can be expected from a student installment within the Atlas sphere. No attempt is made to quantify the initial misalignment since this is presumed to be representative of a typical mounting. A 2-minute test is performed a total of 9 times, (three for each different IMU command) as follows.

- Using a command of 5 in '*3DMG_operate.m*', the instantaneous gyro stabilized quaternion was transmitted.
- Using a command of 2 in '*3DMG_operate.m*', the gyro stabilized angular rate vector was transmitted.
- Using a command of 12 in '*3DMG_operate.m*', both the gyro stabilized instantaneous quaternion and gyro stabilized angular rate vector was transmitted

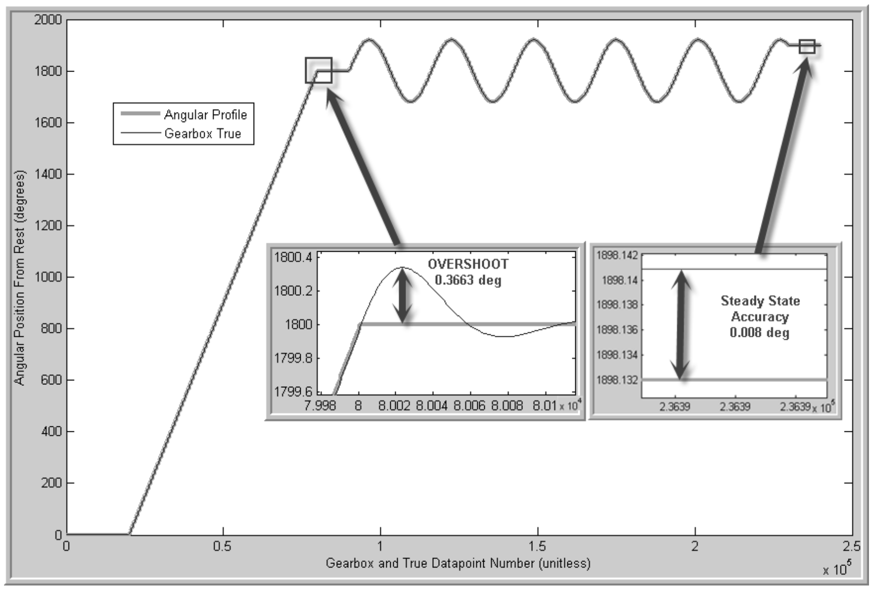


Figure 5.1: The gearbox motor rotation angle vs. the angular profile (deg) for the **120 second test**.

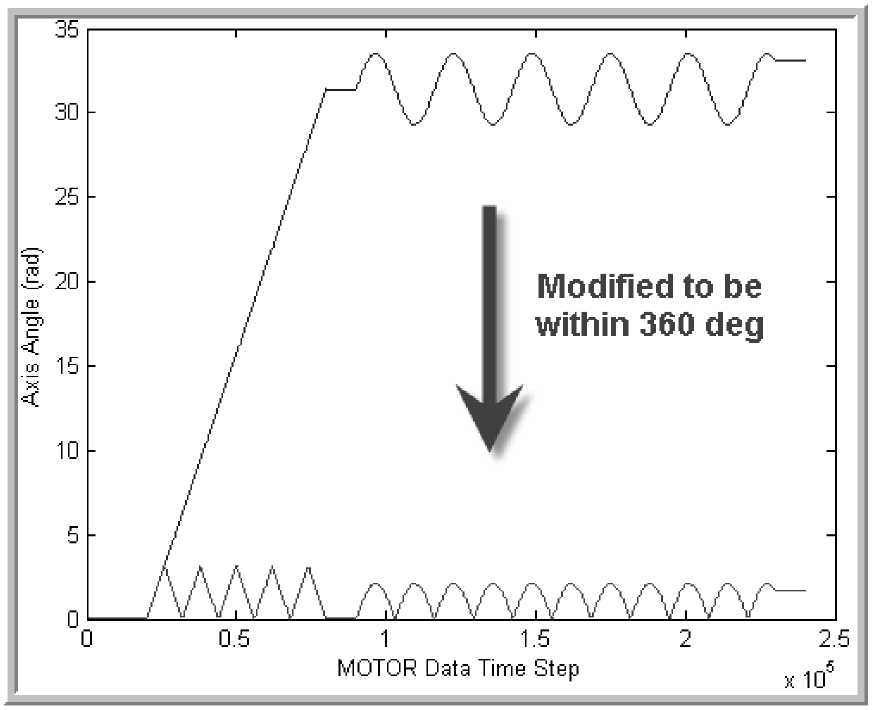


Figure 5.2: The modified true encoder motor rotation angle (rad) within 360° (2π radians).

Stage	$\lvert\Omega(t)\rvert(^\circ/sec)$	$\Delta(t)$(sec)	Comments
1	rest	60	hold in position
2	*constant* 60	30	5 complete rotations about IMU axis
3	rest	5	hold in position
4	*non-constant* $57.6\cos(0.48t)$	70	*sinusoidal* 120° back and forth rotation
5	rest	5	hold in position

Table 5.2: The angular rate profile for one 120 second dynamic test.

During Stage 2, the gearbox maintains a constant angular rate of $10\frac{rev}{min}$ ($60°/sec$ or $1.05\frac{rad}{sec}$) for 5 rotations.

During Stage 4, the gearbox produces non-constant angular rates, reaching a maximum rate of $9.6\frac{rev}{min}$ ($57.6°/sec$ or $1.01\frac{rad}{sec}$) for a sinusoidal angular rotation profile.

At such a low speed, the gearbox measurements are found to follow the designed angular position path within acceptable limits: the RMS error for one entire 120 second test was only 0.0004614° with the maximum error 0.3663°. This error is largely due to overshoots and undershoots between each stage. The gearbox records rotation data at 2KHz which is deemed acceptable for testing the algorithm. It is subsequently treated as the **true** rotation profile path followed for all filter test results. A Matlab file for calculating the RMS gearbox error can be found in Appendix C.11.

5.2.2 Choosing an IOS Command

With no theoretical basis for choosing one method of measurement over another, a process of elimination was used to decide on the best IOS command. Shown in Figure 5.6, the stabilized angular rate measurements from the 3DM-GX1 correspond closely to the true motor angular rate measurements before the UKF is run. Acquiring this data corresponds to using *Command 2* for the stabilized angular rate vector. When using *Command 4*, IOS gyro instantaneous quaternion without stabilization, large spikes of noise are seen in the data compared to using *Command 5*, instantaneous gyro-stabilized quaternion which demonstrates less noise in Figure 5.4.

Only *Command 2*, *Command 4* and *Command 5* are implemented for use in the UKF with sensor error estimation at this time.

For handling instantaneous orientation quaternions, within the IOS process, the IOS mechanization is modified slightly since it avoids having to use discrete integration. Instead, the incremental rotation vector, $\widetilde{\boldsymbol{\Theta}}_k$, is obtained by using Equation (2.10) and Equation (2.11). Details are found within the Matlab file *IOSprocess2.m* within *UKF2.m* in Appendix C.4.

When using *Command 2* with IOS angular rates, *IOSprocess.m* in *UKF.m* is utilized. Details of these functions are found in Appendix C.2 and they follow the approach developed in Section C.3.

Figure 5.3 shows the error in estimating the equivalent rotation vector magnitude, found when measuring instantaneous quaternion with *Command 5*. This is compared to Figure 5.8 using angular rates, *Command 2*, for 30 seconds. The instantaneous measurements display noisier uncompensated data but follow the true path more closely. The stabilized quaternion orientation measurement shows less noise but deviates away from the initial start position.

Summarized in Table 5.3, results of the 30 second testing show using direct gyro-stabilized angular rate measurements *Command 2* from the 3DM-GX1 produce the most accurate UKF estimation. It is presumed that offline discrete integration in Matlab yields a more accurate measurement and that this is due to the higher precision that the computer can handle when compared to the 3DM-GX1 on-board processor. This may also be due to less initial noise entering the gyro measurement compared to the 3DM-GX1 on-board integration process performed using *Command 5*.

	Average RMS Value (deg)
CMD 2	0.5443
CMD 5	1.8507
CMD 4	1.9767

Table 5.3: The RMS axis-angle true error after UKF, VOS 20 Hz, 30 second test.

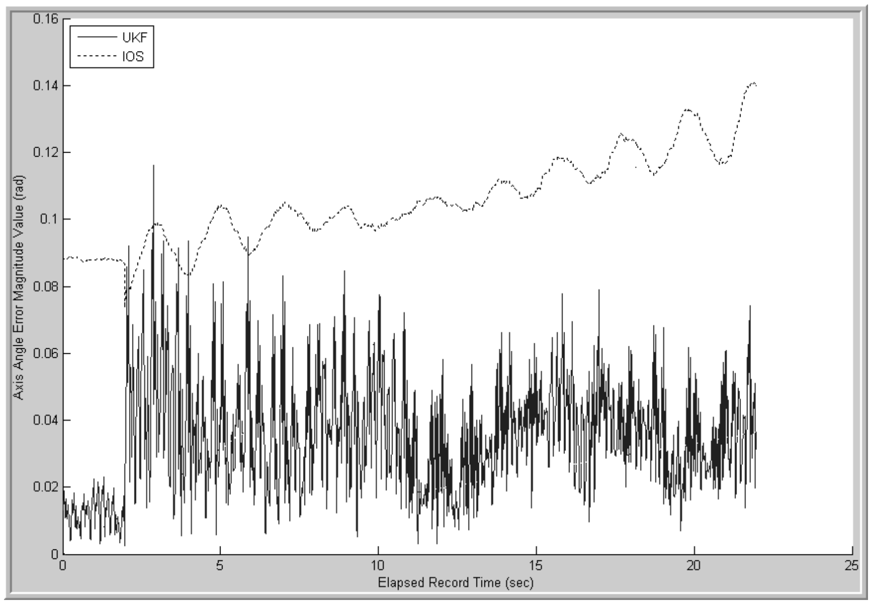
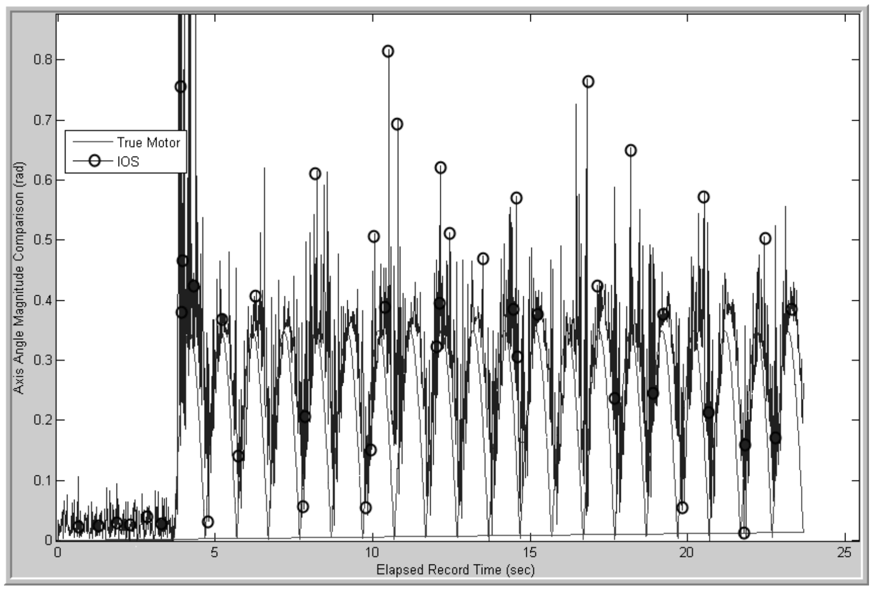
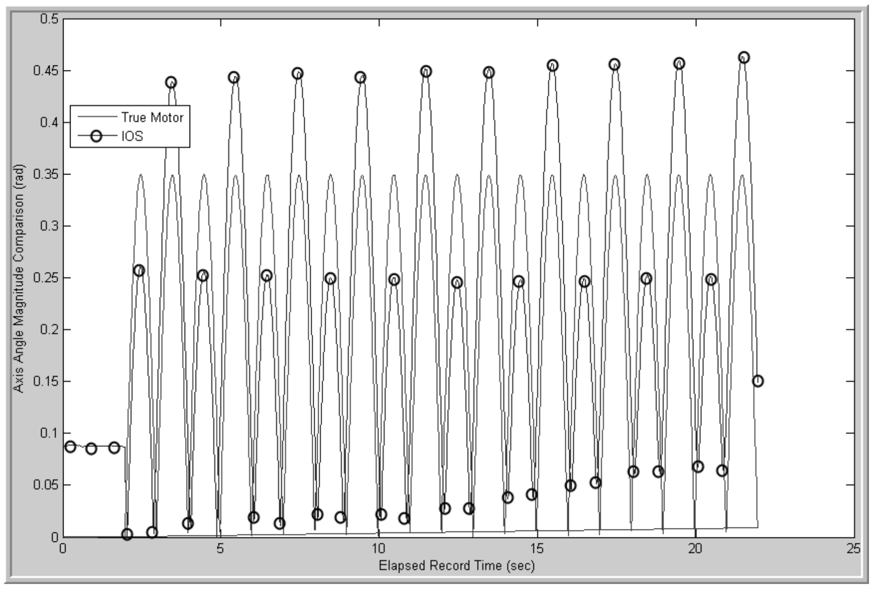


Figure 5.3: Depiction of the error in estimating the equivalent rotation vector magnitude estimate from the estimated quaternion, using *Command 5*, for a 30 second test.



(a) The transformed instantaneous quaternions made into an equivalent axis-angle magnitude value, using *Command 4*.



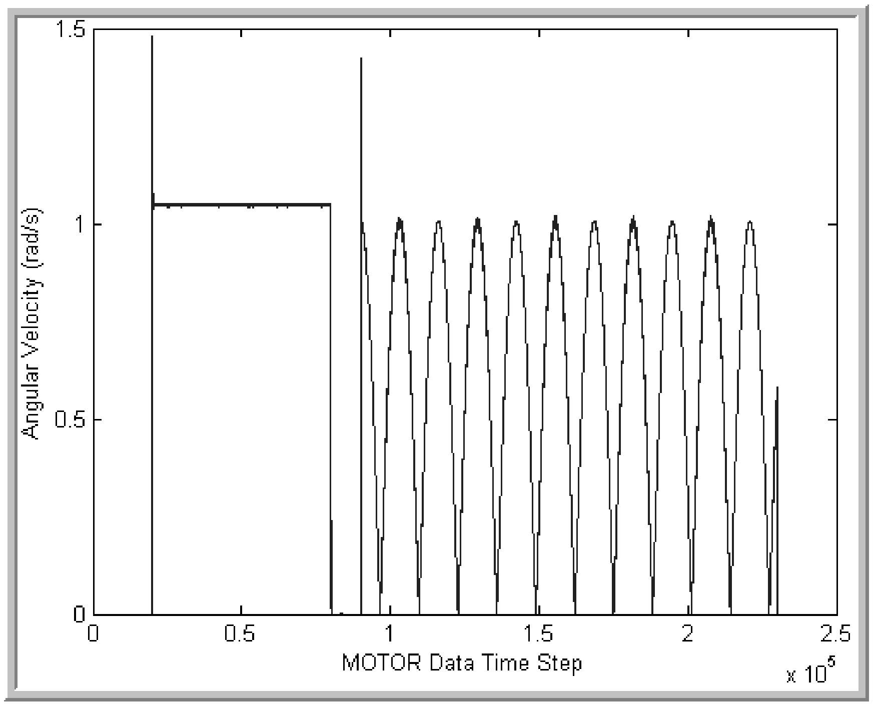
(b) The transformed gyro stabilized instantaneous quaternions made into an equivalent axis-angle magnitude value, using *Command 5*.

Figure 5.4: A comparison of 30 seconds of measured IOS uncompensated data vs. measured true path motor data, with various 3DM-GX1 Commands.

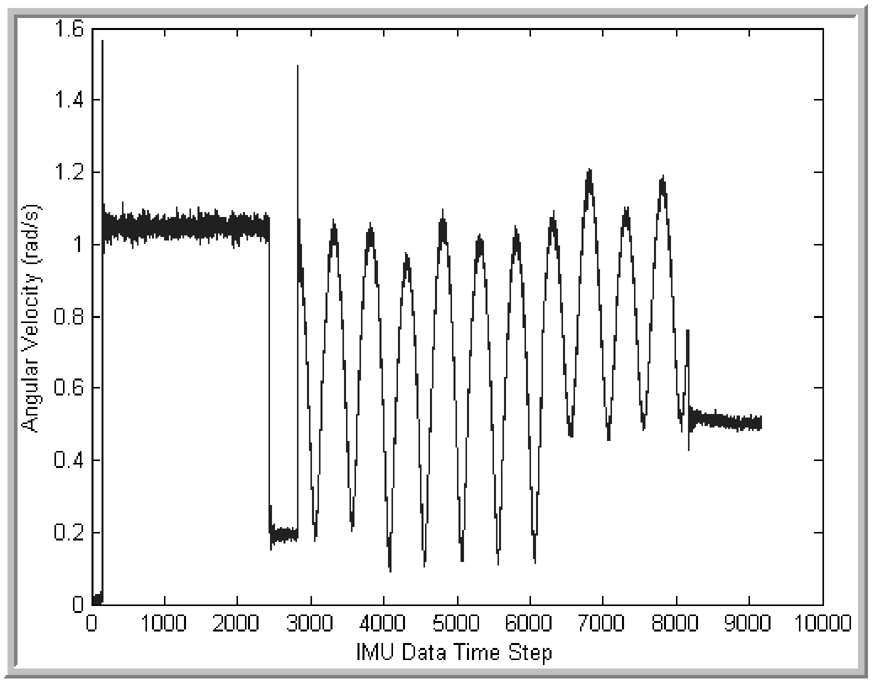
5.2.3 Data Synchronization

Synchronization of true motor path data and IOS data is achieved using with a Matlab function, *convertmotordata.m* found in Appendix (C.13).

Using the motor encoder, the 'true' angular positions are measured at a frequency of $2KHz$. This 'true' data set is then differentiated to yield the 'true' angular rate profile that is followed by the IMU. Using Matlab, plots of the axis-angle angular velocity magnitude is used to synchronize the angular rate data sets recorded from both the IMU and motor shown in Figures (5.5) and (5.6).



(a) The motor angular rate vector magnitude.



(b) The IOS angular rate vector magnitude.

Figure 5.5: Comparison of true motor data vs. IOS angular rate vector magnitude for one 120 second test.

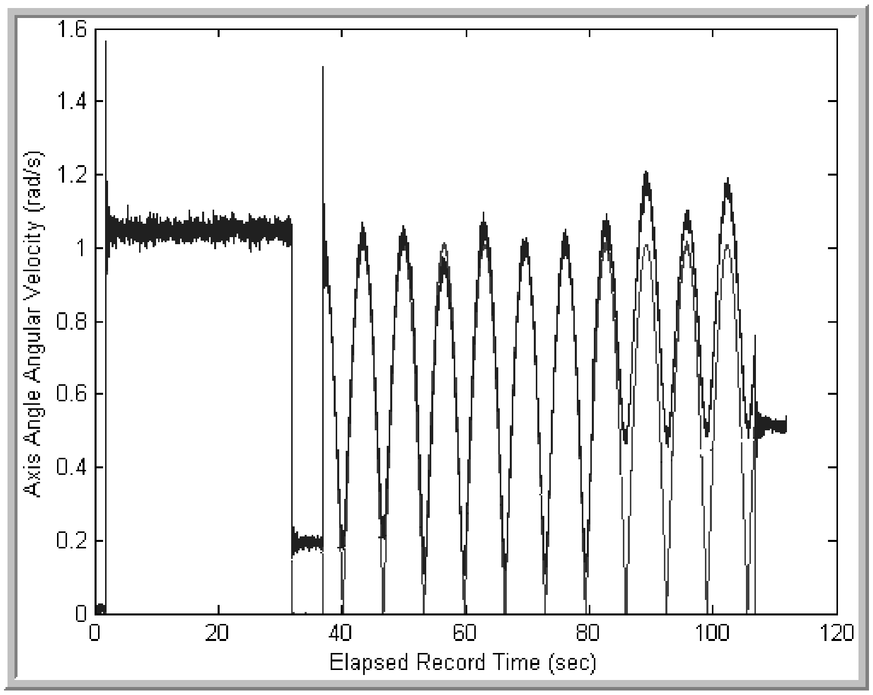


Figure 5.6: Re-aligned IOS angular rate magnitude data synchronized to motor data for one 120 second test

Currently, the VOS is non-operational due to colour recognition issues. Because of this, the decision was opted to simulate VOS measurements using the true motor data and simulate noise corrupting the VOS measurement based on proposed camera parameters. Each simulated VOS orientation measurement is created using a perspective camera model which uses the expected identification of markers in the image at that orientation. The simulated VOS measurement is modified to further incorporate an expected delay. Together this fully simulates out of sequence measurements caused by image processing time. These are made to mimic any frequency up to the faster IMU frequency and are passed to the UKF presumably just as actual VOS measurements eventually will.

5.3 Secondary filters used for comparison

Aside from using only the developed 'augmented' UKF with sensor error estimation developed herein this paper, two other filter forms are modified and used for test comparison; SSF, and UKF_NoSE (see Appendix C.9 and C.10). The two filters are further described in the following subsections.

5.3.1 Simple Sensor Fusion (SSF)

The simple sensor fusion (SSF) algorithm, initially developed by Carleton CUSP, fourth year student Brian Rasquinha at Carleton University [32], originally only dealt with Euler orientation representations and was tested solely on stationary IMU data. The SSF algorithm has been updated in this research to take angular rate measurements from the IMU, integrate to get an orientation quaternion and correct for the drift due to integration by resetting/correcting the IOS quaternion measurement based on the difference between the IOS and VOS orientation measurements. The code incorporates a method for dealing with out of sequence VOS measurements.

The SSF code makes no attempt to optimally fuse any of these measurements using statistical information, instead the code assumes the VOS measurement is always a completely accurate measurement. The underlying concept with this is to obtain a correction factor which is the difference between the IOS orientation and VOS orientation. Each IOS measurement between subsequent VOS measurements is modified with the correction factor obtained using

$$Corr_k = IOS_{k-N} - VOS_{k-N}, \tag{5.6}$$

$$Fused_k = IOS_k - Corr_k, \tag{5.7}$$

where N represents the number of discrete timesteps a measurement is lagged by. This measurement becomes available to the function at timestep k.

The modified SSF Matlab code, '*Simple_sensor_fusion.m*', which incorporates quaternions is found in Appendix (C.9). This code is run in parallel to the UKF with sensor error estimation and the UKF without sensor error estimation to get an overall sense of each method's effectiveness and also to reveal the benefits of using one filter over the others. The above experiments are performed using exactly the same IOS and VOS measurements to keep comparisons similar.

5.3.2 UKF with No Sensor Error Compensation (UKF_NoSE)

A UKF developed by Kyle Chisholm [28], incorporates no sensor error estimation and assumes additive noise opting for use of a non-augmented version [28]. To account for OOSM, the *UKF_NoSE* modifies the *lagged* VOS state measurements forward in time in an attempt to utilize the Kalman gain correction factor during each iteration. While having the benefit of smoothing the estimate between each time-step, the downside of this implementation is the added complexity from having to create a modified VOS measurement at every time-step. Also, un-accounted for error between two IOS estimates is less optimal than fusing an actual *lagged* VOS measurement with the actual IOS measurement corresponding to the same timestep [7]. Nevertheless, for purposes of demonstrating improvement that can be achieved by estimating sensor error terms, results using the *UKF_NoSE* are included as a filter for comparison. The *UKF_NoSE.m* Matlab file is also included in Appendix (C.10).

5.4 Filtering Results

True measured synchronized motor encoder data, along with measured and aligned IOS data is fed through each of the three filters offline using the function in Matlab, *Run_simulation.m*, found in Appendix (C.14). It is found that the most accurate orientation estimate occurs using the UKF with sensor error estimation. Test results indicate that use of the UKF with sensor error estimation maintains estimates within 3 standard deviations thereby allowing it to remain a viable option for estimating the Atlas orientation. Atlas specification for orientation control is $\pm 1^\circ$ accuracy [19]. For one 120 second dynamic test, the maximum error found exceeded this at 3° but had a RMS error of 0.83°. It is presumed that longer test runs, through multiple trajectories, would ultimately improve the misalignment term estimates used by the UKF, thereby improving the estimation further. Without a method for testing, it is difficult to state with any certainty whether the UKF is successfully estimating the misalignment G matrix terms properly. According to Lam "successful estimation normally requires multiple rate profiles whose individual rate magnitudes need to meet a certain minimum requirement for state observability" [11]. Although this is attempted, the fact remains that the rate profiles tested may not completely excite all the misalignment errors sufficiently for the UKF filter to fully observe them.

The full performance testing requires having the VOS with the Atlas motion platform completely operational for longer periods of time and conducted over multiple rate profiles. It stands to reason that such tests will construct better sensor error term estimates as well as allow for further tuning of the covariance $\mathbf{Q}_{IMU}$ and $\mathbf{R}_{VOS}$ matrices. Ultimately, this will further improve the UKF filter. However, without equipment that has the ability to simultaneously rotate and accurately measure the Atlas sphere orientation, over a rotation rate profile incorporating more than one axis, it is difficult to determine the complete extent to which the UKF can estimate sensor error terms.

5.4.1 30 second test results

Although short in duration, the 30 second test performing cyclic yaw motion confirmed the importance of eventually estimating sensor error terms online. SSF is computationally more efficient when considering near stationary rotations, however, once relatively small misalignment errors become present, degraded estimates become increasingly apparent as time unfolds. Figure 5.7 shows the effect of fusing the VOS and IOS into an overall axis-angle magnitude estimate created from each quaternion estimate. Figure 5.8 shows the orientation estimate's error magnitude away from the true recorded motor axis-angle magnitude. Figures 5.9 and 5.10 demonstrate the Euler angle estimation error for each IMU axis with the 3 standard deviation bounds. Results show, the 30 second test is just long enough to demonstrate divergence beginning to occur in the SSF & UKF_NoSE estimates, while the UKF with sensor error estimation maintains an estimate within 3 standard deviation bounds.

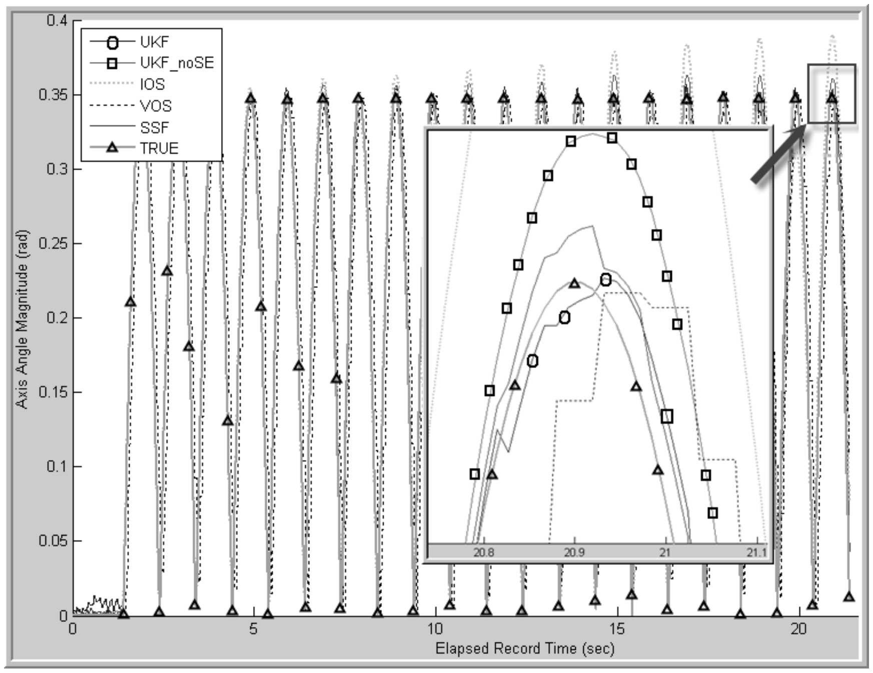


Figure 5.7: A comparison of all fusion filters to estimate the equivalent rotation vector magnitude, using angular rate measurements from the IMU, *Command 2*, for one 30 second test.

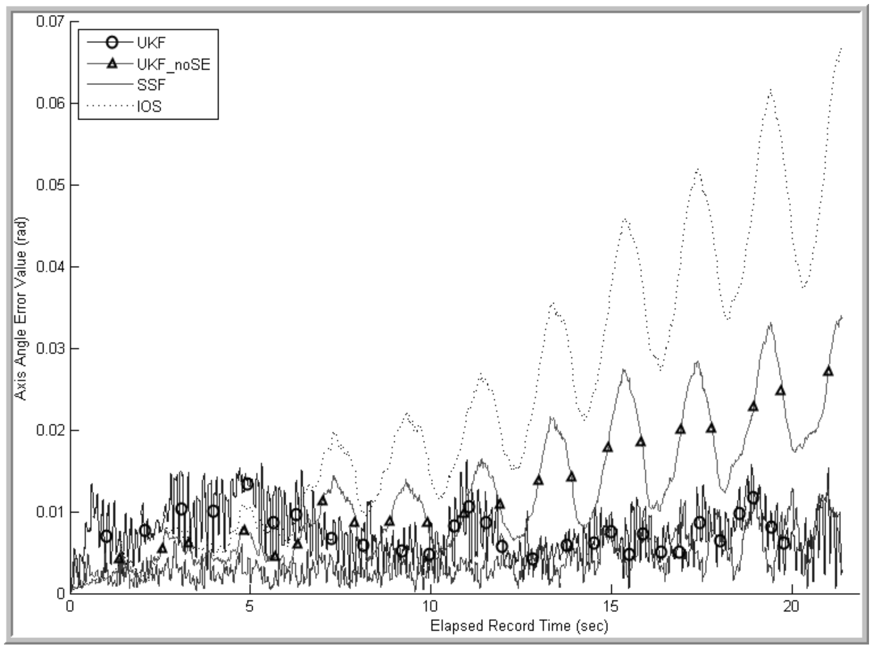


Figure 5.8: A comparison of all fusion filter's error in estimating the equivalent rotation vector magnitude estimate, for one 30 second test, with zero being the true value.

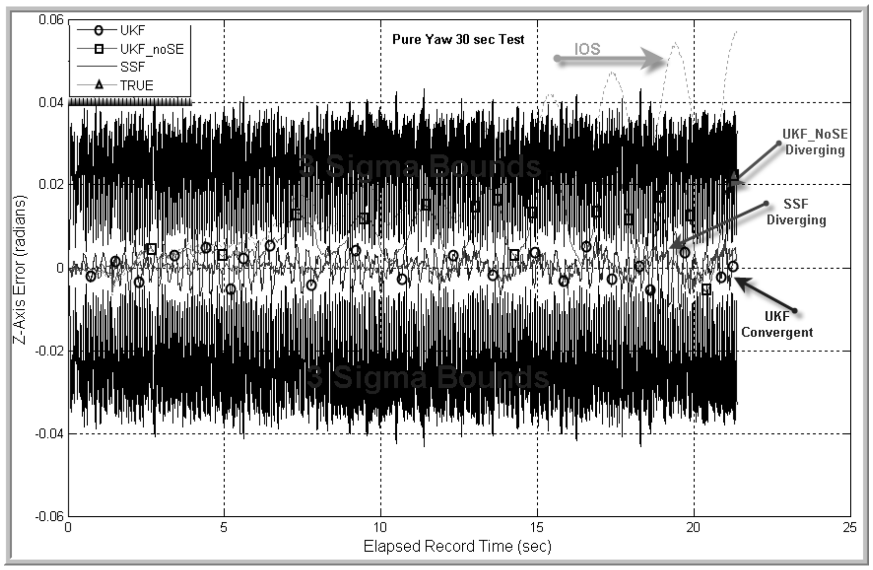
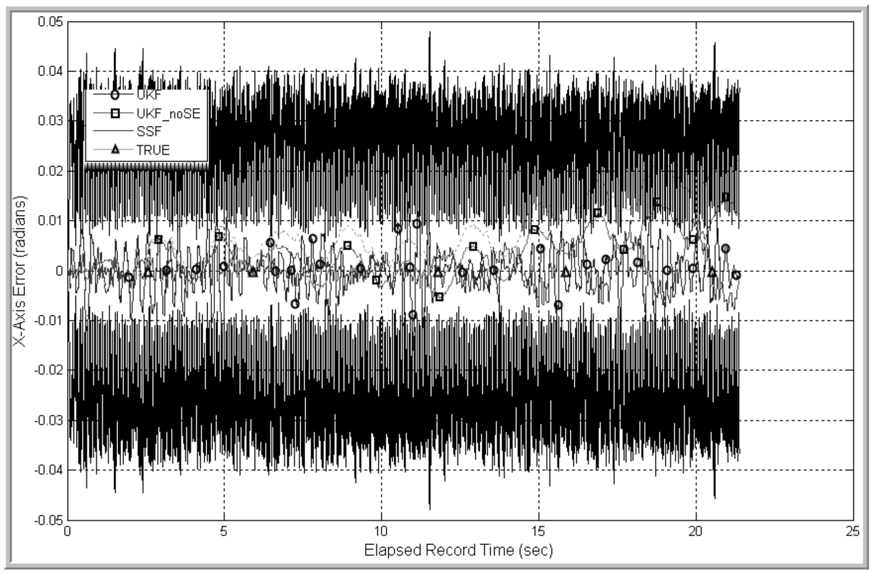
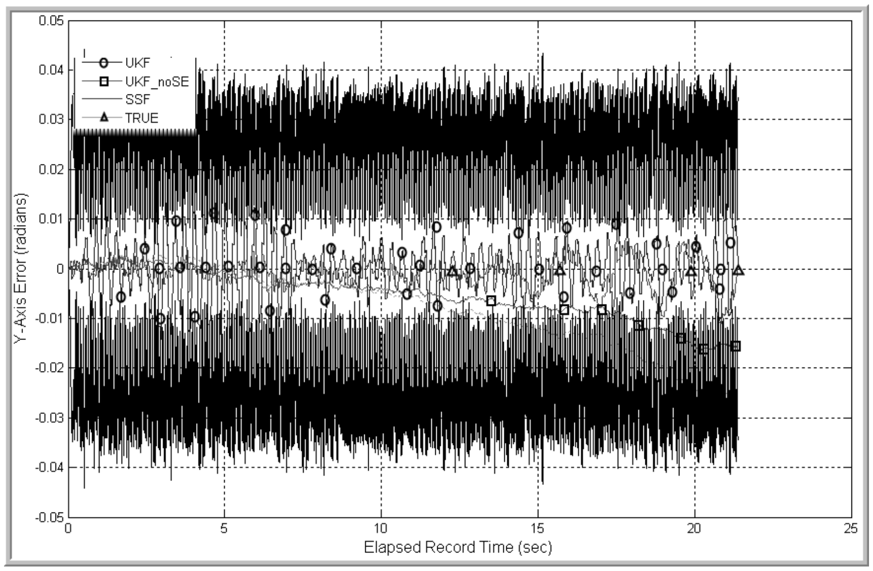


Figure 5.9: The individual IMU yaw (ψ) angle estimation error including 3σ bounds, for one 30 second test.



(a) Individual IMU roll (ϕ) angle estimation error.



(b) Individual IMU pitch (θ) angle estimation error.

Figure 5.10: The Atlas X and Y axis-angle estimation error for one 30 second test.

The results for the 30 second test are summarized in Table (5.4).

Filter Type	Axis-angle Magnitude RMS Error (degrees)	Maximum Error (degrees)
UKF with Sensor Error Estimation	0.5443	1.4267
UKF without Sensor Error Estimation	0.8251	1.9251
Simple Sensor Fusion	0.2750	0.8365
IMU	1.6272	3.8216

Table 5.4: The RMS error of axis-angle magnitude estimate **30 second test**

5.4.2 120 second test results

With a longer trial it is evident the estimated gyro sensor error terms play a role in ameliorating the individual Euler angle estimates. For this test, the IMU is initialized to rotate so that no angular movement about the local IMU x-axis occurs.

From the 120 second test, it becomes apparent that neither the SSF nor the *UKF without sensor error estimation* has the ability to account for IMU misalignments away from the assumed IMU axis of rotation $[0; 1; -1]$ for dynamic motion estimation. As shown in Figure 5.11 misalignment errors in the SSF overall axis-angle orientation estimate is hidden when only the magnitude is calculated. This is similar when the result of the scalar quaternion estimate is calculated and plotted, shown in Figure 5.12. The first scalar quaternion component, q_0, estimate, incorporates rotation (including misalignments) of all three vector quaternion components, and therefore this hides the misalignment errors. However, once the SSF estimated orientation quaternion is decomposed further into individual Euler angle rotations about the local IMU x, y, z axes ϕ, θ, ψ, the misalignment errors become immediately noticeable for each rotation angle away from the initial rest position. This is believed to be due to misalignment found in the vector portion of the quaternion estimates as seen in Figure 5.12 and Figure 5.13. As shown, once the quaternion is decomposed further into Euler angle rotations, the only filter that maintains a reliable estimate is the UKF which is shown following closest to zero. Estimation error is especially evident during Stage 4 of one 120 second test with non-constant angular velocity of the IMU. Figure 5.14 shows the estimated Atlas orientation angle ϕ about the IMU roll x-axis for the 120 second test. Again, this should remain zero since the rotation axis is perpendicular to the x-axis. It is only by using the UKF with sensor error correction that the remaining three vector quaternion portions become properly estimated.

A compelling result, shown in Figure 5.15, is obtained by finding the rotation

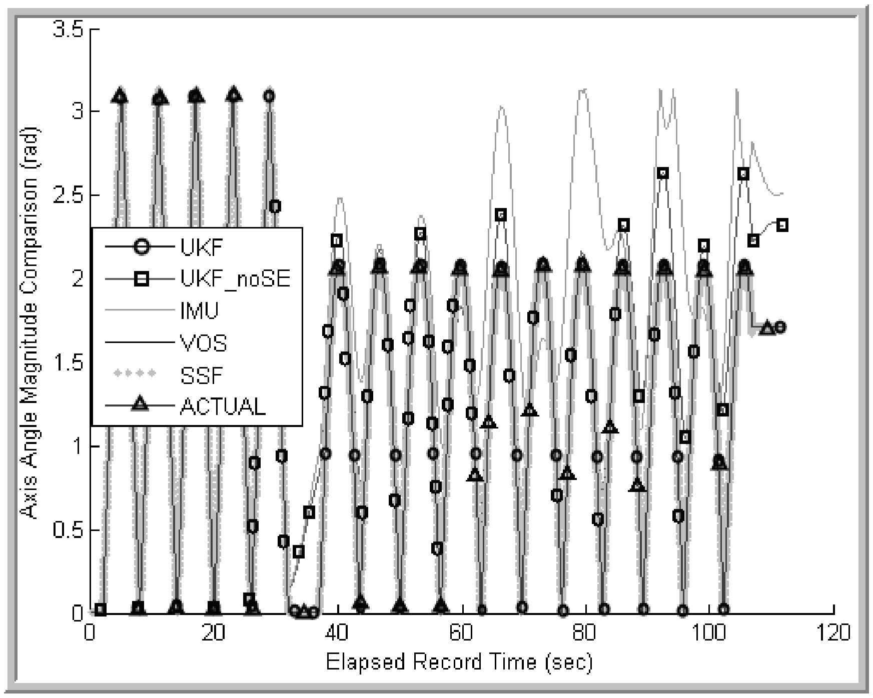
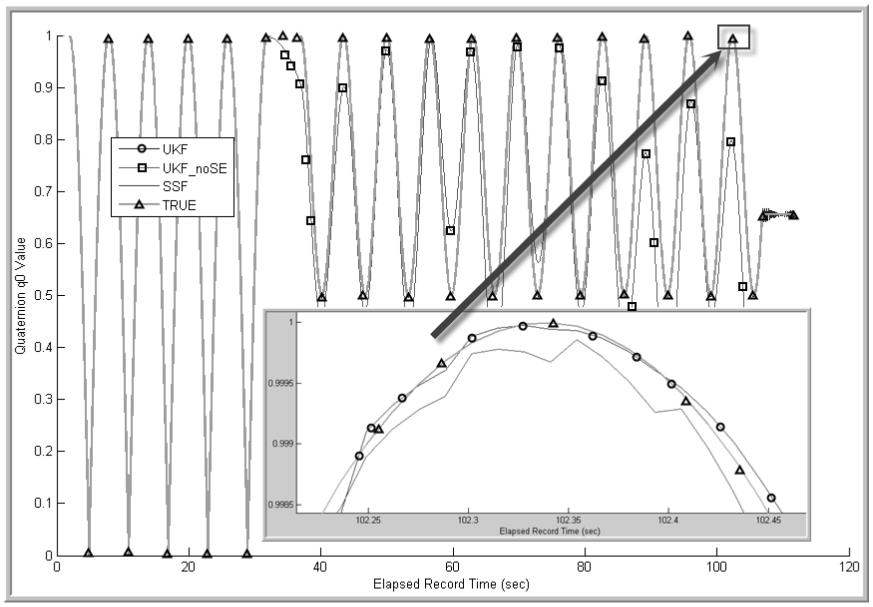


Figure 5.11: The equivalent rotation vector magnitude estimate for one 120 second test.

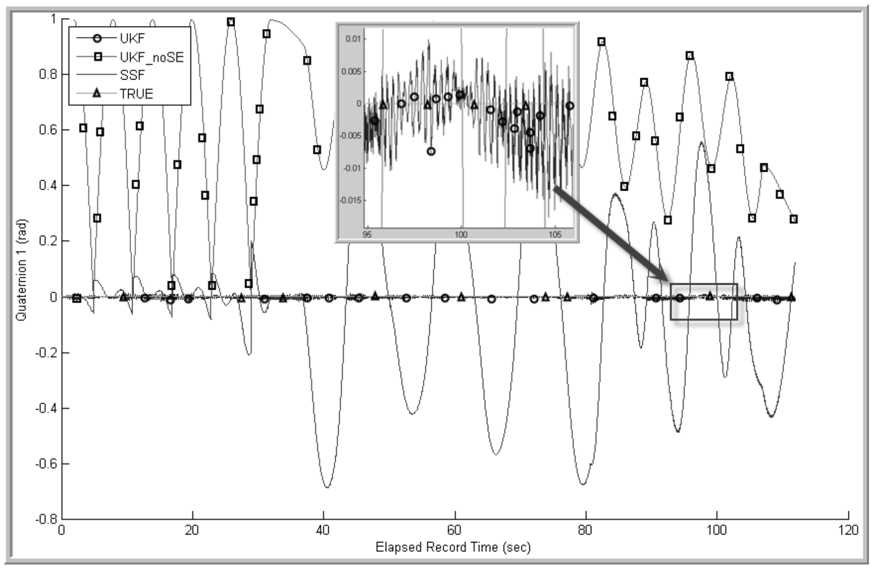
quaternion, $\mathbf{q}_{rot}$, between the estimated quaternion orientation and the true quaternion orientation, using Equation (2.10), followed by transforming into axis-angle error, using Equation 2.11, and then calculating the equivalent rotation vector error magnitude for each timestep. Plotting this, the overall effectiveness of the UKF with sensor error estimation outside of pure stationary motion is shown to have non-divergent behaviour.

The estimate can be shown to remain convergent over the entire tested sampling period. However, for clarity, only a small portion of the overall test is shown since the full 120 second figures appear very dense. Figure 5.16 demonstrates the Euler angle estimation error of the UKF with sensor error estimation algorithm for the local IMU yaw or z-axis along with the 3 standard deviation bounds.

Figure 5.17 shows cross axis error between IMU Y and X axis, further validating the UKF with sensor error estimation.

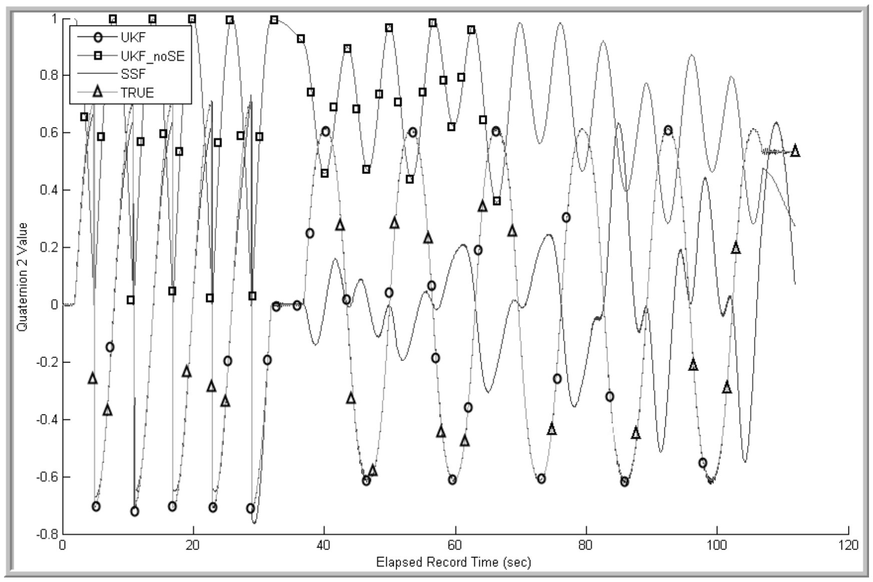


(a) q_0

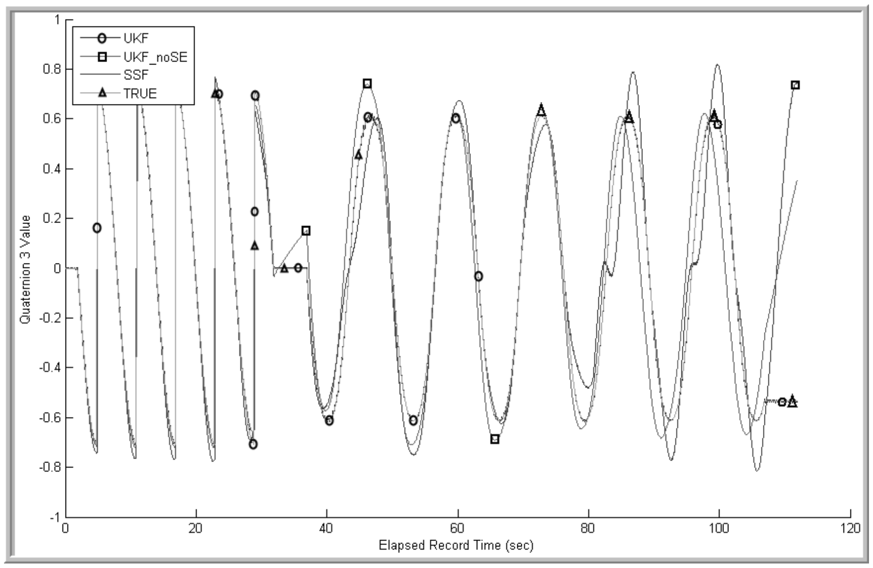


(b) q_1

Figure 5.12: The individual quaternion component estimates, q_0 and q_1, for one 120 second test.



(a) q_2



(b) q_3

Figure 5.13: The individual quaternion component estimates, q_2 and q_3, for one 120 second test.

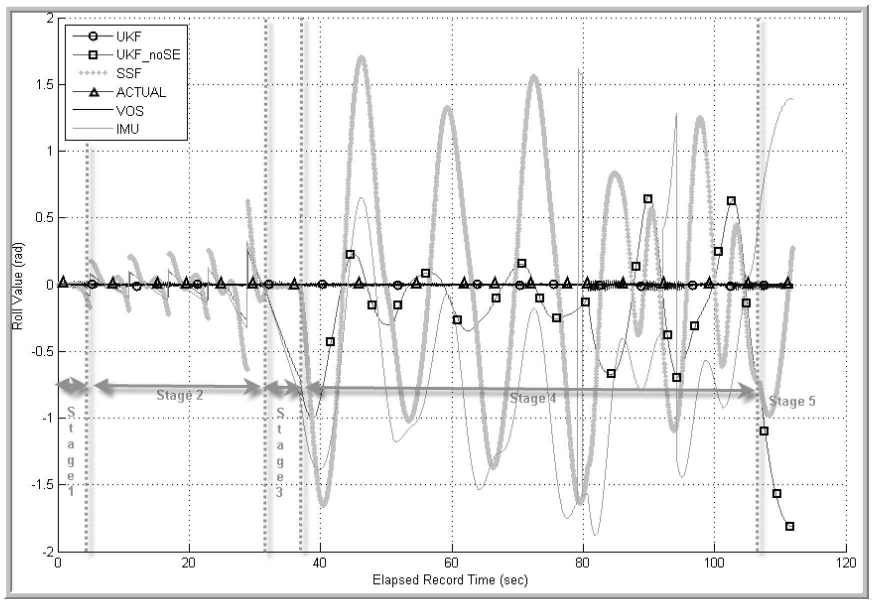


Figure 5.14: The x-axis Phi Angle, ϕ , estimation error for one 120 second test.

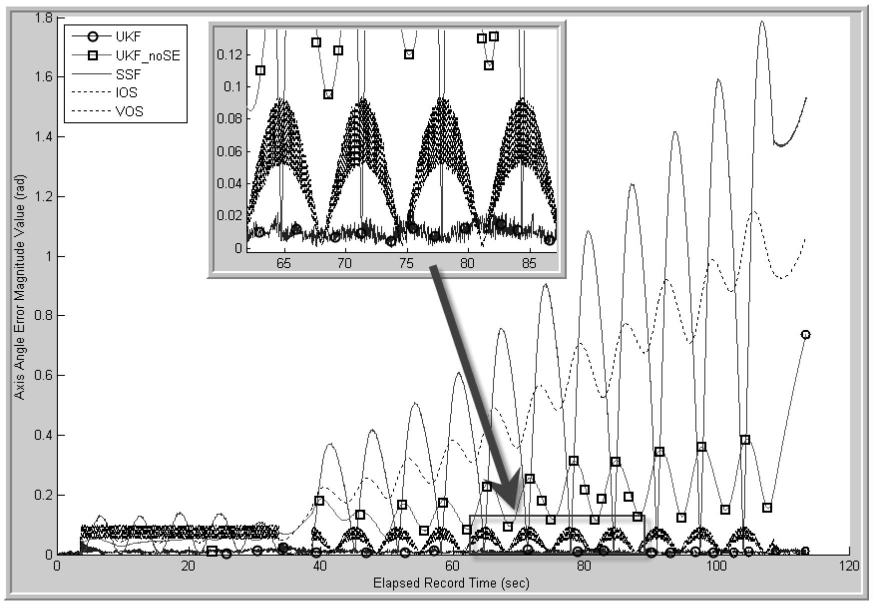


Figure 5.15: The axis-angle error magnitude using VOS measurements at 20 Hz for one 120 sec test.

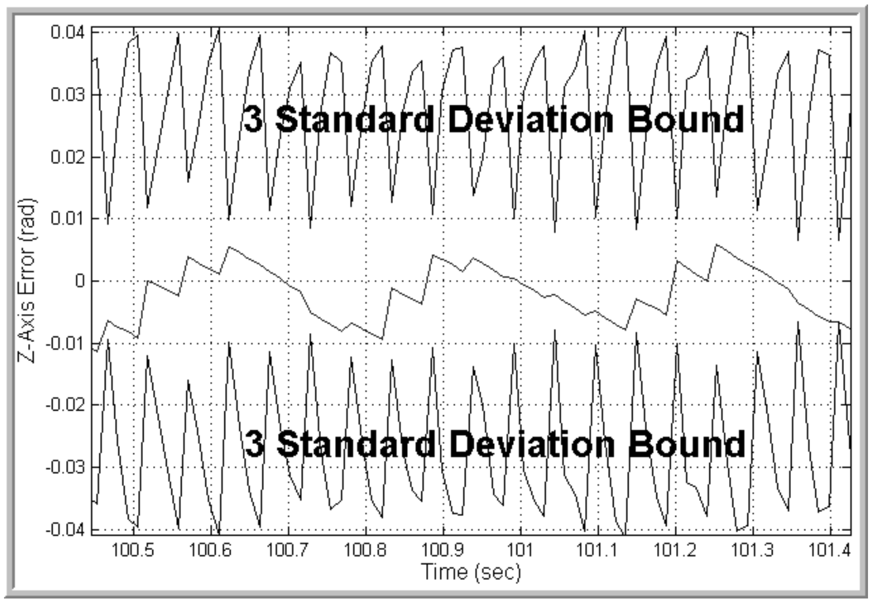


Figure 5.16: The Atlas (ψ) yaw Euler angle estimation error with 3σ bounds, for a portion of one 120 second test.

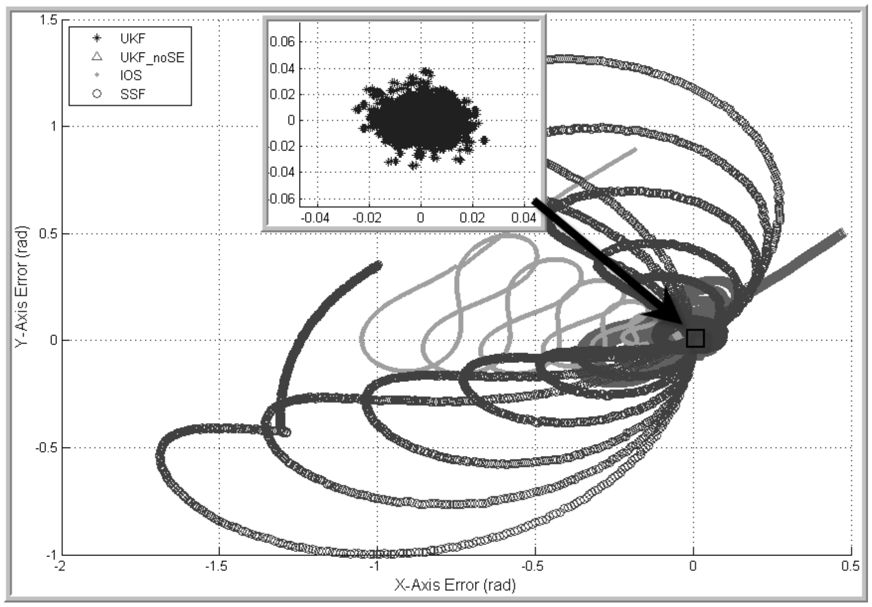


Figure 5.17: The cross Y vs. X axis-angle error for one 120 second test.

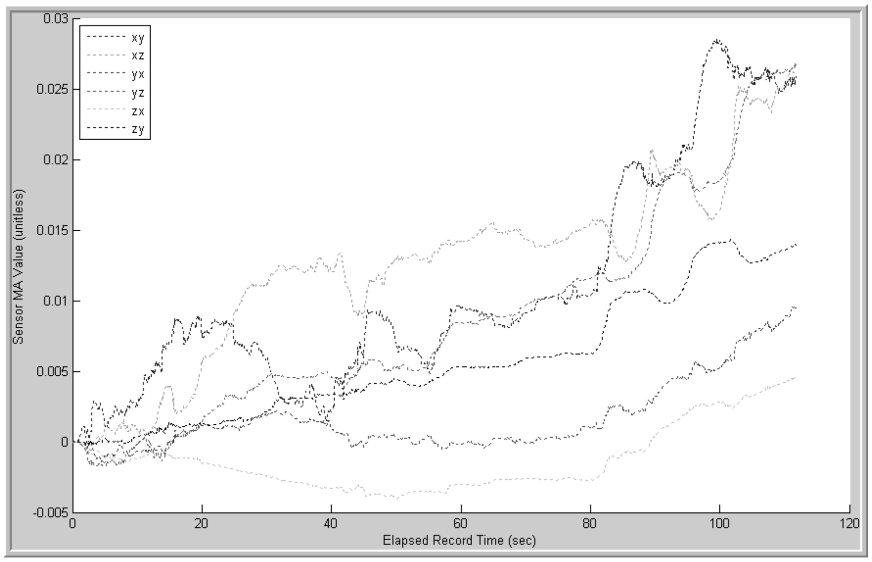


Figure 5.18: Misalignment Factors, $\mathbf{ma}_g$ sensor error estimates for one 120 second test determined using the UKF with sensor error estimation.

Gyro bias, scale factor, and misalignment factor sensor error estimates are shown in Figures 5.18, 5.19 and 5.20. Aside from the evidence of a stable estimation, it is difficult to know how well these error term values are estimated since there is no measured or otherwise actual known values. According to Shin, sensor error values will slowly change due to rate random walk but remain close to constant [14, 25]. This appears to be the case, however, without further testing it is not known whether further non-linear rotation would help the estimates to converge.

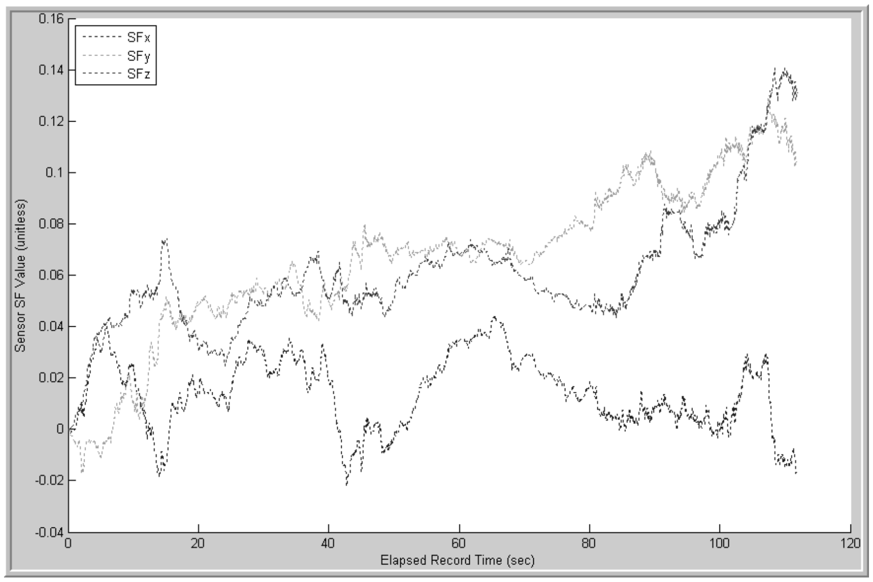


Figure 5.19: Scale Factors, $\mathbf{s}_g$, sensor error estimates for one 120 second test determined using the UKF with sensor error estimation.

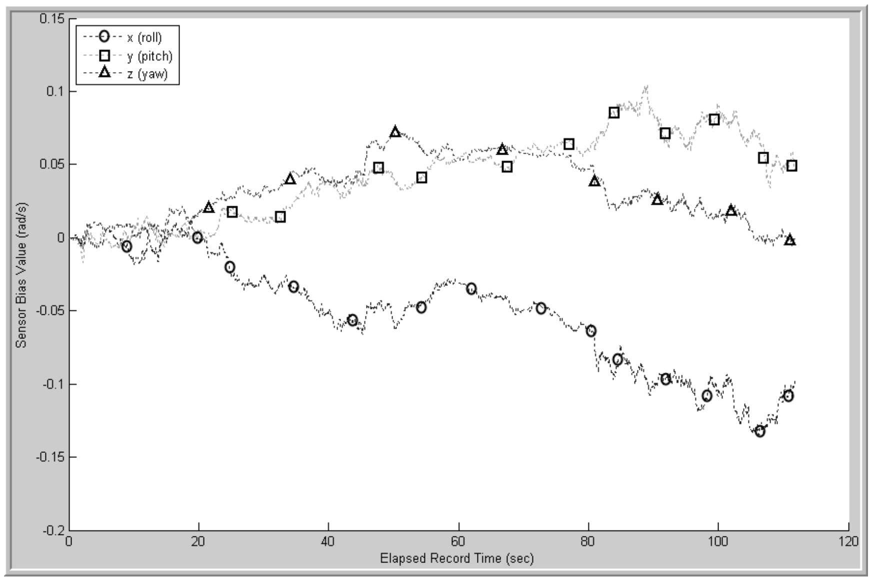


Figure 5.20: Gyro biases, $\mathbf{b}_g$, sensor error estimates for one 120 second test determined using the UKF with sensor error estimation.

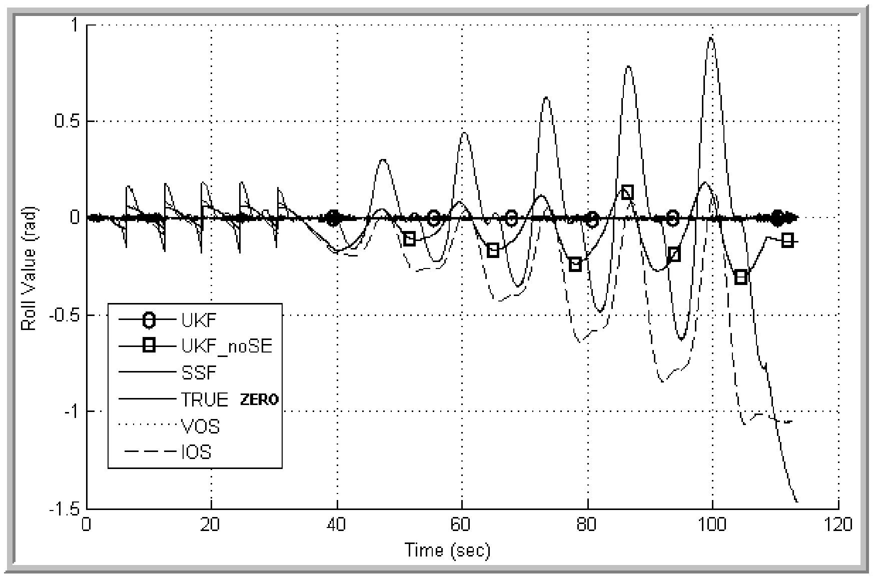


Figure 5.21: The estimated IMU roll angle for one 120 second test with the true value zero since the actual IMU roll axis is perpendicular to the axis of rotation.

Table 5.5 displays the root mean squared error from the equivalent rotation vector magnitude estimate calculated for one entire 120 second angular rate profile test, using each filter, corresponding to having latent simulated VOS measurements available at 20 Hz. The results clearly demonstrate benefit to using the UKF with sensor error estimation compared to the other filters and individual sensors by themselves. This result is even more clearly illustrated in Figure 5.21 which shows the estimated IOS roll angle displacement throughout one 120 second test, which in actuality, is zero since the IOS roll axis is perpendicular to the axis of rotation.

It should be noted that although the UKF with sensor error estimation performs accurately, the performance for use in a real-time application has not been investigated in any great detail. Currently, simulation performance using Matlab on a 1.8 GHz AMD Turion 64 ML-32 processor, 1GB DDR, notebook computer running Microsoft Windows XP is greatly in excess of real-time requirements with a measured second taking approx. 11.5 seconds to compensate off-line. The entire 120 second test was compensated in 1200 seconds which again is 11 times slower than needed. One proposal is to move the estimation algorithm into a more suitable real-time software, operated in a faster quad-core microprocessing computer. This will remain a task for fourth year CUSP students to investigate further although the assumption at this time is that faster runs are achievable with the code eventually compiled into an executable or run within Labview. Table 5.6,5.7 and 5.8 summarize RMS values of all the equivalent rotation vector magnitude estimate errors when compared to the true motor profile for each of three 120 second tests. The tables are separated for varying VOS frequencies of 17.5, 20 and 30 Hz, respectively. All of the tests conducted use CMD 2 to transmit IMU angular rates.

Filter Type	Rotation Vector Magnitude Average RMS Error (degrees)	Maximum RMS Error (degrees)
UKFwSE	0.87	3.55
UKF_NoSE	10.92	42.44
SSF	37.97	102.81
IMU	41.88	66.58
VOS	2.87	5.42

Table 5.5: The RMS values of the equivalent rotation vector magnitude estimate's error using VOS 20 Hz measurements for 120 second tests and averaged for one trial.

VOS **17.5** Hz	RMS of AA Mag Error (deg)			Maximum Error (deg)		
Trial	1	2	3	1	2	3
UKFwSE	4.4279	4.4428	5.1672	16.8059	15.0350	22.5024
VOS	4.1467	4.1713	4.1447	7.2653	7.3023	7.4093
IOS	30.0518	74.0519	89.9960	66.0861	178.7743	179.9356

Table 5.6: Three trials showing the RMS values for the axis-angle magnitude estimate's error using VOS 17.5 Hz measurements for a 120 second test.

VOS **20** Hz	RMS of AA Mag Error (deg)			Maximum Error (deg)		
Trial	1	2	3	1	2	3
UKFwSE	0.6653	0.9784	1.0155	2.8693	3.1980	2.9110
VOS	3.2596	3.2824	3.2604	5.7088	5.8511	5.7845
IOS	30.0518	74.0519	89.9960	66.0861	178.7743	179.9356

Table 5.7: Three trials showing the RMS error of the axis-angle magnitude estimate's error using VOS 20 Hz measurements for a 120 second test.

Table 5.9 lists the RMS value for 17.5 Hz data, comparing the second half (60 sec) to the entire 120 second estimated states. The results indicate how, after some length of time used for initial sensor error terms to become estimated, the overall attitude estimate is improved substantially beyond the first dynamic sequence. This result adds emphasis to the possibility of only using the UKF with sensor error estimation for initial start-up, in order to calibrate sensor error terms (which are relatively constant), if future real-time UKF computation time is too cumbersome. It is expected once the VOS becomes operational, sensor error estimation fusion may only need to be performed for a set pattern rate profile to determine initial sensor error values. Once performed, a modified filter could treat the sensor error values as constant (without need for continual estimation). This will reduce the number of states in the filter, thereby, increasing computational speed. Suggestions could be to eliminate state estimation for the gyro sensor error terms after a specified period of time, or include

VOS **30** Hz	RMS of AA Mag Error (deg)			Maximum Error (deg)		
Trial	1	2	3	1	2	3
UKFwSE	0.4498	0.5259	0.6187	3.2924	2.3179	2.4515
VOS	2.3734	2.3890	2.3715	4.1275	4.1848	4.2670
IOS	30.0518	74.0519	89.9960	66.0861	178.7743	179.9356

Table 5.8: Three trials showing the RMS error of the axis-angle magnitude estimate using VOS 30 Hz measurements for a 120 second test.

only gyro drift error estimation, since gyro drift tends to be one of the more prominent sources of error.

VOS **17.5** Hz	RMS Full (deg)	RMS 2^{nd} half (deg)
UKFwSE	4.4279	1.1477
VOS	4.1467	3.6255
IOS	30.0518	41.6969

Table 5.9: A comparison of the RMS errors, using VOS 17.5 Hz measurements, for the 2^{nd} half of trial 1 of the 120 second test vs. the complete 120 second test.

Chapter 6

Conclusions and Recommendations For Future Work

6.1 Conclusions

6.1.1 Oriention Estimator

In this thesis a quaternion-based UKF with sensor error estimation was developed for the fusion of Atlas IMU angular rates and VOS absolute orientation quaternion measurements. As such, a unique estimation algorithm was developed which can be implemented into the overall Atlas sphere to improve the orientation estimate beyond the current capabilities one single sensor. Making use of the unique characteristic that quaternions are free from representational singularities, the Atlas UKF allows for estimation of unconstrained 3D attitudes through any set of rotation rate profiles. The Atlas UKF formulation also takes into consideration known latency issues due to out-of-sequence-measurements from the VOS. Through use of the UKF, sensor fusion of the VOS and IOS measurements is accomplished to improve upon either sensor measurement alone.

2. A Simulation program is developed in Matlab to verify the adapted unscented

Kalman

lter for use within the Atlas platform showing that there is no divergence within the operational range.

6.1.2 Simulation

A simulation was developed to test any non-linear rate profile. Noise was simulated for each of the estimated states and OOSM from the VOS were also simulated. The simulation can test any suspected IOS or VOS frequency and results from simulation reveal good feasibility of the UKF for attitude estimation through a non-linear trajectory which led to further testing using the actual measured data from the IOS. The simulation verified the adapted unscented Kalman filter was non-divergent within the Atlas operational range.

6.1.3 Experimentation

Using a Microstrain 3DM-GX1 IMU and simulating a 'slower', 'lagged' vision attitude arriving at some later known time-step, an experimental procedure and test setup was developed for recording a known dynamic rotational trajectory followed by the IOS. As well, using a high precision motor with encoder, a known nonlinear angular rate profile can be followed while simultaneously acquiring IOS angular rate measurements. Moreover, the settings to use when operating the IOS (3DM-GX1) were tested and the Matlab code that can be used to communicate with the IOS inside Matlab was created.

It was demonstrated, using real IOS data, that accounting for IOS sensor errors is crucial to allow for online calibration and accurate estimation of changing orientations which surpass the small angle assumption in typical attitude estimation filters. Although SSF and UKF, *without sensor error estimation*, work reasonably well for stationary and low frequency motion, it was found that only the UKF with sensor error estimation maintains a reasonable estimate under higher frequency rotation.

Two different IOS test scenarios were conducted to test various axes of rotation. The first test was a 30-second test used to acquire angular rate measurements by rotating solely about the Atlas $[0, 0, -1]$, z or 'Yaw' axis. This test revealed little to no effect from the misalignments. This is assumed to be due to the short duration of testing and a lack of motion to excite/reveal the off axis sensor errors. The second test setup involved rotating about a tilted $45°$ axis of rotation, $[0, 1, -1]^T$. This process further excited the IOS sensor errors and caused noticeable misalignment issues and drift, further validating that sensor errors should be represented in the process model.

Simulated VOS measurements were created at three frequencies; 17.5, 20, and 30 Hz. Results from these tests indicate that with relatively inaccurate vision measurements, receiving absolute VOS measurements more frequently, will yield improved fused estimates.

Results using true dynamic data have shown that after an initial period to obtain sensor error estimates, which must include an adequate dynamic rate profile, sensor error terms can be estimated and help to improve the overall fused IOS/VOS estimate beyond either sensor's original estimate. It is unknown, at this time, whether longer tests would further improve the overall estimation, although based on literature [9], it is presumed the filter estimate would become better over time since slow varying sensor errors become excited and increasingly observable with nonlinear motion.

Eventually, a full real-time version of the UKF with sensor error attitude estimation could be implemented on-board the Atlas motion platform. This version will need to have the ability to handle real-time computational loads, which is currently not the case. Arguments can also be made for the possibility of using only the UKF with sensor error estimation algorithm for an initial start-up process in order to calibrate for the misalignment and scale factor sensor error terms which, presumably, remain relatively constant over a short period of time after an initial warm up period. These terms could be held constant, eliminating need for use of a complex UKF algorithm, if computationally cumbersome.

Through development of a quaternion-based indirect UKF filter with sensor error estimation, this thesis successfully demonstrated how an improved Atlas attitude can be estimated when compared to either individual sensor by itself. The results may not be optimal, and no attempt was made to prove optimality. Several angular rate profiles have been tested using real IMU measurements in an attempt to excite off axis gyroscopic errors. Results reveal suitability for use of a sensor fusion algorithm, however, further tests could help to properly tune the process noise covariance matrices, $\mathbf{Q}_{IOS}$ and $\mathbf{R}_{VOS}$, to bring the fused orientation estimate further into agreement with the true trajectory. These can be tuned once the VOS is operational and bench-testing has been completed.

This research serves as ground work for an eventual real-time sensor fusion algorithm to be implemented in the Atlas motion platform. All pertinent Matlab m-code is included in Appendices and further supplied online through the CUSP SVN server [19]. The code can be reused in the future real-time control of the Atlas platform.

6.2 Future Recommendations

The following are recommendations as a result of this research:

1. Once the VOS is fully operational, the VOS process should be updated to incorporate any changes to the VOS process model and VOS parameters. Further offline tests involving actual measurements from the VOS can then be conducted with the current UKF.

2. Using the algorithm outlined in this thesis, a real-time version of the fusion filter should be implemented to receive both VOS & IOS measurements within the control loop.

3. The current algorithm needs to be optimized for speed since it currently runs 11 times slower than required. Suggestions for future modifications are to properly tune the process and measurement covariance matrices once the VOS is operational. Compile the code into a program other than Matlab for real-time use to improve performance and speed.

List of References

[1] CUSP, "Cusp website." Department of Mechanical & Aerospace Engineering, Carleton University, http://cusp.mae.carleton.ca, viewed November 2008.

[2] A. Weiss, R. Langlois, and M. Hayes, "The effects of dual row omnidirectional wheels on the kinematics of the atlas spherical motion platform," *Mechanism and Machine Theory*, 2008.

[3] A. Paterson, "Inertial orientation sensor, dr-sys-ap.08.ios.01," design report, Department of Mechanical & Aerospace Engineering, Carleton University, July 2007.

[4] E. Kraft, "A quaternion-based unscented kalman filter for orientation tracking.." Cairns, Australia: Proceedings of the 6th International Conference on Information Fusion (ISIF), July 2003.

[5] P. Maybeck, *Stochastic models, estimation, and control.*, vol. 1. New York, New York: Academic Press Inc., 1979.

[6] M. Ahmadi, A. Khayatian, and P. Karimaghaee, "Orientation estimation by error-state extended kalman filter in quaternion vector space," in *SICE Annual Conference*, pp. 60–67, Sept. 2007.

[7] S. Julier, R. Merwe, and E. A. Wan, "Sima-point kalman filters for nonlinear estimation and sensor-fusion - applications to integrated navigation," *American Institute of Aeronautics and Astronautics*, pp. 1–30, 2004.

[8] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," *Aerosense: 11th Int. Symp. Aerospace/Defense Sensing, Simulation and Controls*, vol. 3, pp. 182–193, 1997.

[9] J. J. LaViola, "A comparison of unscented and extended kalman filtering for estimating quaternion motion," *In Proc. of the American Control Conference*, vol. 3, pp. 2435–2440, 2003.

[10] Y. Wu and D. Hu, “Unscented kalman filtering for additive noise case: Augmented vs. non-augmented,” *IEEE, Proceedings of the American Control Conference*, vol. 6, pp. 4051–4055, June 2005.

[11] Q. M. Lam, T. Hunt, P. Sanneman, and S. Underwood, “Analysis and design of a fifteen state stellar inertial attitude dtermination system,” *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 11–14, Aug. 2003.

[12] E.-H. Shin, “A quaternion-based unscented kalman filter for the integration of gps and mems ins,” *Proceedings of the 17th International Technical Meeting of the Satellite Division of the Institute of Navigation ION GNSS*, pp. 1060–1068, 2004.

[13] E.-H. Shin and N. El-Sheimy, “An unscented kalman filter for in-motion alignment of low-cost imus,” *in Proceedings of the IEEE Frames Conference*, pp. 273–279, 2004.

[14] E.-H. Shin, *Estimation Techniques for Low-Cost Inertial Navigation.* Ph.d. dissertation, The University of Calgary, Department of Geomatics Engineering, May 2005.

[15] Y.-J. Cheon and J.-H. Kim, “Unscented filtering in a unit quaternion space for spacecraft attitude estimation,” *IEEE International Symposium on Industrial Electronics*, pp. 66–71, 2007.

[16] S. Roumeliotis, G. Sukhatme, and G. Bekey, “Circumventing dynamic modelling: Evaluation of the error-state kalman filter applied to robot localization,” *in Proceedings of IEEE International Conference on Robotics and Automation, Detroit, MI*, vol. 2, pp. 1656–1663, May 1999.

[17] K. Park, *GPS Receiver Self Survey and Attitude Determination Using Pseudolite Signals.* Ph.d. dissertation, Texas A&M University, Aug 2004.

[18] D. H. Titterton and J. L. Weston, *Strapdown inertial navigation technology.* American Institute of Aeronautics and Astronautics, 2 ed., 2004. Hardback Book.

[19] CUSP, “2008-2009 final report.” Carleton University, Department of Mechanical & Aerospace Engineering, 2008-2009.

[20] S. W. R. Hamilton, “Lectures on quaternions,” *Proceedings of the Royal Irish Academy*, pp. 1–16, 1847.

[21] J. C. Chou, "Quaternion kinematic and dynamic differential equations," *IEEE Trans Robotics and Automation 8*, pp. 53–64, Feb. 1992.

[22] EuclideanSpace, "Matrix to quaternion." http://www.euclideanspace.com, viewed August 2009.

[23] Wikipedia, "Axis-angle." website, http://en.wikipedia.org/wiki/State_space_(controls), viewed November 2008.

[24] F. Orderud, "Comparison of kalman filter estimation approaches for state space models with nonlinear measurements," *in Proceedings of Scandinavian Conference on Simulation and Modeling*, 2005.

[25] Q. M. Lam, N. Stamatakos, C. Woodruff, and S. Ashton, "Gyro modeling and estimation of its random noise sources," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pp. 1–11, Aug. 2003.

[26] Microstrian Inc., *Detailed Specifications For 3DM-GX1.* Technical Details, http://www.microstrain.com/3dm-gx1.aspx, viewed Nov. 2008.

[27] W. Stockwell, "Angle random walk." Crossbow Technology Inc., http://www.xbow.com, viewed April 2009.

[28] K. Chisholm, "Inertial measurement unit and vision fusion for a novel simulator motion platform," private communication, Department of Mechanical & Aerospace Engineering, Carleton University, February 2009.

[29] Microstrain Inc., *3DM-GX1 Comarison With and Without Temperature Compensation.* Technical Report, http://www.microstrain.com/3dm-gx1.aspx, viewed Nov. 2008.

[30] Microstrain Inc., *3DM-GX1 Data Communication Protocol 3.1.01.*, February 2005. Firmware version 2.1.00, http://www.microstrain.com/3dm-gx1.aspx, viewed Nov. 2008.

[31] Z. Prime, *Using the Microstrain 3DM-G(X1) IMUs.* The School of Mechanical Engineering, viewed Oct. 2007. http://www.mecheng.adelaide.edu.au/∼zprime/documents/IMUTechNote.pdf.

[32] B. Rasquinha, "Simple sensor fusion algorithm development for atlaslite," tech. rep., Department of Mechanical & Aerospace Engineering, Carleton University, CUSP Design Report, DR-CNT-br.09.SimpleSensorFusion.02.pdf, 2009.

[33] K. Chisholm, "Camera calibration for the vision orientation sensor," private communication, Department of Mechanical & Aerospace Engineering, Carleton University, July 2009.

[34] X. Pennec, "Computing the mean of geometric features - application to the mean rotation," *Institute National de Recherche en Informatique et en Automatique (IN-RIA)*, March 1998.

[35] Apex Dynamics Inc., *Apex Dynamics AD047-1 Specifications.* http://www.apexdyna.com/, viewed Aug. 2009.

[36] E. Trucco and A. Verri, *Introductory techniques for 3-D computer vision.* Prentice Hall, 1998.

[37] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," *In Proc. of the Seventh IEEE International Conference on Computer Vision*, vol. 1, pp. 666–673, 1999.

[38] G. Bebis, "The geometry of perspective projection." University of Nevada Computer Science Department, http://www.cse.unr.edu/ bebis/CS791E/Notes/PerspectiveProjection.pdf,viewed February 2009.

Appendix A

Finding the Mean Quaternion

Unlike vector quantities, which can use simple barycentric averaging to calculate the mean, quaternions will not yield correct results for the computation of the mean quaternion in this fashion. A series of orientation quaternions is averaged keeping in mind that quaternions are members of a homogeneous Riemannian manifold (the four dimensional unit sphere) and not of a vector space (see Kraft [4]). The approach uses the intrinsic gradient descent algorithm outlined briefly here [34], [13], [4], [15].

Note: The Matlab m file for this routine is provided in Appendix D.1.

A.1 The Intrinsic Gradient Decent Algorithm

Given multiple quaternion points; $\mathbf{q}_i, i = 1, ..., 2n$, the weighted mean quaternion, $\bar{\mathbf{q}}$, can be computed as follows:

1. Choose any of the $\mathbf{q}_i$'s as the initial mean quaternion, $\bar{\mathbf{q}}$.
2. Calculate the attitude difference (*quaternion error*), as outlined in Equation (2.10) in Section (2.3). $\mathbf{q}_{err} = \mathbf{q}_i \otimes (\bar{\mathbf{q}})^{-1}$.
3. Convert each $\mathbf{q}_{err}$ into their corresponding axis angle rotation vector errors, $\mathbf{\Phi}_i, i = 1, ..., 2n$.

4. Calculate the weighted barycentric mean of the rotation vectors, $\bar{\boldsymbol{\Phi}} = \sum_{i=1}^{2n} \mathcal{W}_i \bar{\Phi}_i$.

5. Convert $\bar{\boldsymbol{\Phi}}$ into the corresponding correction quaternion, $\mathbf{q}_{corr}$.

6. Update the mean quaternion, $\tilde{\mathbf{q}} = \mathbf{q}_{corr} \otimes \tilde{\mathbf{q}}$.

7. Repeat steps (2) to (6) until $\left\|\bar{\boldsymbol{\Phi}}\right\|$ falls below a specified threshold; 0.000001 radians in this case.

Typically, the number of iterations for convergence is very small (in most cases, one).

Appendix B

VOS & IOS Specifications

B.1 VOS Intrinsic Parameters

Parameter	Description	Value	Units
$\mathbf{r}_{sphere}$	Atlas Sphere Radius	1.4778×10^3	mm
$(x, y)_{res}$	Camera Resolution	640×480	$pixels$
ccd_{size}	Camera Sensor CCD	12.700	mm
f	Camera Focal Length	8	mm
(o_x, o_y)	Principal Point	$(320, 240)$	$pixel$
$(s_x, s_y$	Scaling Factors	$(1.59, 1.59) \times 10^{-2}$	$mm/pixel$
(f_x, f_y)	Combined Scaling and Focal Length	$(503.9370, 503.9370)$	$pixel$

Table B.1: Intrinsic Camera Parameters

B.2 VOS Extrinsic Parameters

Rotation matrix from world to camera frame

- rotate 90 degrees about World x axis
- then rotate 60 degrees about World y axis

$$^{c}\mathbf{R}_w = \begin{bmatrix} 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & -1 \\ -\sqrt{3}/2 & 1/2 & 0 \end{bmatrix} (mm)$$

Calibrated, including radial and tangential distortions (as per [33]) but **currently not implemented**.

$$^{c}\mathbf{R}_w = \begin{bmatrix} 0.853634809828412 & 0.520398049122761 & 0.0222144529177602 \\ 0.0191206445695910 & 0.0113123450142586 & -0.999753185442048 \\ -0.520520904864016 & 0.853848874988720 & -0.000293734647024813 \end{bmatrix} (mm)$$

Translation from world frame origin to camera frame origin, calibrated to include radial and tangential distortions (as per [33]).

$$^{c}\mathbf{T}_w = \begin{bmatrix} -9.32862410622034 \\ -1.05455265129037 \\ -532.512422264170 \end{bmatrix} (mm)$$

Parameter	Description	Value	Units
$\sigma_{u,v}$	Pixel Frame StDev	(0.30,0.54)	*pixel*
$\sigma_{CMM_{x,y,z}}$	Marker Position CMM Measurement StDev	0.0254	*mm*

Table B.2: Extrinsic camera parameters.

B.3 VOSParameters.m

This m file lists the VOS specifications for use in the UKF

```
% Marker locations on the sphere wrt the sphere frame and the sphere radius
% load .mat file created by SphereMarkerPositions.m script
    Spts = open('SpherePts_r4.75.mat');
    PsAll = Spts.Sphere_points; %mm


% visualize local sphere points
%     hold on;
%     grid on;
%     plot3(Ps(1,:),Ps(2,:),Ps(3,:),'.','MarkerSize',15) %nonuniform
%     hold off;
%number of points on sphere
    n = size(PsAll, 2); %usually 32
%number of points in frame to be analyzed
    N = 3;
%sphere radius
    r = Spts.r_s; %mm
```

Intrinsic Camera Parameters

```
%Resolution
    x_res = 640; %pixels
    y_res = 480; %pixels
%ccd size (diagonal)
    ccd_size = 0.5*25.4; %mm
%focal length
```

```
    f = 8; %mm
%principle axis location
    ox = x_res/2;%pix
    oy = y_res/2;%pix
%scaling factors
    sx = ccd_size/sqrt(x_res^2+y_res^2);%mm/pix
    sy = ccd_size/sqrt(x_res^2+y_res^2);%mm/pix
%combined scaling and focal length
    fx = f/sx;%pix
    fy = f/sy;%pix
```

Extrinsic Camera Parameters

Rotation matrix from world to camera frame rotate 90 degrees about x axis then rotate 60 degrees about y axis

```
    Rcam = [1/2        sqrt(3)/2  0;
         0          0          -1;
        -sqrt(3)/2 1/2        0];

%find optimal distance from world frame origin to camera origin
    alpha = atan2(sy*y_res/2, f);
    Z = r/sin(alpha); %mm

%location of world frame origin wrt camera frame origin
%(x and y minor ~3cm offset)
    Po = [30*randn; 30*randn; Z]; %mm
```

```
%location of camera frame origin wrt world frame origin
    T = -Rcam'*Po;%mm


%create perspective projection matrix
M = [-fx*Rcam(1, 1)+ox*Rcam(3, 1)  -fx*Rcam(1, 2)+ox*Rcam(3, 2)...
  -fx*Rcam(1, 3)+ox*Rcam(3, 3)  -fx*Po(1)+ox*Po(3);
-fy*Rcam(2, 1)+oy*Rcam(3, 1)  -fy*Rcam(2, 2)+oy*Rcam(3, 2)...
  -fy*Rcam(2, 3)+oy*Rcam(3, 3)  -fy*Po(2)+oy*Po(3);
Rcam(3, 1) Rcam(3, 2) Rcam(3, 3) Po(3)];
```

VOS Measurement Noises

```
%marker position CMM measurement Std. Dev. in x, y and z components
    Sigma_CMM = 0.001*25.4; %mm


%Measured point locations on sphere (from CMM - stays constant)
    PsAll_m = PsAll + Sigma_CMM*randn(3, n);
    save('SpherePts_r4.75_m.mat', 'PsAll_m');


%Pixel location Std. Dev. in xim, yim components
    Sigma_imu = 0.30; %pixels
    Sigma_imv = 0.54; %pixels
    Sigma_im = blkdiag(Sigma_imu,Sigma_imv);
```

VOS Measurement Covariance Matrix 'Rp' - For pixel image

CMM sphere points Covariance matrix

```
    Rs = Sigma_CMM^2*eye(3*N, 3*N);
```

```
% Pixel location points covariance matrix

    Rim_pt = Sigma_im^2;

    Rim = Rim_pt;

for k = 2:N

    Rim = blkdiag(Rim,Rim_pt);

end
```

create structure of camera parameters

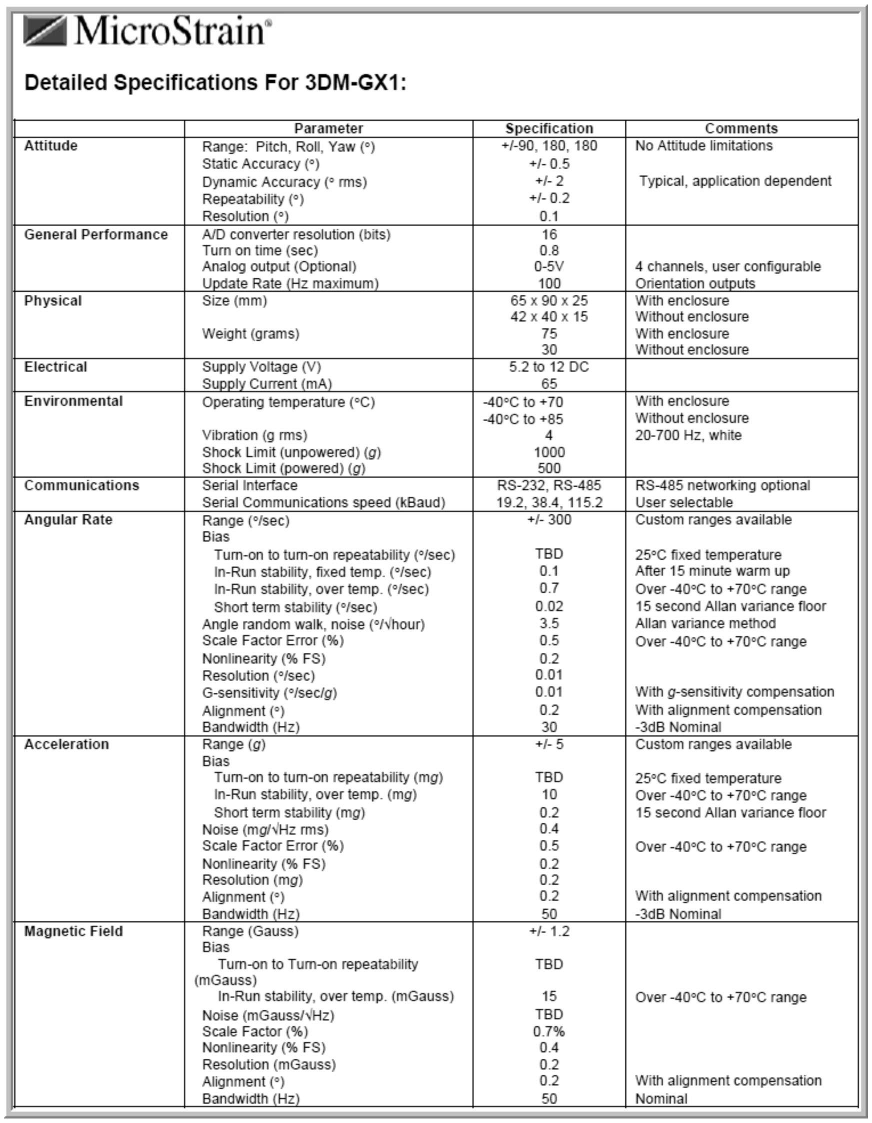
```
    CamCalInfo = struct('T', T,...
                        'O', [ox; oy], ...
                        'F', [fx; fy], ...
                        'Rcam', Rcam, ...
                        'Po', Po, ...
                        'M', M, ...
                        'Sigma_CMM',Sigma_CMM, ...
                        'Sigma_im',Sigma_im, ...
                        'Rs',Rs, ...
                        'Rim',Rim);
```

B.4 IOS Detailed Specifications

B.5 IMUspecs.m

This m file lists the IOS specifications for use in the UKF

```
% Created June 29, 2009

% Jesse Linseman

% Carleton University
```



MicroStrain®

Detailed Specifications For 3DM-GX1:

	Parameter	Specification	Comments
Attitude	Range: Pitch, Roll, Yaw (°)	+/-90, 180, 180	No Attitude limitations
	Static Accuracy (°)	+/- 0.5	
	Dynamic Accuracy (° rms)	+/- 2	Typical, application dependent
	Repeatability (°)	+/- 0.2	
	Resolution (°)	0.1	
General Performance	A/D converter resolution (bits)	16	
	Turn on time (sec)	0.8	
	Analog output (Optional)	0-5V	4 channels, user configurable
	Update Rate (Hz maximum)	100	Orientation outputs
Physical	Size (mm)	65 x 90 x 25	With enclosure
		42 x 40 x 15	Without enclosure
	Weight (grams)	75	With enclosure
		30	Without enclosure
Electrical	Supply Voltage (V)	5.2 to 12 DC	
	Supply Current (mA)	65	
Environmental	Operating temperature (°C)	-40°C to +70	With enclosure
		-40°C to +85	Without enclosure
	Vibration (g rms)	4	20-700 Hz, white
	Shock Limit (unpowered) (*g*)	1000	
	Shock Limit (powered) (*g*)	500	
Communications	Serial Interface	RS-232, RS-485	RS-485 networking optional
	Serial Communications speed (kBaud)	19.2, 38.4, 115.2	User selectable
Angular Rate	Range (°/sec)	+/- 300	Custom ranges available
	Bias		
	Turn-on to turn-on repeatability (°/sec)	TBD	25°C fixed temperature
	In-Run stability, fixed temp. (°/sec)	0.1	After 15 minute warm up
	In-Run stability, over temp. (°/sec)	0.7	Over -40°C to +70°C range
	Short term stability (°/sec)	0.02	15 second Allan variance floor
	Angle random walk, noise (°/√hour)	3.5	Allan variance method
	Scale Factor Error (%)	0.5	Over -40°C to +70°C range
	Nonlinearity (% FS)	0.2	
	Resolution (°/sec)	0.01	
	G-sensitivity (°/sec/*g*)	0.01	With *g*-sensitivity compensation
	Alignment (°)	0.2	With alignment compensation
	Bandwidth (Hz)	30	-3dB Nominal
Acceleration	Range (*g*)	+/- 5	Custom ranges available
	Bias		
	Turn-on to turn-on repeatability (m*g*)	TBD	25°C fixed temperature
	In-Run stability, over temp. (m*g*)	10	Over -40°C to +70°C range
	Short term stability (m*g*)	0.2	15 second Allan variance floor
	Noise (m*g*/√Hz rms)	0.4	
	Scale Factor Error (%)	0.5	Over -40°C to +70°C range
	Nonlinearity (% FS)	0.2	
	Resolution (m*g*)	0.2	
	Alignment (°)	0.2	With alignment compensation
	Bandwidth (Hz)	50	-3dB Nominal
Magnetic Field	Range (Gauss)	+/- 1.2	
	Bias		
	Turn-on to Turn-on repeatability (mGauss)	TBD	
	In-Run stability, over temp. (mGauss)	15	Over -40°C to +70°C range
	Noise (mGauss/√Hz)	TBD	
	Scale Factor (%)	0.7%	
	Nonlinearity (% FS)	0.4	
	Resolution (mGauss)	0.2	
	Alignment (°)	0.2	With alignment compensation
	Bandwidth (Hz)	50	Nominal

Figure B.1: Microstrain detailed specifications [26].

```
% This m file specifies all necessary 3DMG
% constant values and standard deviation values
% to be used in sensor fusion code.
```

3DM-GX1 Gyroscope Specifications

An IMU CLOCK TICK is constant.

```
clock_tick = 0.0065536; %sec
```

The following gets the number of IMU clock ticks for transmitting a packet as per 3DM-GX1 Data Communication Protocol (Comm spec. Rev 3.01) February 17, 2005 - Wireless Transmission, [30].

```
[num,packetLength] = getNumTicks(user_reply);
% Frequency of INS (FASTER)
ins_freq = 1/(num*clock_tick); %Hz
%Effective Bandwidth of INS
% ins_BW = 30; %Hz (-3dB Nominal)
```

3DM-GX1 Timestep dt is the time between each INS measurement based on user selection

```
dt = 1/ins_freq; %s
```

Noise specification terms for 3DM-GX1 are based on Microstrain datasheet [26].

```
% Obtain standard deviation values
% as per 3DM-GX1 Specifications Sheet:

% Obtain gyro sensor angular velocity standard deviation
% from ARW = 3.5 deg/sqrt(hr)
ARW = (3.5*pi)/(180*sqrt(3600)); %rad/sqrt(sec)
Sigma_omega = ARW*sqrt(dt) %rad
```

Sigma_omega = $1.1656e - 004$ radians, (0.0067) degrees.

Alternatively for Rate Random Walk use pre integration. **Not currently implemented**.

$$Sigma\_omega = 60 * ARW * \sqrt{ins\_BW}\frac{rad}{s} \text{ or } Sigma\_omega = \frac{\frac{\frac{3.5*pi}{180}}{\sqrt{\frac{dt}{3600}}}}{3600}\frac{rad}{s}$$

```
% Gyro bias error for each axis (STD) 15 second Allan Variance Floor
Sigma_bias = 0.1*pi/180; %rad/s
% Gyro Scale Factor Error (-40 to 70 deg C) (STD)
Sigma_sf = 0.005; % unitless (5000 ppm or 0.5 percent FS)
% Gyro Axis Misalignment errors (with compensation) (STD)
Sigma_ma = 0.002; % unitless (2000 ppm or 0.2 percent FS)

% Gyro Resolution Error - not currently modelled
% Sigma_res = 0.01*pi/180; %rad/s
% Gyro G-Sensitivity Error - currently not modelled
% Sigma_grav = 0.01*pi/180; %rad/s/g
```

B.6 Motor/IMU Test Setup Specifications

$*$: Values provided from Apex Dynamics for the AD 0471 Gearbox, [35].

Description	Value	Units
Gearbox Reduced Backlash	$\leqslant 3^*$	arcmin
Gearbox Standard Backlash	$\leqslant 5^*$	arcmin
Max Backlash Motor/IMU	$\leqslant 7.5$	arcmin
Max Avail. Torque Motor/IMU (30 rpm)	40	Nm
Max Torque during Tests ($\approx$ 14 rpm)	≈ 20	Nm

Table B.3: Motor/IMU Setup Specs

Figure B.2: Apex Dynamics AD 0471, [35]

Appendix C

Main Matlab Functions

With permission, the following Matlab m files are available from the Carleton CUSP SVN server. The following descriptions are also available in HTML format from the server for easier reading with coloured headings and descriptions.

C.1 RUN_MAIN_PROGRAM.m

This function can currently be used to run recorded test data offline. It oversees each subfunction filter. The only requirement is having the correct ‘alldata.mat’ recorded data file in the same folder as this and all the subfunction m files called.

C.2 UKF.m

Contents

- UKF.m
- Get noise disturbances di (Weights) based on Covariance
- Transform disturbances
- Create Sigma Points from 'augmented' state vector x_prev

- Transform/Propagate xi_prev through system process model to get yi 'augmented' state sigma points
- PREDICTION
- Find P_pri – 'a-priori' state covariance
- Midstep - Added by Jesse Linseman June 17,2009
- PERFORM UPDATE
- Estimate measurement z_hat and covariance Pzz
- For speed savings
- Innovation covariance Pvv
- Compute Kalman Gain (lag compensated)
- A posteriori

```
function [x_post, P_post] = ...
UKF(u_m, z_m, Rvos, x_prev, P_prev, dt,l,tol,CamCalInfo,N,r)
```

NOTE: Kappa is maintained at equal to 0

Inputs:

- u_m - current measured IMU AngVel (including WGN & sensor errors).
- z_m - current 'lagged' VOS measurement q (including WGN).
- R_VOS - current VOS measurement noise covariance.
- x_prev - 'augmented' state vector.
- P_prev - 'augmented' state covariance matrix [30x30].
- dt - IMU timestep (sec).
- l - latency term indicates number of IMU timesteps VOS measurement is lagged by.
- tol - tolerance on mean quaternion.
- CamCalInfo - VOS parameters used in VOSprocess

- N - number of observed markers
- r - Atlas sphere radius

Outputs:

- x_post - 'augmented' posteriori state estimate [31×1]
- P_post - 'augmented' posteriori state covariance Matrix [30×30]

Get noise disturbances di (Weights) based on Covariance

The augmented process covariance matrix P_prev includes state covariance 'P' and process noise covariance 'Q'.

```
    n = size(P_prev, 1); %number of rows (30)
% Initialize
    di = zeros(n, 2*n);
% Obtain sigma points matrix (pre-quaternion sigma points)
    S = chol(n*P_prev); %[30X30]
    for k = 1:n
        di(:, k) = S(:, k);
        di(:, k+n) = -S(:, k);
    end
```

Transform disturbances

aa noise disturbances di(1:3, 1) into quaternion disturbances qi_d(1:4, 1) convert orientation portion of di into quaternion di initialize quaternion disturbance

```
    qi_d = zeros(4, 2*n);
    for k = 1:2*n
```

```
        qi_d(:, k) = aa2q(di(1:3, k));
    end
```

Create Sigma Points from 'augmented' state vector x_prev

initialize q, sensor error and noise portions of sigma points

```
    qi_prev = zeros(4, 2*n);
    si_prev = zeros(12,2*n);
    wi_prev = zeros(15, 2*n);

    for k = 1:2*n %number of Sigma Points to create (columns)
% apply q disturbance by q multiplication (rotate q_prev)
        qi_prev(:, k) = ...
        qmultiply(x_prev(1:4), qi_d(:, k)); %attitude sigma points
        si_prev(:,k) = x_prev(5:16) + di(4:15,k); %sensor error sigma points
        wi_prev(:,k) = di(16:30,k);
    end
% Construct 'Augmented' sigma point matrix
    xi_prev(:,:) = ...
    [qi_prev(:,:); si_prev(:,:); wi_prev(:,:)]; %State Sigma Points
```

Transform/Propagate xi_prev through system process model to get yi 'augmented' state sigma points

```
    yi = zeros(31,2*n);
    for k = 1:2*n
        % Note: no added noise
        yi(:, k) = IOSProcess(u_m(:,1), xi_prev(:, k), dt);
    end
% Obtain initial guess y0 for finding mean state (faster onvergence).
    y0 = IOSProcess(u_m(:,1), x_prev, dt);
```

PREDICTION

A priori state estimate (MEAN) Calculate quaternion portion of mean using Riemannian manifold intrinsic gradient decent algorithm (Kraft 2003)

- q_pri - quaternion associated with attitude portion of state
- q_ei - aa errors associated with (quaternion) attitude portion of state
- s_pri - sensor error portions
- w_pri - noise portions

```
    [q_pri, q_ei] = meanqRotation(yi(1:4,:), y0, tol);
% Calculate vectorial portion using Barycentric mean sensor error portion
    [s_pri, s_ei] = meanSerror(yi(5:16,:));
% noise portions
    [w_pri, w_ei] = meanSerror(yi(17:31,:));

% If noise portions are all assumed zero mean (WGN)
% no need to calculate: - not currently implemented
% ie. ‘a-priori’ noise w_pri expected value always zero (mean)
   %w_pri = zeros(15, 1);

% State estimate (a priori)
     y_pri = [q_pri; s_pri; w_pri];
```

Find P_pri – 'a-priori' state covariance

```
P_pri = zeros(n,n);


for k = 1:2*n
    P_pri = P_pri + 1/(2*n)*([q_ei(:, k); s_ei(:,k);w_ei(:,k)] ...
    *[q_ei(:, k);s_ei(:,k);w_ei(:,k)]');
end
```

Midstep - Added by Jesse Linseman June 17,2009

To deal with latency issue create memory terms

```
persistent y_pri_hold; %variable to hold IMU term in memory
persistent P_pri_hold; %variable to hold IMU term in memory
persistent qi_hold;    %variable to hold 'augmented' IMU sigma states
persistent q_ei_hold;


if isempty(y_pri_hold) || isempty(P_pri_hold) || isempty(q_ei_hold)
% Performed on 1st iteration only
```

- stores current 'a-priori' IMU aug state vector
- stores current 'a-priori' IMU aug state covariance
- stores current 'a-priori' transformed sigma points
- stores current 'a-priori' transformed sigma points errors

```
  y_pri_hold(:,1) = y_pri(1:4);
  P_pri_hold(:,:,1) = P_pri;
  qi_hold(:,:,1) = yi(1:4,:);
  q_ei_hold(:,:,1) = q_ei;

  y_pri_hold(:,2) = y_pri(1:4);
  P_pri_hold(:,:,2) = P_pri;
  qi_hold(:,:,2) = yi(1:4,:);
  q_ei_hold(:,:,2) = q_ei;
end

if l == 0 %VOS measurement available from k-N lagged timesteps back
```

PERFORM UPDATE

ONLY WHEN VOS MEASUREMENT IS AVAILABLE

```
  y_pri_hold(:,1) = y_pri_hold(:,2);    %Use stored estimate
P_pri_hold(:,:,1) = P_pri_hold(:,:,2);
   qi_hold(:,:,1) = qi_hold(:,:,2);
 q_ei_hold(:,:,1) = q_ei_hold(:,:,2);

  y_pri_hold(:,2) = y_pri(1:4);  % Re-store current IOS measurement
P_pri_hold(:,:,2) = P_pri;
   qi_hold(:,:,2) = yi(1:4,:);
 q_ei_hold(:,:,2) = q_ei;
```

C.2.1 Estimate measurement z_hat and covariance Pzz

Use VOSprocess - not needed if increased speed is required Obtain zi sigma points by propagating yi Sigma points through VOS observation model **To use comment**

out following section

```
% zi = zeros(4,2*n);
% for k = 1:2*n
% % Note: no added noise
% [zi(:,k)] = VOSprocess(qi\_hold(:,k,1), CamCalInfo, N, r,0);
% end
% % Initial Guess for mean VOS measurement used for speed savings only
% z0 = VOSprocess(y\_pri\_hold(:,1), CamCalInfo, N, r,0);
% % find z mean estimate z\_hat and errors ei\_z
% [z\_hat, ei\_z] = meanqRotation(zi, z0, tol);
% % VOS Measurement process covariance Pzz
% Pzz = zeros(3, 3);
% for k = 1:(2*n)
% Pzz = Pzz + 1/(2*n)*(ei\_z(:, k)*ei\_z(:, k)’);
% end
```

C.2.2 For speed savings

For speed savings the lagged expected VOS measurement ’z_hat’ can simply be the ’a priori’ ’augmented’ state attitude (quaternion) from k-N timesteps ago.

```
        z_hat = y_pri_hold(:,1);
```

Similarly, the expected measurement state covariance Pz_hat of the VOS measurement would simply be the IOS ’a priori’ state process covariance (P_pri attitude parts only)

```
        Pzz = P_pri_hold(1:3,1:3);
```

Innovation covariance Pvv

```
        Pvv = Pzz+Rvos;
```

Cross correlation covariance Pyz (lag compensated).

```
     Pyz = zeros(30, 3); % Initialize Pyz
     for k = 1:2*n
         %Using VOSprocess function? => un-comment following line
         %Pyz = Pyz + 1/(2*n)*[q_ei(:, k);s_ei(:,k);w_ei(:,k)]*ei_z(:, k)';

         %For speed savings => comment line if not needed
         Pyz = Pyz + 1/(2*n)*[q_ei(:, k);s_ei(:,k);w_ei(:,k)]*q_ei_hold(:,k,1)';
     end
```

Compute Kalman Gain (lag compensated)

```
         K = Pyz*inv(Pvv);
% innovation of attitude
  e_qinnov = qerror(z_m,z_hat(1:4));
% to axis angle
   e_innov = q2aa(e_qinnov);
% state correction in axis angle
      corr = K*e_innov;
% quaternion correction
    q_corr = aa2q(corr(1:3));
```

A posteriori

Add Kalman gain correction terms to states

```
    q_post = qmultiply(q_corr, q_pri);
   se_post = y_pri(5:16) + corr(4:15);
    w_post = y_pri(17:31) + corr(16:30);
% posteriori estimates
    x_post = [q_post; se_post;w_post];
    P_post = P_pri - K*Pvv*K'; %Kalman Update

else
    % No Kalman update
        x_post = y_pri;
        P_post = P_pri;
end
```

C.3 IOSprocess.m

Contents

- IOS Mechanization
- IOS Sensor Error Compensation
- Apply quaternion rotation

```
function x_new = IOSProcess(omega,x,dt)
```

Inputs:

- omega: (rad/s): measured angular velocity
- x: : 'augmented' state vector
- dt: (sec): timestep

Output:

- x_new: : new 'augmented' state vector

Extract Current States From 'x -augmented state vector'

```
    q = x(1:4);    %current attitude (quaternion)

    bias = x(5:7); %current est biases

    sf = x(8:10);  %current est scale factors

    ma = x(11:16); %current est misalignments

    noise = x(17:31); %current est state noises
```

IOS Mechanization

Integrate angular velocity (omega) based on small angle assumption

```
    sigmaIMU = dt*(omega); %rad
```

IOS Sensor Error Compensation

```
    [sigmanew,bias_new,sf_new,ma_new] = senscomp(dt,sigmaIMU,bias,sf,ma,noise);
```

Apply quaternion rotation

axis angle to quaternion rotation 'r'

```
    r = aa2q(sigmanew); %updated for code speed increase by Jesse Linseman

% Applies the quaternion differential (r) through to

% get new orientation

    q_new = qmultiply(q, r);

    if q_new(1)<0

        q_new = -q_new;

    end
```

Recreate new augmented state vector

```
x_new = [q_new;bias_new;sf_new;ma_new;noise];
```

C.4 UKF2.m

This is the same as *UKF.m* but it calls *IOSprocess2.m* instead (see Appendix C.5).

Contents

- Get noise disturbances di (Weights) based on Covariance
- Transform disturbances
- Create Sigma Points from 'augmented' state vector x_prev
- Transform/Propagate xi_prev through system process model to get yi 'augmented' state sigma points
- PREDICTION
- Find P_pri – 'a-priori' state covariance
- Midstep - Added by Jesse Linseman June 17,2009
- Estimate measurement z_hat and covariance Pzz
- Innovation covariance Pvv
- Compute Kalman Gain (lag compensated)
- A posteriori

Also the input line is as follows.

```
function [x_post, P_post] = UKF2(q_m, z_m, Rvos, x_prev, P_prev, dt,l,tol)

% NOTE: Kappa is maintained at equal to 0

%
```

Inputs:

- q_m - current measured IMU attitude (including WGN & sensor errors)
- z_m - current 'lagged' VOS measurement q (including WGN)
- R_VOS - current VOS measurement noise covariance
- x_prev - 'augmented' state vector
- P_prev - 'augmented' state covariance matrix [30x30]
- dt - IMU timestep (sec)
- l - latency term indicates number of IMU timesteps VOS measurement is lagged by
- tol - tolerance on mean quaternion

Otherwise this function is the same as *UKF.m*

C.5 IOSprocess2.m

Contents

- IOS Mechanization
- IOS Sensor Error Compensation.
- Apply quaternion rotation

```
function x_new = IOSProcess2(q_m,x,dt,k)
```

Inputs:

- q_m: (rad): measured IMU orientation (quaternion)
- x: : 'augmented' state vector
- dt: (sec): timestep
- k: : k = 1 indicates when to update q_last_m

Outputs:

- x_new: : new 'augmented' state vector

Extract Current States From 'x -augmented state vector'

```
  q = x(1:4);    %current attitude (quaternion)
  bias = x(5:7); %current est biases
  sf = x(8:10);  %current est scale factors
  ma = x(11:16); %current est misalignments
  noise = x(17:31); %current est state noises
```

IOS Mechanization

```
persistent q_last_m
if isempty(q_last_m)
q_last_m = [1;0;0;0];
end
    qrot = qerror(q_last_m,q_m);
    if k==1
    q_last_m = q_m;
    end
    sigmaIMU = q2aa(qrot);
```

IOS Sensor Error Compensation.

```
    [sigmanew,bias_new,sf_new,ma_new] = senscomp(dt,sigmaIMU,bias,sf,ma,noise);
```

Apply quaternion rotation

Axis angle to quaternion rotation 'r'

```
    r = aa2q(sigmanew); %updated for code speed increase by Jesse Linseman
```

Applies the quaternion differential (r) through to get new orientation.

```
    q_new = qmultiply(q, r);
    if q_new(1)<0
        q_new = -q_new;
    end
```

Recreate new augmented state vector

```
    x_new = [q_new;bias_new;sf_new;ma_new;noise];
```

C.6 senscomp.m

This function gets called by *IOSprocess.m* and *IOSprocess2.m* It also has two internal functions *V2G* & *getTol*.

Contents

- senscomp.m
- Gmatrix calculation
- Tolerance check on errors

Created by Jesse Linseman. This function applies sensor error compensation used by UKF.m or UKF2.m.

```
function [sigmanew,bias,sf,ma] = senscomp(dt,sigma,biasold,sfold,maold,noise)


bias = biasold+noise(4:6);
    bias = gettol(bias);
```

```
biashalf = (biasold+bias)/2; % interpolation
    biashalf = gettol(biashalf);

sf = sfold+noise(7:9);
    sf = gettol(sf);
sfhalf = (sfold+sf)/2; % interpolation
    sfhalf = gettol(sfhalf);

ma = maold+noise(10:15);
    ma = gettol(ma);
mahalf = (maold+ma)/2; % interpolation
    mahalf = gettol(mahalf);

Gmatrix = v2G(sfhalf,mahalf); % Create G matrix
sigmanew = inv(eye(3)+Gmatrix)*(sigma - biashalf*dt - noise(1:3)); %rad
end
```

Gmatrix calculation

```
function G = v2G(A,B)
G = diag(A);
G(1,2) = (B(1));
G(1,3) = (B(2));
G(2,1) = (B(3));
G(2,3) = (B(4));
G(3,1) = (B(5));
G(3,2) = (B(6));
end
```

Tolerance check on errors

Used to keep value zero if very small

```
function valout = gettol(valin)
tol = 0.0000001;
valout = valin;
    for val = 1:size(valin,1)
        if abs(valin(val))< tol
            valout(val)=0;
        end
    end
end
```

C.7 VOSprocess.m

This function represents the VOS process in its entirety. Developed by Jesse Linseman, August 31, 2009. Portions of this function were originally created by Kyle Chisholm for his UKF design which have been modified to fit within one overlaying function herein.

```
% Purpose:
% STAGE 1: Based on an IMU estimated Atlas Sphere orientation quaternion,
% develop the estimated map of marker positions for that orientation.

% STAGE 2: Using this estimated map of marker locations, estimate the
% VOS orientation measurement that is expected observed.
```

```
function  [qi] = VOSprocess(q, CamCalInfo, N, r, sim)

%  *q            current orientation quaternion of Atlas sphere
%                 (TRUE or ESTIMATE)
%  N            number of markers seen in image (usually 3)
%  CamCalInfo   structure of camera parameters (CONSTANT)
```

Extract All necessary variables

```
     Spts = open('SpherePts_r4.75.mat');
     PsAll = Spts.Sphere_points; %mm


 if sim == 1
     % point locations on sphere (used for simulation only)
     Spts = open('SpherePts_r4.75_m.mat');
     PsAll_m = Spts.PsAll_m; %mm
 end


   T = CamCalInfo.T;     % translation from World 2 Cam frame (rel to world)
   M = CamCalInfo.M;     % perspective projection matrix
Rcam = CamCalInfo.Rcam;  % rotation matrix from world to camera coordinates
  ox = CamCalInfo.O(1);  % optical x-axis location in pixel image
  oy = CamCalInfo.O(2);  % optical y-axis location in pixel image
  fx = CamCalInfo.F(1);  % focal x length scaling factors
  fy = CamCalInfo.F(2);  % focal y length scaling factors
 Rim = CamCalInfo.Rim;   % Pixel location points covariance matrix
   n = size(PsAll, 2);   % total number of markers
```

STAGE 1: Create Map Of World Marker Locations Based On current Sphere Orientation

Used to get markers in field of view (FOV) transform current quaternion into rotation matrix.

```
Rs2w = q2R(q);
% Transform true sphere marker point positions from local to world frame
Pw = Rs2w*PsAll;
% Transform all sphere marker points from world to camera frame
Pc = Rcam*(Pw-T*ones(1, n));
% Determine absolute distances from camera origin
% to all true sphere marker points
d = sqrt(Pc(1, :).^2+Pc(2, :).^2+Pc(3, :).^2);
```

Find N closest sphere marker points to camera origin sort d from shortest to largest distances

```
[d_sorted IX] = sort(d);


Psfov = zeros(3*N, 1); %(9x1)
Psfov_m = zeros(3*N, 1); %used for simulation only
Pimfov = zeros(2*N, 1); %(6x1)
for k = 1:N
% get expected local sphere marker point 'k' in camera Field Of View (FOV)
    Psfov(3*k-2:3*k, 1) = PsAll(:, IX(k));

    if sim == 1 % *code used for simulation only*
        Psfov_m(3*k-2:3*k, 1) = PsAll_m(:, IX(k));
    end

% transform sphere marker point 'k' from local to world frame
    Pw = Rs2w*Psfov(3*k-2:3*k, 1);
% use projection perspective matrix M
    Z = M*[Pw; 1];
% determine 2D pixel coords from Z
    IM = Z/Z(3);
% determine x',y' image frame coords of points in fov
    Pimfov(2*k-1:2*k, 1) = IM(1:2, 1);
end

if sim == 1 % *code used for simulation only*
    Pimfov_m = Pimfov+chol(Rim)*randn(2*N, 1);
    Pm = [Psfov_m; Pimfov_m];
    save('Pm.mat', 'Pm');
    Pimfov = Pimfov_m; %set for use in Stage 2
end
```

STAGE 2: Obtain VOS Measurement Estimate

Use developed marker position map to determine the estimated VOS orientation quaternion measurement.

```
% rotate translation to make rel to cam coords
P = -Rcam*T;

% Extract P components (camera pos in world frame rel to cam frame)
```

```
Px = P(1);
Py = P(2);
Pz = P(3);

m = size(Pimfov, 1); %usually 6
Pw = zeros(3, m/2); %(3x3)

for k = 1:N
% Extract camera frame x,y pixel location for marker 'k'.
    xim = Pimfov(2*k-1, 1);
    yim = Pimfov(2*k, 1);

% Image plane coords in mm:
    x = (ox-xim)/fx;
    y = (oy-yim)/fy;

% Using Atlas sphere radial constraint, find camera frame image point coord
% with respect to world (relative to camera frame).
    Wx =  -x*(-Px*x-Py*y+x^2*Pz+y^2*Pz+ ...
         (r^2+2*x*Pz*Px-x^2*Pz^2-Px^2-y^2* ...
         Pz^2-Py^2+2*y*Pz*Py+2*Px*x*Py*y+x^2* ...
         r^2-x^2*Py^2+y^2*r^2-y^2*Px^2)^(1/2))/(x^2+y^2+1)+x*Pz-Px;
    Wy =  -y*(-Px*x-Py*y+x^2*Pz+y^2*Pz+ ...
        (r^2+2*x*Pz*Px-x^2*Pz^2-Px^2-y^2* ...
        Pz^2-Py^2+2*y*Pz*Py+2*Px*x*Py*y+x^2* ...
        r^2-x^2*Py^2+y^2*r^2-y^2*Px^2)^(1/2))/(x^2+y^2+1)+y*Pz-Py;
    Wz =  -sqrt(r^2-Wx^2-Wy^2);
    Pcw = [Wx; Wy; Wz];


% Rotate Pcw to world frame coordinates.
    Pw(:, k) = Rcam'*Pcw;
end


% Rotation matrix (only works with N = 3):
    Mnew = Pw*inv(reshape(Psfov, 3, 3));


% Singular Value Decomposition:
    [U,S,V] = svd(Mnew);


% S = (Mnew'*Mnew)^0.5;
% Rnew = Mnew*inv(S);
    Rnew = U*V';


% Convert to quaternion.
    qi = R2q(Rnew);
```

C.8 3DMG_operate.m

This function is used to gather/record a set amount of data from the Microstrain 3DM-GX1 IMU. The user must choose the appropriate command to have the IMU send the proper quantities.

Created by Jesse Linseman Carleton University July 10 2009

```
close all;
```

```
clear all;

clc;

disp('Turn 3DMG off and then back on again');

pause;
```

Set up the serial port

```
s = serial('COM1','BaudRate',38400,'Timeout',2);

% This sets up the serial connection file that

% Matlab needs in order to connect to the serial port
```

Commands List

This is a list of important hexadecimal commands for the 3DM-GX1. All commands are one byte in length. Some commands require additional data bytes following the initial command byte to fully define the action to be taken. All commands generate a response of a fixed number of bytes.

```
%         COMMAND (##)     DEFINITION

% Hexidecimal   Decimal

%   00              0      Null Command (not implemented)

%   01              1      Send Raw Sensor Bits

%   02              2      Send Gyro-Stabilized Vectors

%   04              4      Send Instantaneous Quaternion

%   05              5      Send Gyro-Stabilized Quaternion

%   06              6      Capture Bias (only when stationary)

%   07              7      Send Temperature

%   08              8      Send EEPROM Value

%   09              9      Write EEPROM Value
%   0C             12      Send Gyro-Stabilized Quaternion & Rate Vector
%   0F             15      Tare Coordinate System
%   10 0 ##        16 0 ## Sets Continuous mode (along with a command ##)
%   10 0 0         16 0 0  Set Polled mode
%   11             17      Remove Tare
%   12             18      Send Gyro-Stab. Quaternion & Inst. Vectors
%   24             36      Write System Gains
%   25             37      Read System Gains
%   27             39      Self Test
%   28             40      Read EEPROM Value with Checksum
%   29             41      Write EEPROM Value with Checksum
%   30             48      Send Gyro-Stab. Euler Angles & Rate Vector
%   31             49      Send Gyro-Stab. Euler Angles & Accel & Rate Vector
%   F0            240      Send Firmware Version Number
%   F1            241      Send Device Serial Number

disp('Please choose a decimal command #:');
disp('2 => Send Gyro Stabilized Angular Rate Vectors');
disp('4 => Send Instantaneous Quaternion');
disp('5 => Send Gyro Stabilized Quaternion');
disp('12 => Send Gyro-Stabilized Quaternion & Rate Vector');
disp('48 => Send Gyro-Stabilized Euler Angles & Rate Vector');
user_reply = input('Please choose...  ');

while (isempty(user_reply))
    user_reply = input('invalid response...   Please try again:');
end
```

Hexadecimal example

Comm mode 0x31 - Gyro-stabilized Euler Angles & Rate Vectors command = hex2dec('31');

```
% A Typical Response is 23 bytes including:
% |Header byte|, |Intervening bytes|, |16 bit checksum|

% TYPE                       DESCRIPTION
% Header byte             => same value as the corresponding command byte
% Intervening bytes       => a series of 16 bit signed integers (actual data)
% 16 bit checksum         => sum of all 16 bit integers & header byte (MSB = 00)
```

Runtime Details

```
RunTime = 120; %sec (approx amount of time to record (in seconds))
```

Obtain 3DM-GX1 Details

```
IMUspecs; %Gather IMU specifications of 3DMG based on |user_reply|.
% The above also determines |packetLength| based on |user_reply|.

% Approx. number of discrete simulation steps
t_steps = floor(RunTime/dt); %unitless (floor - answer always rounded down)

% Mean tolerance for convergence of
% estimated quaternion.
tol = 0.000001; %rad
```

Scaling constants for the data

```
     scaling = [1/8192; 360/65536; 7000/32768000; ...
                8500/32768000];
```

Initialize Arrays for speed

anonymous function needed to calculate the unsigned 16bit int

```
un = @(msb,lsb) double(256*msb+lsb);
IMUstats; %Create statistical values for use in UKF
IMUdata = zeros(packetLength,t_steps); %original data packet (no change)
IMUconv = zeros(floor((packetLength-3)/2),t_steps); %2's complement number
StabQ = zeros(4,t_steps);
StabAngRate = zeros(3,t_steps);
```

Put 3DM-GX1 into Continuous Mode

RS-232 only

```
disp('Press any key to start recording')
pause;

try
```

Header Packet

```
    fopen(s);               % Open the serial connection
    fwrite(s,[16 0 user_reply(1)]); % Command for IMU

    % Check & Remove Header
```

```
Header_bytes = 7; % not used

Discraded_header = fread(s,Header_bytes);
```

Record IMU Data

```
for i = 1:t_steps

    packet = double(fread(s,packetLength));

    IMUdata(:,i) = packet;
```

Received checksum

```
    checksum = un(packet(packetLength-1),packet(packetLength));
```

Computed checksum from the packet

```
    computed = un(0,packet(1)); % MSB = 0


    for j = 1:floor((packetLength-3)/2)

         temp = un(packet(2*j),packet(2*j+1));

         computed = computed + temp;

         % correct data to a 2's complement number.

         if temp > 32767

             temp = temp-65536;

         end

         IMUconv(j,i) = temp;

    end
```

Compare the two checksums

Rule: The last two bytes in a recieved packet must equal the sum of the preceeding bytes (including header byte) in the packet to be valid

```
        if mod(computed,65536) ~= checksum
            error('IMUSerialReceive:ChecksumError','Checksum error from IMU');
        end
```

Compare Packet Header

```
if packet(1) ~= user_reply(1)
  error('IMUSerialReceive:misalignment','IMU packet handling is not correct');
end
```

Scale Packet to SI Units

This section depends on the `user_reply` selection

```
 if (user_reply == 4) || (user_reply == 5) || (user_reply == 12)
   StabQ(:,i) = scaling(1)*IMUconv(1:4,i);
 elseif user_reply == 2
   StabAngRate(:,i) = scaling(4)*IMUconv(7:9,i); %Angular Rate Vector (rad/s)
 end
end
   fclose(s);  %Close connection to IMU.
   delete(s);  %Cleanup the serial port.
catch
% If there is an error
   fclose(s);      % close connection
   delete(s);      % delete the object (required)
   rethrow(lasterror);
end
```

Visualize Results

```
if (user_reply == 4) || (user_reply == 5) || (user_reply == 12)
    PlotStabQ;  % Visualize Quaternion
    % Convert StabQ back to aa to visualize.
    PlotAxisAng; % Visualize Axis Angle Results.
end


if user_reply == 2
    PlotStabAngRate; %Visualize Angular Rate
end
```

C.9 Simple_Sensor_Fusion Function

Contents

Simple_Sensor_Fusion.m

Original SSF created to handle Euler angle only by Brian Rasquinha 2008/2009 CUSP. The following SSF code was developed by Jesse Linseman, Carleton University, July 20 2009 for use with quaternions. Note: No statistical information used.

```
function SF_quat = Simple_Sensor_Fusion(u_m,z_m,dt,l)
```

Inputs:

- u_m - current measured IMU AngVel (including WGN & sensor errors)

- z_m - current 'lagged' VOS measurement q (including WGN)
- dt - IMU timestep (sec)
- l - latency term indicates number of IMU timesteps VOS measurement is lagged by

Outputs:

- SF_quat - Sensor Fusion estimated Quaternion

Allocate Memory Terms

```
persistent Corr

persistent IMU_q;
```

Performed on First Iteration Only

```
if isempty(Corr)||isempty(IMU_q)

    Corr = [1;0;0;0];

    IMU_q = [1;0;0;0];

end
```

Integrate to get IMU Orientation Quaternion

```
sigmaIMUaa = dt*u_m;

r = aa2q(sigmaIMUaa);

IMU_q = qmultiply(IMU_q, r);

    if IMU_q(1)<0

        IMU_q = -IMU_q;

    end
```

Update Correction Term If proper VOS measurement available

```
if l == 0;

% We have vision data, so update correction and

% store current IMU_q for later use

    Corr = getnewCorr(z_m,IMU_q);

end
```

Apply most recent VOS correction factor Corr

```
SF_quat = qmultiply(IMU_q,Corr);

    if SF_quat(1)<0

        SF_quat = -SF_quat;

    end

end
```

getnewCorr.m

Internal function to SSF.m used to handle OOSM from VOS. Obtains the valid correction term to be used at the current timestep and stores the current IMU quaternion for later use.

```
function Corr = getnewCorr(z_m,IMU_q_store)


    persistent IMU_q_hold


    if isempty(IMU_q_hold)

        IMU_q_hold = [1;0;0;0];

    end
    Corr = qerror(z_m,IMU_q_hold);

    IMU_q_hold = IMU_q_store;

end
```

C.10 UKF No Sensor Error Estimation Function

Contents

- UKF_NoSE.m
- findZmod.m
- IOSProcess_NoSE.m

UKF_NoSE.m

from VOS_UKF developed by Kyle Chisholm, Carleton University comments added by Jesse Linseman for clarity findZmod.m moved in as internal function. IOSProcess_NoSE.m moved in as internal function.

```
function [q_post, P_post] = ...

UKF_NoSE(u_m, z_m, Rvos, q_prev, P_prev, Qw, dt, l, tolerance)
```

Inputs

- u_m - Angular Rates from IOS
- z_m - VOS lagged measurement
- Rvos - VOS measurement covariance matrix
- q_prev - previous state quaternion estimate
- P_prev - preious state covariance matrix estimate
- Qw - state noise covariance matrix
- dt - timestep

- l - lag term
- tolerance - tolerance for mean quaternion calc

Outputs

- q_post - posterior state measurment estimate
- P_post - posteriori state covariance estimate

```
% from generalized UKF, Kappa is equal to 0
% Get noise disturbances di
% size of covariance matrix
n = size(P_prev, 1);
% Initialize
di = zeros(n, 2*n);
% get sigma matrix
S = chol(n*P_prev);
for k = 1:n
    di(:, k) = S(:, k);
    di(:, k+n) = -S(:, k);
end
% Transform aa noise disturbances di(:, 1) to q state disturbances qi_d

% initialize quaternion disturbance
qi_d = zeros(4, 2*n);
for k = 1:2*n
    % axis angle to quaternion
    qi_d(:, k) = aa2q(di(:, k));
end
```

```
% create Sigma Points of state x

% initialize sigma points
qi_prev = zeros(4, 2*n);
for k = 1:2*n
   % apply q disturbance by q multiplication (rotate q_prev)
   qi_prev(:, k) = qmultiply(q_prev, qi_d(:, k));
end

% Propagate xi_prev through process model to get xi

% initialize qi
qi = zeros(4, 2*n);
for k = 1:2*n
    % run through process
    qi(:, k) = IOSProcess_NoSE(u_m, qi_prev(:, k), dt);
end

% Find mean quaternion q_pri (a priori) and aa errors ei

% initial guess for q_pri (mean rotation)
q0 = IOSProcess_NoSE(u_m, q_prev, dt);
% find q_pri and ei
[q_pri, ei] = meanqRotation(qi, q0, tolerance);

% A priori state covariance without process noise
% covariance Pq
Pq = zeros(3, 3);
for k = 1:(2*n)
    Pq = Pq + 1/(2*n)*(ei(:, k)*ei(:, k)');
end
```

Find Q (process noise) create ui sigma points

```
m = size(Qw, 1);
S = sqrtm(m*Qw);
ui = zeros(m, 2*m);
for k = 1:m
    ui(:, k) = u_m + S(:, k);
    ui(:, k+m) = u_m - S(:, k);
end
% propagate ui through process model
qi_u = zeros(4, 2*m);
for k = 1:2*m
    qi_u(:, k) = IOSProcess_NoSE(ui(:, k), q_prev, dt);
end
% find qu_mean
[qu_mean, eu_i] = meanqRotation(qi_u, q0, tolerance);
% process noise Covariance
Q = zeros(n, n);
for k = 1:(2*n)
    Q = Q + 1/(2*n)*(eu_i(:, k)*eu_i(:, k)');
end
```

```
% A priori covariance
P_pri = Pq+Q;
% Estimate measurement z_hat and covariance Pz_hat
% Since z is simply the state quaternion
z_hat = q_pri;
Pz_hat = Pq;
% modify state measurement forward in time to account for OOSM
[z_mod, Rvmod] = ...
            findZmod(q_pri, P_pri, q_prev, P_prev, z_m, Rvos, l, tolerance);
% Innovation Covariance Pz
Pz = Pz_hat+Rvmod;
% Cross Covariance Pqz is simply Pq
Pqz = Pq;
% Compute Kalman Gains
K = Pqz*inv(Pz);
% this gain corresponds to noise in aa
% we need to convert innovation to correction of x_pri in quaternion
% innovation
e_qinnov = qerror(z_mod, z_hat);
% to axis angle
e_innov = q2aa(e_qinnov);
% state correction in axis angle
a_corr = K*e_innov;
% quaternion correction
q_corr = aa2q(a_corr);
```

A posteriori

```
q_post = qmultiply(q_corr, q_pri);

P_post = P_pri - K*Pz*K';

P_post = real(P_post);

end
```

findZmod.m

Used to modify VOS measurements forward in time for OOSM handling for UKF_NoSE.

```
function [z_mod, Rvmod] = ...
findZmod(q_pri, P_pri, q_prev, P_prev, z_m, Rvos, k, tolerance)
persistent q_hold;
persistent P_hold;
persistent holdit;
% if first run
if isempty(q_hold) || isempty(P_hold)
    q_hold(:, 1) = q_prev;
    P_hold(:, :, 1) = P_prev;
    q_hold(:, 2) = q_prev;
    P_hold(:, :, 2) = P_prev;
    holdit = 0;
% if q_prev (a posteriori) is just updated at time k-1,
% use q on hold for OOSM and save current q_prev for later
elseif k == 0
    q_hold(:, 1) = q_hold(:, 2);
    P_hold(:, :, 1) = P_hold(:, :, 2);
    holdit = 1;
```

```
elseif holdit == 1
    q_hold(:, 2) = q_prev;
    P_hold(:, :, 2) = P_prev;
    holdit = 0;
end
    % create covariance matrix of "inputs" to OOSM handling process
    Q = blkdiag(Rvos, P_pri, P_hold(:, :, 1));
    % disturbances
    % size of covariance matrix
    n = size(Q, 1);
    % Initialize
    di = zeros(n, 2*n);
    % get sigma matrix
    S = real(sqrtm(n*Q));
    for k = 1:n
        di(:, k) = S(:, k);
        di(:, k+n) = -S(:, k);
    end
    % convert di to quaternions
    % initialize quaternion disturbance
    % from camera (VOS measurement)
    Dzi_c = zeros(4, 2*n);
    % a priori
    Dqi_pri = zeros(4, 2*n);
    % a posteriori from time camera took image
    Dqi_hold = zeros(4, 2*n);
    for k = 1:2*n
        Dzi_c(:, k) = aa2q(di(1:3, k));
        Dqi_pri(:, k) = aa2q(di(4:6, k));
        Dqi_hold(:, k) = aa2q(di(7:9, k));
    end
    % initialize sigma points
    zi_c = zeros(4, 2*n);
    qi_pri = zeros(4, 2*n);
    qi_hold = zeros(4, 2*n);
    for k = 1:2*n
       % apply q disturbance by q multiplication
       zi_c(:, k) = qmultiply(z_m, Dzi_c(:, k));
       qi_pri(:, k) = qmultiply(q_pri, Dqi_pri(:, k));
       qi_hold(:, k) = qmultiply(q_hold(:, 1), Dqi_hold(:, k));
    end
    % initialize zi
    zi = zeros(4, 2*n);
    for k = 1:2*n
        % OOSM handling process
        q_rot = qerror(qi_pri(:, k), qi_hold(:, k));
        zi(:, k) = qmultiply(q_rot, zi_c(:, k));
    end
    % initial guess for z
    q_err = qerror(q_pri, q_hold(:, 1));
    z0 = qmultiply(q_err, z_m);
    % find mean measurement z and rotation errors ei
    [z_mod, ei] = meanqRotation(zi, z0, tolerance);
    % measurement covariance Rvmod
```

```
    Rvmod = zeros(3, 3);
    for k = 1:(2*n)
        Rvmod = Rvmod + 1/(2*n)*(ei(:, k)*ei(:, k)');
    end
end
```

IOSProcess_NoSE.m

Internal IOS process function used to integrate IOS measurements.

```
function q_new = IOSProcess_NoSE(omega, q, T)
%extract states
%integrate omega
sigma = T*(omega);
%axis angle to quaternion r
sigma_mag = sqrt(max(0, sigma'*sigma));
if sigma_mag == 0;
    ac =1;
    as = 0;
else
    ac = cos(sigma_mag/2);
    as = sin(sigma_mag/2)/sigma_mag;
end
r = [ac;
     as*sigma];
% Apply rotation
q_new = qmultiply(q, r);
```

```
if q_new(1)<0
    q_new = -q_new;
end
end
```

C.11 TrueTraj.m

This function creates an array of the TRUE rotation profile attempted to be followed by the gearbox July 13, 2009, 120 second data set

```
L = size(yout,1); %number of recorded data points
t = 0.0005; % increment (sec/datapoint).
TrueAng = zeros(L,1);
elapsedtime = 0;
for k = 1:L
    TrueAng(k,1) = getMotorAng(elapsedtime);
    elapsedtime = elapsedtime + t ;
end
```

Calculate RMS Error of Motordata

```
err = zeros(1,L);
RMS = 0;
maxTheta = 0;
for k = 1:L

    err = TrueAng(k,:)-yout(k,1);
    ThetaSqr = err^2;
    if sqrt(ThetaSqr) > maxTheta
        maxTheta = sqrt(ThetaSqr);
    end
    RMS = RMS + (1/L)*ThetaSqr;
end

Results = {'RMS_degrees','Max_degrees';RMS,maxTheta};
```

C.12 getMotorAng.m

Created by Jesse Linseman July 13, 2009 Used to command gearbox to angular position

Input:

- elapsedtime: seconds: amount of time since start

Output:

- MotorAng: degrees: Desired motor position

```
function MotorAng = getMotorAng(elapsedtime)
```

Conversions:

```
% MaxMotorRPM = 10; %Rev/min
% Motorrps = MaxMotorRPM/60; %rev/s
% Motordps = Motorrps*360; %degrees/s
if elapsedtime <= 10
% Hold for 10 sec and start IMU
```

```
  MotorAng = 0;
elseif elapsedtime <= 40
% 5 rotations (constant AngVel)
  MotorAng = (elapsedtime-10)*60; %Constant Rotation at 10 RPM to 1800
elseif elapsedtime <= 45
% Hold still for 5 sec
  MotorAng = 1800;
elseif elapsedtime <= 115
% Back and forth 120 degrees from zero position
% cosine non-constant ang rate
% Note: Maximum RPM is < 10
%(actually 57.6 deg/s => 9.6 Rev/min)
  MotorAng = 1800+120*sin(0.48*(elapsedtime-45));
else
% hold still for last 5 seconds
  MotorAng = 1800+120*sin(0.48*(115-45));
end
```

Note: Total elapsed time should be 120 seconds.

C.13 Convertmotordata.m

This function is used to synchronize 3DM-GX1 IMU data and true Motor Encoder data which are recorded separately but at approximately the same time. Matlab plots are used to determine the closest point for synchronization and the data is shifted accordingly. It is required that the motor recording is started before the IOS to properly utilize this function.

Contents

```
%Created July 10, 2009

%By Jesse Linseman

%Carleton University


%Modified:

%Author:                Date:                 Description of Changes:

%Jesse Linseman         July 17 2009          Axis Angle Only

% This code converts 2Khz motor data into 76Hz true path synchronized data

% for comparing with the 3DMG IMU data
```

MOTOR DATA MANIPULATION (True Data)

```
%Need to create Motor Orientation Quaternion Right Away

L = size(yout,1);


motorAngPos = (pi/180)*yout(:,1);

motorAngVel = (pi/180)*yout(:,2);

% clear yout;

clc;

%IMU_Axis_of_Rot = input('Please Enter Axis Of Rotation (eg [#;#;#]):  ');
```

Pure Yaw 30 second test	120 second test
IMU_Axis_of_Rot= $\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$	IMU_Axis_of_Rot= $\begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$

```
IMU_Axis_of_Rot = [0;1;-1]
Mag_IMU_Axis_of_Rot = sqrt((IMU_Axis_of_Rot)'*IMU_Axis_of_Rot);
Unit_IMU_Axis_of_Rot = IMU_Axis_of_Rot./Mag_IMU_Axis_of_Rot;

clear IMU_Axis_of_Rot, clear Mag_IMU_Axis_of_Rot

%Motor Position Data
preMOTORaa = zeros(3,L);
preMOTORaa_mag = zeros(1,L);
preMOTORq = zeros(4,L);
postMOTORaa = zeros(3,L);
postMOTORaa_mag = zeros(1,L);



for i = 1:L
   preMOTORaa(:,i) = motorAngPos(i,1)*Unit_IMU_Axis_of_Rot;
   preMOTORaa_mag(:,i) = sqrt(preMOTORaa(:,i)'*preMOTORaa(:,i));
   preMOTORq(:,i) = aa2q(preMOTORaa(:,i));
   postMOTORaa(:, i) = q2aa(preMOTORq(:,i));
   postMOTORaa_mag(:,i) = sqrt(postMOTORaa(:, i)'*postMOTORaa(:, i));
end
```

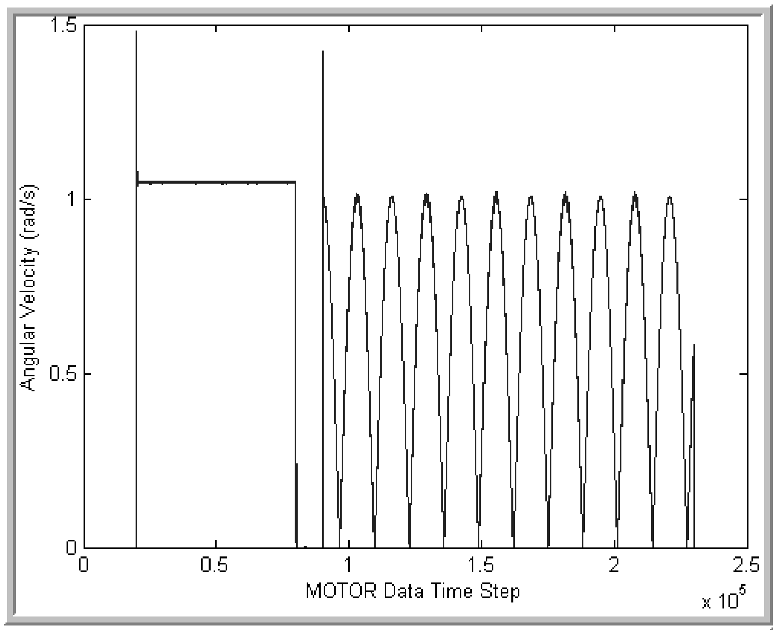
```
%Motor Angular Velocity Data
premotorAngVel = zeros(3,L);

for i = 1:L
premotorAngVel(:,i) = motorAngVel(i,:)*Unit_IMU_Axis_of_Rot;
end

%%Check Motor Position Data
% hold off;
%         figure(1);
%         plot(preMOTORaa_mag(1,:),'-b');  %Motor (Actual) Data
%         hold on;
%         plot(postMOTORaa_mag(1,:),'-r');  %Motor (Actual) Data
%         %plot(motorAngPos(:,1),'-k'); %Motor (Actual) Data
%         title('Motor Axis Angle Magnitude Comparison')
%         xlabel('MOTOR Data Time Step')
%         ylabel('Axis Angle (rad)')
%         hold off;
motor_freq = 2000; %Hz

clc;
disp('In order to synchronize the two data sets,')
disp('Matlab plots are utilized to obtain two data points corresponding to the')
disp('same instant in time.')
disp('')
disp('ie. Choose a peak or trough that correspond to one another')
```

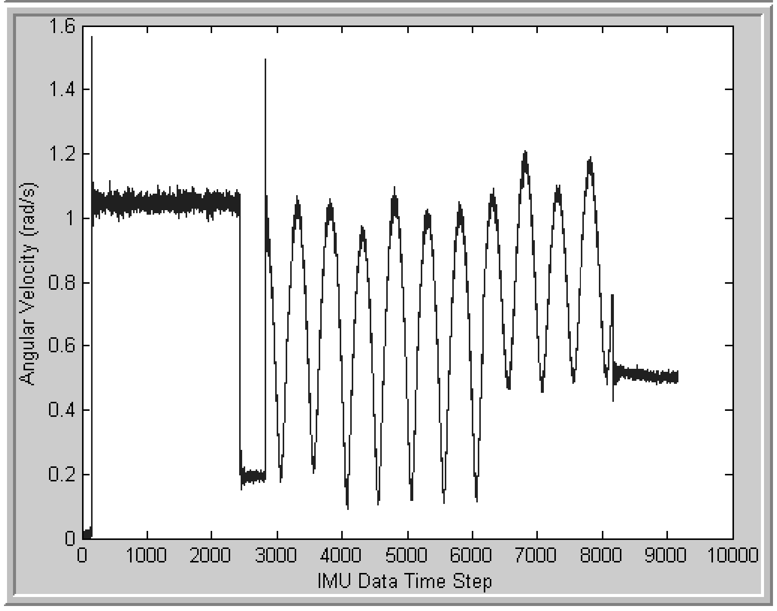
In order to synchronize the two data sets, Matlab plots are utilized to obtain two data points corresponding to the same instant in time. ie. Choose a peak or trough that correspond to one another

Using IOS Stabilized Rate Vector Data

```
if user_reply == 2
hold off;
% Motor
  motorAngVel_mag = zeros(1,L);
  for k = 1:L
     motorAngVel_mag(:,k) = sqrt(premotorAngVel(:,k)'*premotorAngVel(:,k));
  end

figure(1);
plot(motorAngVel_mag());  %Motor (Actual) Data
title('Motor Angular Velocity Magnitude')
xlabel('MOTOR Data Time Step')
ylabel('Angular Velocity (rad/s)')
hold off;

% IMU
  L = size(StabAngRate,2);
  StabAngRate_mag = zeros(1,L);
  for k = 1:L
      StabAngRate_mag(:,k) = sqrt(StabAngRate(:,k)'*StabAngRate(:,k));
  end
```

```
figure(2);

plot(StabAngRate_mag());  %IMU Data

title('IMU Angular Velocity Magnitude')

xlabel('IMU Data Time Step')

ylabel('Angular Velocity (rad/s)')

hold off;

end
```





Using IOS Instantaneous ‘Gyro-Stabilized’ Quaternion Data

This function can also be used on non-stabilized Inst. quaternion output.

```
if (user_reply == 5) || (user_reply == 4)
```

In order to get the average of the beginning stationary IMU data, use ‘*meanqRotation.m*’ to average about the Riemannian Manifold for N ‘stationary’ quaternion data points

```
% Choose N dynamic data steps from first '2' seconds of IMU stationary data
    steps = floor(2/dt); %number of stationary data points to average
    stationaryIMUq = StabQ(:,50:steps+50);
    [startIMUq, ei] = meanqRotation(stationaryIMUq, StabQ(:,1), tol);

% Obtain rotation quaternion to rotate all IMU data into Motor orientation
    IMU2Motq = qerror([1;0;0;0],startIMUq);

    L = size(StabQ,2);
    preIMUq = zeros(4,L);
    preIMUaa = zeros(3,L);
    preIMUaa_mag = zeros(1,L);

    for i = 1:L
        preIMUq(:,i) = qmultiply(IMU2Motq, StabQ(:,i));
        preIMUaa(:, i) = q2aa(preIMUq(:,i));
        preIMUaa_mag(:, i) = sqrt(preIMUaa(:, i)'*preIMUaa(:, i));
    end

%Plots
```

```
  %Motor
  hold off;
  figure(1);
  plot(postMOTORaa_mag(),'-r');  %Motor (Actual) Data
  title('Motor Axis Angle Magnitude')
  xlabel('MOTOR Data Time Step')
  ylabel('Axis Angle (rad)')
  hold off;

  %IMU
  hold off;
  figure(2);
  plot(preIMUaa_mag(),'-b');  %IMU Data
  title('IMU Axis Angle Magnitude')
  xlabel('IMU Data Time Step')
  ylabel('Axis Angle (rad)')
  hold off;
end

choice = 0;
while choice ~= 1

    clc;
    disp(' In order to synchronize the two data sets,' )
    disp(' look at Matlab plots and choose' )
    disp(' the x-axis data_step corresponding to the same instant in time.')
    %IMUpt = input('enter IMU TIMESTEP: ');
```

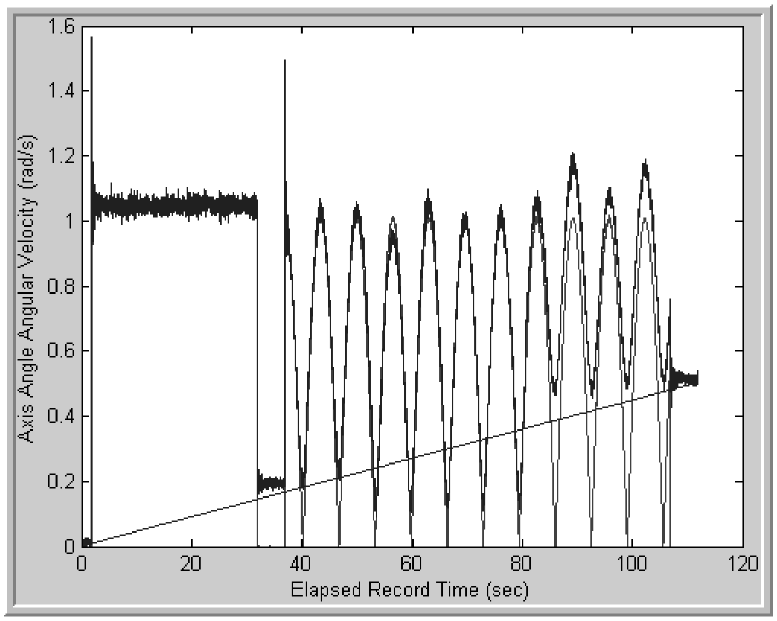
Example Choice: $IMUpt = 2432$ - Use plots $MOTORpt = 80005$ - Use plots

```
IMUpt = 2432
%MOTORpt = input('enter approx corresponding MOTOR TIMESTEP: ');
MOTORpt = 80005
firstmotorpt = floor(MOTORpt-(IMUpt)*(motor_freq/ins_freq));

firstIMUpt = 1;

lastIMUpt = size(StabQ,2);

if user_reply == 2
AngRatedatamanipulation;
end

if user_reply == 5
Posdatamanipulation;
end

clc;
RMSvalue;

disp('Satisfied? Type 1');
disp('Type 1');
disp('otherwise, to improve choice');
%choice = input('Type 2: ');
```

```
    choice = 1
end
```

RMS_AxisAngle_Error_in_Degrees = 74.0519 Maximum_Error = 178.7743

```
Satisfied? Type 1
Type 1
otherwise, to improve choice
choice =

     1
```



AngRatedatamanipulation.m

Internal subfunction within Convertmotordata.m used to integrate and compare synchronized data sets from Motor and IMU.

Contents

- AngRate DATA MANIPULATION

- ANGULAR VELOCITY DATA (rad/s)
- POSITION DATA (rad)

AngRate DATA MANIPULATION

```
% Assume Initial IMU Position is Zero [1;0;0;0];
IMU_q = [1;zeros(30,1)];


L = lastIMUpt-(firstIMUpt-1);
MOTORandIMUquatsync =zeros(9,L); %initialize array


MOTORandIMUAngVelsync = zeros(7,L);
MOTORAngVelsync_mag = zeros(1,L);
IMUAngVelsync_mag = zeros(1,L);
try
    k =0;
    for i = firstIMUpt:lastIMUpt
        k=k+1;
```

ANGULAR VELOCITY DATA (rad/s)

```
% Motor Angular Velocity (rad/s)
  MOTORandIMUAngVelsync(1:3,k) = ...
  premotorAngVel(:,floor(firstmotorpt+(k-1)*(motor_freq/ins_freq)));


  MOTORAngVelsync_mag(k)= ...
  sqrt(MOTORandIMUAngVelsync(1:3,k)'*MOTORandIMUAngVelsync(1:3,k));


% IMU Angular Velocity (rad/s)
```

```
  MOTORandIMUAngVelsync(4:6,k) = ...
  StabAngRate(1:3,i);
  IMUAngVelsync_mag(k)= ...
  sqrt(MOTORandIMUAngVelsync(4:6,k)'*MOTORandIMUAngVelsync(4:6,k));
```

ANGULAR POSITION DATA (rad)

```
% Integrate IMU AngVel to get IMU Position Data
% Motor Position (quaternion)
  MOTORandIMUquatsync(1:4,k) = ...
  preMOTORq(:,floor(firstmotorpt+(k-1)*(motor_freq/ins_freq)));

% IMU Position (quaternion)
  IMU_q = ATLASProcess(MOTORandIMUAngVelsync(4:6,k),IMU_q,dt);
  MOTORandIMUquatsync(5:8,k) = IMU_q(1:4);

% TIME SIGNATURE
% Elapsed seconds
  MOTORandIMUquatsync(9,k) = dt*i;
  MOTORandIMUAngVelsync(7,k) = MOTORandIMUquatsync(9,k);
end

figure(3);
%Motor (True) Data
plot(MOTORandIMUAngVelsync(7,1:L),MOTORAngVelsync_mag(), 'r-');
hold on
%IMU (Sensor) Data
plot(MOTORandIMUAngVelsync(7,1:L),IMUAngVelsync_mag(), 'b-');
```

```
title('Axis Angle Angular Velocity Magnitude Comparison')
xlabel('Elapsed Record Time (sec)')
ylabel('Axis Angle Angular Velocity (rad/s)')
hold off;


catch
figure(3);
%Motor (True) Data
plot(MOTORandIMUAngVelsync(7,1:L),MOTORAngVelsync_mag(), 'r-');
hold on
%IMU (Sensor) Data
plot(MOTORandIMUAngVelsync(7,1:L),IMUAngVelsync_mag(), 'b-');
title('Axis Angle Angular Velocity Magnitude Comparison')
xlabel('Elapsed Record Time (sec)')
ylabel('Axis Angle Angular Velocity (rad/s)')
hold off;
end
```

C.14 UKFsim.M

Script for simulating UKF created by Jesse Linseman May 2009 for inclusion of IMU bias, sensor misalignment, sensor scale factor estimation and general speed savings.

```
clc;
disp('GPSUKFSIM: Please wait while the simulation runs ...');
clear all;
```

VOS Parameters

Extrinsic & Intrinsic

```
VOSParameters;
```

IOS Parameters

```
user_reply = input('Please choose an IOS command  ');


while (isempty(user_reply))
    user_reply = input('invalid response...   Please try again:');
end


IMUspecs;
```

Run-Time (Simulation)

Length of simulation.

```
RunTime = input('Enter length of simulation (sec)');
%RunTime = 30; %s
```

Select VOS frequency

```
    clc;
    disp('Please select VOS frequency value in Hertz')
    vos_freq = input('Typically 20Hz: ');
    if vos_freq > ins_freq
        vos_freq = ins_freq;
    end
```

```
%total time between VOS measurements
    tmdelay = 1/vos_freq; %s
%h => approx integer number of IMU sampled steps between VOS measurements
%assumed constant - however, this may not be valid
    h = ceil(tmdelay/dt); %unitless (ceil - answer always rounded up)
```

initialize correlation time

```
corrtime = dt; %s
% number of discrete simulation steps.
% (floor - answer always rounded down).
t_steps = floor(RunTime/dt); % unitless
% mean tolerance for convergence
tol = 0.000001; %rad
```

Covariance Matrices

IOS Process Noise Covariance Matrix

```
Q = blkdiag(Qq, Qbias, Qsf, Qma);
```

```
% VOS Measurement Noise Covariance Matrix
Rp = blkdiag(Rs, Rim); %Overall VOS measurement noise covariance matrix
```

Initialize State Vector Covariance Matrix 'P_init'

Initially set to either Identity or 'Q' [15x15]

```
%P_init = eye(15); %aa state covariance Matrix

P_init = Q;

P_prev = blkdiag(P_init,Q); % 'augmented' state covariance matrix
```

State Vector Initialization (Estimate)

Initial 'Augmented' State Vector Estimate

```
% includes attitude quaternion

% [31 X 1]:

%

% x_error =

%  [ attitude quaternion      (4x1)

%    gyro biases              (3X1)

%    gyro scale factors       (3X1)

%    gyro misalignments       (6X1)]

% x_noises =

%  [ aa attitude noise        (3x1)

%    gyro bias noise          (3X1)

%    gyro scale factor noise  (3X1)

%    gyro misalignment noise  (6X1)]

q_guess = [1;0;0;0];

x_state = [ q_guess; 0;0;0; 0;0;0; 0;0;0;0;0;0];

x_noise = zeros(15,1); %Assume zero noise to start

x_prev =  [x_state; x_noise]; %Augmented State Vector
```

State Vector Initialization (Actual)

Initial 'augmented' state vector 'actual'

```
aa_init = Sigma_omega*randn(3, 1);

% initial quaternion attitude (offset from zero position slightly)

q_rot = aa2q(aa_init);

q_init = qmultiply(q_rot,q_guess);

bias_init = x_state(5:7) + Sigma_bias*randn(3,1); %rad/s

sf_init = x_state(8:10) + Sigma_sf*randn(3,1);

ma_init = x_state(11:16) +  Sigma_ma*randn(6,1); %0.2 percent FS

x_init = [q_init(1:4); bias_init(1:3); ...

  sf_init(1:3); ma_init(1:6); zeros(15,1)];
```

Memory Allocation

Initialize arrays (for speed) input (each axis)

```
u = zeros(3, t_steps); % Allocate space

% actual 'augmented' state (quaternion)

x_act = zeros(31, t_steps);   % Allocate space

% measured input (Euler ang vel from IMU unit)

u_m = zeros(3, t_steps); %Allocate space

%Fake VOS measurement (quaternion from VOS)

VOS_fake = zeros(4, t_steps);

%Available VOS measurement (quaternion from VOS)

VOS_avail = zeros(4, t_steps);

% measurement noise covariance

Rvos = zeros(3, 3, t_steps); %Allocate space

% estimated 'augmented' state (quaternion)

x_hat = zeros(31, t_steps); %Allocate space

% covariance of estimated state
```

```
P_hat = zeros(30, 30, t_steps); %Allocate space

% no VOS

x_noVOS = zeros(31, t_steps); %Allocate space

% Initial 'Previous' State [31x1]

x_noVprev = x_prev; %Estimate with no VOS to aid

% Initial 'Actual' State [31X1]

x_act(1:16,1) = x_init(1:16);


disp('Press any key to perform UKF compensation');

pause();

clc;

disp('Running simulation. Please wait...')
```

START OF SIMULATION & UKF Iteration

```
for i = 1:t_steps


% i = 1; % only used for testing purposes
```

Latency Compensation & Correlation Time (lag time)

Code to deal with latency issues between INS & VOS

```
% h(integer) approx number of simulation timesteps between VOS measurement

    if i >= 1 && i <= h

        l = rem(i+h-1, h)-h;    % l => latency term

                                % rem - remainder after division

        corrtime = dt + dt*(h + l);

    else

        l = rem(i-1, h);        %remainder after division
        corrtime = dt + dt*l;
    end
```

Note: Above code assumes tmdelay is constant It determines the current correlation time based on current lag term to be used in sensor error update (corrtime term) ie. corrtime = elapsed time(sec) since last VOS measurement Note: Matlab is able to handle division properly if corrtime = zero.

Dynamic Q

Not currently Implemented Dynamic update of IOS Process Noise Covariance Matrix 'Q' Attitude STD - Angular Random Walk Process

```
% Sigma_omega = ARW*sqrt(corrtime); %rad
% To be added after integration
% if l ~= 0 % Performed only if VOS Measurement not available
% Matrix is Dynamic (based on corrtime)
        % Qq = Sigma_omega^2*eye(3); %depends on corrtime
% Gyro bias error covariance - (Gauss Markov Process)
        % Qbias = Sigma_bias^2*(1-exp(1)^(-2*dt/corrtime))*eye(3);
% Gyro Scale Factor covariance -(Gauss Markov Process)
        %Qsf = Sigma_sf^2*(1-exp(1)^(-2*dt/corrtime))*eye(3);
% Gyro Axis Misalignment covariance - Random Constant
        % Qma = Sigma_ma^2*eye(6); %No change
%Combine to Make aa Process Noise Covariance (15X1)
        Q = blkdiag(Qq, Qbias, Qsf, Qma); %Process Noise Covariance Matrix
% Re-create new 'augmented' P_prev
        % P_prev = P_prev+blkdiag(zeros(15,15),Q);
        P_prev = blkdiag(P_prev(1:15,1:15),Q);
% end
```

Angular Rate Profile (Euler rates)

Angular rate inputs to follow about each Global axis of Atlas sphere.

```
if dt*i <= 10
u(:, i) = [0; %rad/s
           0; %rad/s
           0];%rad/s
elseif dt*i <= 40

u(:, i) = -1.0472*[0; %rad/s
           1/sqrt(2); %rad/s
           -1/sqrt(2)];%rad/s
elseif  dt*i <= 45
u(:, i) = [0; %rad/s
           0; %rad/s
           0];%rad/s
elseif dt*i <= 115
u(:, i) = -1.0053*cos(0.48*(dt*i))*[0; %rad/s
           1/sqrt(2); %rad/s
           -1/sqrt(2)];%rad/s
elseif dt*i <= 120
u(:, i) = [0; %rad/s
           0; %rad/s
           0];%rad/s
```

```
end
```

Actual 'Augmented' State (Run the model)

True attitude of Atlas sphere follows the IOS system process model perfectly random sensor errors are considered It is used to create the next ACTUAL Atlas attitude without noise and create the actual random gyro sensor errors

```
x_act(:,i) = IOSProcess(u(:,i),x_init, ...
dt,Sigma_omega,Sigma_bias,Sigma_sf,Sigma_ma,0);
x_init = x_act(:, i);
```

Create FAKE IMU Sample for Simulation

```
% create 'G' matrix from scale factors & misalignment factors
Gmatrix = v2G(x_act(8:10,i),x_act(11:16,i)); %unitless
% Measured Angular Velocity from IMU (Euler Angular Rates)

u_m(:,i) = u(:,i)....                      %actual angular velocity
      + x_act(5:7,i)...                     %gyro bias errors
      + (Gmatrix)*u(:,i); %... %...         %sf & ma errors
% Note: FAKE AWN is added after integration in simulation IOSprocess

% Extra unmodelled noise added if desired
  %+ 60*ARW*sqrt(ins_BW)*randn(3, 1);... %ARW - only if not using AWN
      %+ Sigma_res*randn(3,1)...            %res error
      %+ unmodelled*randn(3,1);             %unmodelled errors

% if user_reply == 4 - Not currently implemented
% -OR- Create a Fake IMU orientation measurement using actual noises
% sigmaact = q2aa(x_act(1:4,i)); %quaternion to axis angle
%
% sigmaIMU(:,i) = (eye(3)+Gmatrix)*sigmaact...
%      + x_act(5:7,i)*dt...
%      + 60*ARW*sqrt(ins_BW)*randn(3, 1)*dt... %ARW
%      + Sigma_res*randn(3,1)*dt...            %res error
%      + unmodelled*randn(3,1)*dt...           %unmodelled errors
%      + x_act(17:19,i); %rad                  %AWN error
% quatIMU(:,i)= aa2q(sigmaIMU(:,i));
% end
```

IMU State without VOS or UKF (Used For comparison only)

Obtains IMU state estimate based on current IMU measured state with no VOS, no UKF.

```
% ttime_noVOS = i*dt;
% Sigma_omega = ARW*sqrt(ttime_noVOS);
x_noVOS(:,i) = IOSProcess(u_m(:,i), x_noVprev, dt,0,0,0,0,1);
% Set the current state estimate as the next
% state for us in the i+1 iteration.
x_noVprev = x_noVOS(:, i); %noise not estimated
```

Create a Fake VOS Sample for Simulation

```
% Measure orientation with VOS process (add random noise)
   [VOS_fake(:, i)] = VOSprocess(x_act(1:4, i), CamCalInfo, N, r,1);
```

```
% Dynamic axis angle Rvos measurement covariance matrix
   makeRvos;


% Use lag term to mimic available lagged VOS measurements.
    VOS_avail(:,i) = VOS_fake(:, i-(l+h));
```

Run the UKF estimator

```
[x_hat(:,i), P_hat(:,:,i)] = ...
  UKF(u_m(:,i), VOS_avail(:,i), Rvos(:, :, i-(l+h)), ...
  x_prev, P_prev, dt,l,tol);


x_prev = x_hat(:,i);
P_prev = P_hat(:,:,i);

end
disp('done');
```

Visualize Quaternion Results

```
figure(1);


plot(dt*(1:t_steps), x_hat(1, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(1, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(1, :), 'm-')
plot(dt*(1:t_steps), VOS_avail(1, :), 'k-')
title('q1 - Scalar')
xlabel('Simulation Time Elapsed (sec)')
```

```
ylabel('Quaternion 1 Value (rad)')

figure(2);

plot(dt*(1:t_steps), x_hat(2, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(2, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(2, :), 'm-')
plot(dt*(1:t_steps), VOS_avail(2, :), 'k-')
title('q2')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Quaternion 2 Value (rad)')

figure(3);

plot(dt*(1:t_steps), x_hat(3, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(3, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(3, :), 'm-')
plot(dt*(1:t_steps), VOS_avail(3, :), 'k-')
title('q3')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Quaternion 3 Value (rad)')



figure(4);
```

```
plot(dt*(1:t_steps), x_hat(4, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(4, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(4, :), 'm-')
plot(dt*(1:t_steps), VOS_avail(4, :), 'k-')
title('q4')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Quaternion 4 Value (rad)')
```

Visualize Axis Angle Results

```
aa_err = zeros(3,t_steps);
aa_herr = zeros(3,t_steps);
aa_err_VOS = zeros(3,t_steps);

for k = 1:size(x_noVOS, 2)
    q_err = qerror(x_noVOS(1:4, k), x_act(1:4, k));
    aa_err(:, k) = q2aa(q_err);
end

for k = 1:size(x_noVOS, 2)
    q_err = qerror(VOS_avail(:,k), x_act(1:4, k));
    aa_err_VOS(:, k) = q2aa(q_err);
end

for k = 1:size(x_hat, 2)
    q_err = qerror(x_hat(1:4, k), x_act(1:4, k));
    aa_herr(:, k) = q2aa(q_err);
```

```
end

figure(5);
plot(dt*(1:t_steps), aa_err(1,:), 'r-');
hold on
plot(dt*(1:t_steps), aa_err_VOS(1,:), 'k-');
plot(dt*(1:t_steps), aa_herr(1,:), 'b-');
title('Axis Angle ERROR')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Axis Angle Value (rad)')

figure(6);
plot(dt*(1:t_steps), aa_err(2,:), 'r-');
hold on
plot(dt*(1:t_steps), aa_err_VOS(2,:), 'k-');
plot(dt*(1:t_steps), aa_herr(2,:), 'b-');
title('Axis Angle ERROR')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Axis Angle Value (rad)')

figure(7);
plot(dt*(1:t_steps), aa_err(3,:), 'r-');
hold on
plot(dt*(1:t_steps), aa_err_VOS(3,:), 'k-');
plot(dt*(1:t_steps), aa_herr(3,:), 'b-');
title('Axis Angle ERROR')
xlabel('Simulation Time Elapsed (sec)')
```

```
ylabel('Axis Angle Value (rad)')
```

Check Sensor Error Estimation

```
figure(8);


plot(dt*(1:t_steps), x_hat(5, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(5, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(5, :), 'm-')
title('Gyro Bias')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Sensor Bias Value (rad/s)')
hold off;


figure(9);


plot(dt*(1:t_steps), x_hat(8, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(8, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(8, :), 'm-')
title('Scale Factor')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Sensor SF Value (unitless)')
hold off;


figure(10);
```

```
plot(dt*(1:t_steps), x_hat(16, :), 'b-')
hold on
plot(dt*(1:t_steps), x_act(16, :), 'g-')
plot(dt*(1:t_steps), x_noVOS(16, :), 'm-')
title('Misalignment Factor')
xlabel('Simulation Time Elapsed (sec)')
ylabel('Sensor MA Value (unitless)')
hold off;

RMSplots;
```

Appendix D

Smaller Matlab functions

D.1 meanqRotation.m

Contents

This function converges to find the mean quaternion using the Riemannian manifold approach with an initial guess

```
function [q_mean, ei] = meanqRotation(qi, q0, tol)
```

Inputs:

- q0: initial mean guess (if no guess, use any qi)
- qi: array of all quaternions to be averaged
- tol: tolerance for convergence

Outputs:

- q_mean: mean quaternion converged upon
- ei: variance between each qi and q_mean

Obtain number of quaternions points to average

```
n = size(qi, 2); %number of columns
```

Initialize mean quaternion estimate

```
q_mean = q0;
```

Perform Iterations

```
% initialize e_mag, ei, counter and begin iteration
e_mag = tol+1;
ei_q = zeros(4, n);
ei = zeros(3, n);
count = 0;
while e_mag > tol && count < 100
    % get error quaternion rotations
    for k = 1:n
        ei_q(:, k) = qerror(qi(:, k), q_mean);
        % convert to  axis angle
        ei(:, k) = q2aa(ei_q(:, k));
    end
    % barycentric mean
    e_mean = 1/(n)*sum(ei, 2);
    % convert back to quaternion rotation
    e_meanq = aa2q(e_mean);
    % update mean by applying qrotation
    q_mean = qmultiply(e_meanq, q_mean);
    % obtain magnitude of e_mean
    e_mag = sqrt(e_mean'*e_mean);
    count = count+1;
end
```

D.2 aa2q.m

Converts a 3 component Euler axis angles phi, theta, psi (radians) into a corresponding rotation quaternion

```
function q = aa2q(aa)

t = sqrt(max(0, aa'*aa));

if t == 0
    q = [1; 0; 0; 0];
else
    q = [cos(t/2);
         aa./t.*sin(t/2)];
end

if q(1) < 0
    q = -q;
end
```

D.3 q2aa.m

Converts a 4 component rotation quaternion into corresponding Euler axis angles phi, theta, psi (radians)

```
function aa = q2aa(q)


t = 2*acos(q(1));
if t == 0 || (1-q(1)^2) < 0
    aa = zeros(3, 1);
else
    y = (1-q(1)^2)^(-0.5);


    aa = t*y*q(2:4, 1);
end
```

D.4 qmultiply.m

This function performs quaternion multiplication

```
function q = qmultiply(qa, qb)


q = [-qa(2)*qb(2) - qa(3)*qb(3) - qa(4)*qb(4) + qa(1)*qb(1);
      qa(2)*qb(1) + qa(3)*qb(4) - qa(4)*qb(3) + qa(1)*qb(2);
 -qa(2)*qb(4) + qa(3)*qb(1) + qa(4)*qb(2) + qa(1)*qb(3);
  qa(2)*qb(3) - qa(3)*qb(2) + qa(4)*qb(1) + qa(1)*qb(4)];
```

D.5 qerror.m

This function obtain the rotation quaternion between two quaternion orientations.

```
function qerr = qerror(qd, qm)

qerr = [ qd(2)*qm(2) + qd(3)*qm(3) + qd(4)*qm(4) + qd(1)*qm(1);
         qd(2)*qm(1) - qd(3)*qm(4) + qd(4)*qm(3) - qd(1)*qm(2);
         qd(2)*qm(4) + qd(3)*qm(1) - qd(4)*qm(2) - qd(1)*qm(3);
        -qd(2)*qm(3) + qd(3)*qm(2) + qd(4)*qm(1) - qd(1)*qm(4)];


if qerr(1) < 0
    qerr = -qerr;
end
```

This would be the same as performing these individual operations

- qa = qd;
- qb = qconj(qm);
- qerr = qmultiply(qa, qb);

D.6 R2q.m

This function create a orientation rotation matrix from an orientation quaternion.

```
function q = R2q(R)


T = max(0, 1+R(1, 1)+R(2, 2)+R(3, 3));
if T > eps
```

```
    s = 0.5/sqrt(T);
    q = [0.25/s;
        (R(3, 2)-R(2, 3))*s;
        (R(1, 3)-R(3, 1))*s;
        (R(2, 1)-R(1, 2))*s];

elseif R(1, 1) > R(2, 2) && R(1, 1) > R(3, 3)

    s = 2*sqrt(max(0, 1+R(1, 1)-R(2, 2)-R(3, 3)));
    q = [(R(3, 2)-R(2, 3))/s;
         0.25*s;
         (R(1, 2)+R(2, 1))/s;
         (R(1, 3)+R(3, 1))/s];


elseif R(2, 2) > R(3, 3)

    s = 2*sqrt(max(0, 1-R(1, 1)+R(2, 2)-R(3, 3)));
    q = [(R(1, 3)-R(3, 1))/s;
         (R(1, 2)+R(2, 1))/s;
         0.25*s;
         (R(2, 3)+R(3, 2))/s];

else

    s = 2*sqrt(max(0, 1-R(1, 1)-R(2, 2)+R(3, 3)));
    q = [(R(2, 1)-R(1, 2))/s;
```

```
         (R(1, 3)+R(3, 1))/s;
         (R(2, 3)+R(3, 2))/s;
         0.25*s];

end
```

D.7 q2R.m

This function creates the rotation matrix from an orientation quaternion

```
function R = q2R(q)


R = [1-2*q(3)^2-2*q(4)^2      2*q(2)*q(3)-2*q(4)*q(1) 2*q(2)*q(4)+2*q(3)*q(1);
     2*q(2)*q(3)+2*q(4)*q(1)  1-2*q(2)^2-2*q(4)^2     2*q(3)*q(4)-2*q(2)*q(1);
     2*q(2)*q(4)-2*q(3)*q(1)  2*q(3)*q(4)+2*q(2)*q(1) 1-2*q(2)^2-2*q(3)^2];
end
```

D.8 sign_mag.m function

created by Jesse Linseman This function will take in an array of any length and generate a magnitude with the associated sign for direction.

```
function valout = sign_mag(valin)


L = size(valin,1); %Number of values (rows)
temp = zeros(L,1);
sum = 0;
for i = 1:L
```

```
    temp(i)= sign_sqrd(valin(i));

    sum = sum + temp(i);

end

    valout = sign_sqrt(sum);

end
```

sign_sqrd.m

Internal function to retain the sign after squaring terms

```
function valout = sign_sqrd(valin)

    if valin < 0

        valout = -(valin*valin);

    else

        valout = valin*valin;

    end

end
```

sign_sqrt.m

Internal function to retain the sign after square rooting terms

```
function valout = sign_sqrt(valin)

    if valin < 0

        valout = -sqrt(abs(valin));

    else

        valout = sqrt(valin);

    end

end
```

Appendix E

VOS Process In Detail

Affixed to the outside of the sphere are 32 uniquely coloured markers specifically placed at known locations relative to the sphere's local coordinate frame. By this method, the digital camera is be able to identify each colour and associate each marker in the image with a known physical location relative to the sphere coordinate (local) frame as prescribed in an external file. By situating the camera at a precise distance away from the sphere, and using the 32 markers allows for at least three coloured markers to be observable in any camera image for any sphere orientation. From an acquired camera image, the pixel locations in the image for three markers can also be determined. Using the pixel marker locations in the image, the physical marker's coordinates relative to the world (global) frame can be determined based on the geometry of the sphere and camera model parameters (Section E.0.1). Having knowledge of the coordinate locations of three markers in both the world and sphere local frames allows the rotation matrix (orientation) to be computed which can then be made into an orientation quaternion.

Involved in the creation of the VOS measurement covariance matrix, $\mathbf{R}_{VOS}$, are extrinsic and intrinsic camera parameters determined through the camera calibration process.

Intrinsic: parameters correspond to the lens and image sensor (scaled focal lengths, principal point and possible distortion corrections).

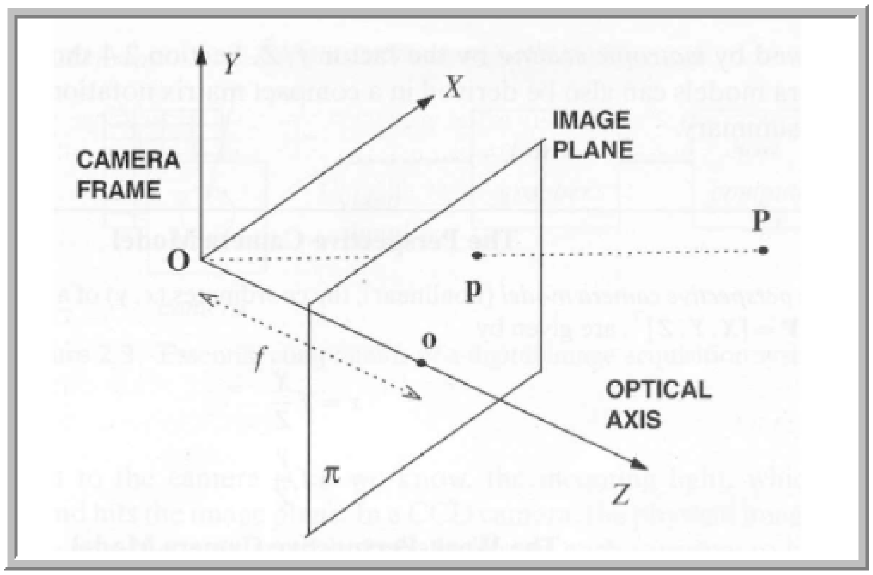
Extrinsic: parameters define the transformation from the world coordinate system to the camera coordinate system (rotation and translation).

The next subsections detail the underlying camera process model which uses geometric transformations to take three known pixel marker locations and determine the Atlas sphere absolute orientation. Note: The following details are also outlined in reports written by Kyle Chisholm ([28, 33])during the 2008/2009 school year (The reports can be accessed with permission through the CUSP SVN [19]). Portions of these texts are re-addressed here for further clarification of the camera process model used in the simulation of the UKF.

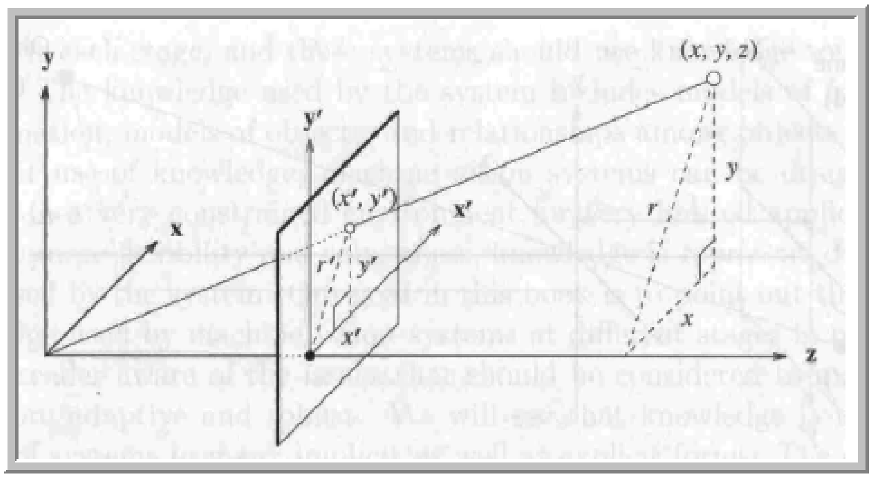
E.0.1 The VOS Perspective Projection Camera Model

The VOS perspective projection camera model, described in [28], [33], [36] and [37], is used to transform a marker location relative to the world frame (in 3-D space) to estimate a pixel location in the image (2-D plane). The model incorporates a 3D coordinate frame transformation from world to camera frame coordinates, perspective projection, and intrinsic parameters involving the camera optics and image sensor. Image distortion can also be corrected using a non-linear image distortion model but was not implemented at the time of this paper's writing.

Referring to Fig. E.1, with respect to the camera frame, a marker point ${}^{c}\mathbf{P}_{x,y,z}$ can be projected onto the image plane at point $\mathbf{p}_{x',y'}$, using similar triangle mathematics in Equation (E.1). For clarity, the image plane, π, is shown in front of the camera frame origin, $\mathbf{O}$, when in actuality it would be inverted behind the camera's focal point. The camera *focal length*, f, is the perpendicular distance from the focal point (camera frame origin, $\mathbf{O}$) along the camera frame optical Z-axis to the image plane origin, $\mathbf{o}$. Transforming marker point ${}^{c}\mathbf{P}_{x,y,z}$ is as follows.



(a) Frames of Reference



(b) Similar Triangles

Figure E.1: Perspective Projection [36]

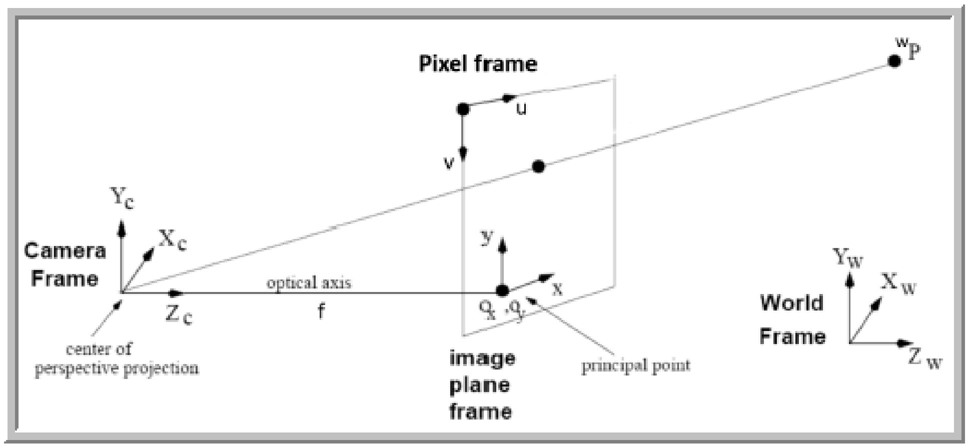


Figure E.2: Camera Projection Model [38]

$$\begin{aligned} p_{x'} &= f\frac{{}^{c}\mathbf{P}_x}{{}^{c}\mathbf{P}_z} \\ p_{y'} &= f\frac{{}^{c}\mathbf{P}_y}{{}^{c}\mathbf{P}_z} \end{aligned} \tag{E.1}$$

where f is the *focal length* of the camera and $\mathbf{p}_{x',y'}$ are the image coordinates of the projected marker point ${}^{c}\mathbf{P}_{x,y,z}$ which is a point relative to the camera frame, [36]. The 3D marker point relative to the camera frame, ${}^{c}\mathbf{P}_{x,y,z}$ is found using extrinsic camera parameters.

The extrinsic camera parameters (a rotation matrix, ${}^{c}\mathbf{R}_w$ and an origin translation, ${}^{c}\mathbf{T}_w$) are used to transform a 3D marker point, ${}^{w}\mathbf{P}_{x,y,z}$, relative to the world frame to the marker point, ${}^{c}\mathbf{P}_{x,y,z}$, relative to the camera frame, as per Equation (E.2).

$${}^{c}\mathbf{P}_{x,y,z} = {}^{c}\mathbf{R}_w \cdot {}^{w}\mathbf{P}_{x,y,z} + {}^{c}\mathbf{T}_w \tag{E.2}$$

Ignoring image distortion and referring to Figure E.2, the last step of the perspective camera model is the transformation of the image plane coordinates, $\mathbf{p}_{x',y'}$, to a pixel location, $\mathbf{p}_{u,v}$, in the pixel frame as per Equation (E.3).

$$\begin{aligned} p_u &= o_x + p_{x'}/s_x \\ p_v &= o_y + p_{y'}/s_y \end{aligned} \tag{E.3}$$

where the *Pixel frame* origin is the top left of the image and the origin of the *image frame* at point $\mathbf{o}$ is defined to be at pixel location (o_x, o_y), called the *principal point.* One camera pixel corresponds to a real world distance in the image plane. This is an intrinsic camera parameter given as a scaling factor, s_x, s_y, in unit length/pixel.

Substituting Equation (E.1) into Equation (E.3), Equation (E.4) is obtained which can be used to transform a 3D world marker point location,(${}^{w}\mathbf{P}_{x,y,z}$), into a corresponding 2D pixel frame point location, ($\mathbf{p}_{u,v}$).

$$\begin{aligned} p_u &= o_x + f_x \frac{{}^{c}P_x}{{}^{c}P_z} \\ p_v &= o_y + f_y \frac{{}^{c}P_y}{{}^{c}P_z} \end{aligned} \tag{E.4}$$

Where components ${}^{c}\mathbf{P}_{x,y,z}$ are found using Eqn. E.2. Also $f_x = \frac{f}{s_x}$ and, $f_y = \frac{f}{s_y}$ are the combined scaling and focal length intrinsic camera parameters, defined in Appendix B. Appendix B lists all the intrinsic and extrinsic camera parameters used in the current UKF simulation.

The perspective projection matrix, M

Simplifying the above, the entire VOS Perspective Projection Camera Model, as depicted in Figure E.2 is encompassed in the following perspective projection matrix, $\mathbf{M}$, re-defined below from [33].

$$\mathbf{M} = \begin{bmatrix} f_x r_{11} + o_x r_{31} & f_x r_{12} + o_x r_{32} & f_x r_{13} + o_x r_{33} & f_x T_x + o_x T_z \\ f_y r_{21} + o_y r_{31} & f_y r_{22} + o_y r_{32} & f_y r_{23} + o_y r_{33} & f_y T_y + o_y T_z \\ r_{31} & r_{32} & r_{33} & T_z \end{bmatrix} \qquad \text{(E.5)}$$

$$\text{Where} \quad {}^{c}\mathbf{R}_w = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Using $\mathbf{M}$, a 2D pixel frame point location, $\mathbf{p}_{u,v}$ can be found from a 3D marker point, ${}^{w}\mathbf{P}_{x,y,z}$, which is relative to the world frame, as follows (E.6).

$$\begin{Bmatrix} x_h \\ y_h \\ h \end{Bmatrix} = \mathbf{M} \begin{bmatrix} {}^{w}P_x \\ {}^{w}P_y \\ {}^{w}P_z \\ 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{p}_{u,v} = \left\{ \tfrac{x_h}{h}, \tfrac{y_h}{h} \right\} \qquad \text{(E.6)}$$

E.0.2 VOS Process

Using Carleton University's coordinate measurement machine (faro CMM), 32 uniquely coloured markers have been placed at precise known locations on the outside of the AtlasLite sphere. Observing markers in an image, the camera software can distinguish their colours from which to lookup their corresponding local sphere frame coordinates, ${}^{L}\mathbf{P}_{x,y,z}$. All 32 local sphere coordinate locations are recorded in a Matlab array file that was created during their CMM placement, named '$SpherePts\_r4.75.mat$'

. Using Matlab, Figure E.3 is a 3D plot that displays the current 32 marker locations surrounding the sphere's centre (local sphere frame origin). These points are what is currently being used to test the UKF since at the time of this work, the VOS was non-operational due to colour identification issues.

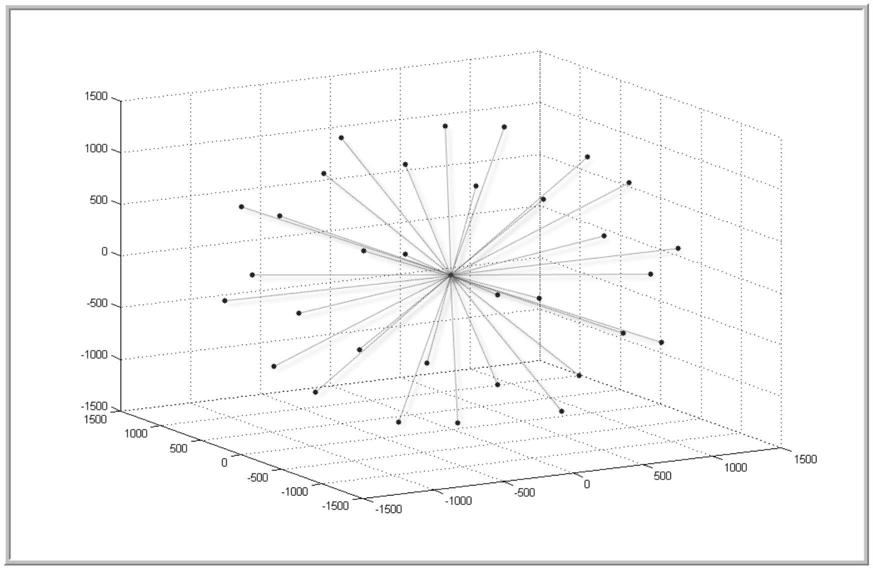


Figure E.3: 32 Local Sphere Marker Positions (mm), ${}^{L}\mathbf{P}_{x,y,z}$

During simulation and testing of the UKF, each simulated VOS orientation measurement is created using a 'true' known orientation quaternion that the UKF attempts to estimate. The simulated VOS orientation quaternion is created as follows.

Creating a simulated VOS image for simulation

1. The current rotation matrix, ($\mathbf{R}_{3\times3}$), is created from the 'true' orientation quaternion using Eqn. 2.12 derived in section 2.4, and all 32 local sphere marker points are rotated into their 'true' world frame marker coordinates as follows.

$${}^{w}\mathbf{P}_{3\times32} = \mathbf{R}_{3\times3}{}^{L}\mathbf{P}_{3\times32}$$

2. Next, ${}^{w}\mathbf{P}_{3\times32}$ gets transformed into the camera frame using Eqn. E.2. 32

absolute distances between the camera frame origin and each camera frame marker point are determined.

$$d_{abs} = \sqrt{({}^{c}P_x)^2 + ({}^{c}P_y)^2 + ({}^{c}P_z)^2}(mm)$$

3. Taking the N shortest d_{abs} distances, N corresponding sphere markers are determined to be in the camera's field of view (F.O.V). (the number of markers assumed in this case is $N = 3$)

4. Using the perspective projection matrix, $\mathbf{M}$, outlined in section E.0.1, the $N = 3$ world marker points are transformed into $N = 3$ corresponding 2D pixel frame point locations using Eqn. E.6.

The above steps complete the process of creating a simulated VOS image with $N = 3$ known pixel locations for simulation. Both pixel image distortion and sphere marker location errors are added in as random noise for the purposes of simulation in the above steps as well. In essence, the camera's image processing time corresponds to this above procedure. In the UKF, before the VOS process model can be used, this above procedure is repeated each iteration to create a map of the $N = 3$ 2D pixel positions,$\mathbf{p}_{u,v}$, observed in the camera's fov (the expected pixel image).

Determining an orientation quaternion from a pixel image, VOS process model

In order to determine an orientation quaternion from an image, the $N = 3$ 2D pixel points must be transformed back to obtain the corresponding world coordinates as well as local coordinates found from the associated colours. Achieving this allows the corresponding rotation matrix to be developed and transformed further into an orientation quaternion. A challenge remains that the perspective projection matrix is not invertible; it can only find a 2D pixel point from a 3D world marker point. However, since the marker locations are fixed to the AtlasLite sphere with radius r_s, and the center of the sphere is at the world frame origin, a new constraint can be added to obtain the inverted perspective model [33]. Shown in Equation (E.7), the radial constraint is as follows:

$$r_s^2 = \sqrt{({}^wP_x)^2 + ({}^wP_y)^2 + ({}^wP_z)^2} \tag{E.7}$$

As per Chisholm [28], [33], Eqn.E.4 and Eqn.E.7 were solved for ${}^c\mathbf{P}^w = {}^c\mathbf{R}_w{}^w\mathbf{P}$ in terms of r_s, ${}^w\mathbf{T}_c$ (Translation relative to camera coordinates), and $\mathbf{p}_{x',y'}$ (image frame coordinates), which, for each 2D pixel location point, yields two 3D points on the sphere for each marker as depicted in Fig. E.4.

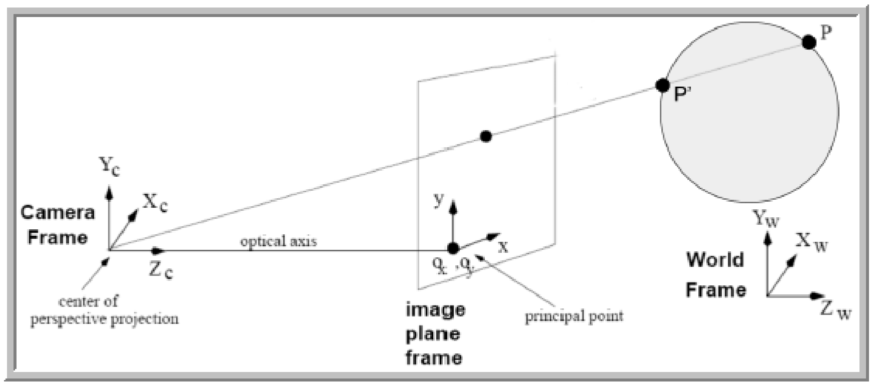


Figure E.4: Inverse projection intersecting AtlasLite sphere at two points [33]

Taking the 3D solution closest to the camera frame ($\mathbf{P}'$), we get Equation (E.8), which transforms a 2D pixel point into an equivalent 3D marker point with respect to the camera frame.

$$\begin{aligned}
{}^{c}P^{w}_{x} &= \frac{-x'}{(x'^2+y'^2+1)}\Big(-x'T_x - y'T_y + x'^2T_z + y'^2T_z + \xi\Big) + x'T_z - T_x \\
{}^{c}P^{w}_{y} &= \frac{-y'}{(x'^2+y'^2+1)}\Big(-x'T_x - y'T_y + x'^2T_z + y'^2T_z + \xi\Big) + y'T_z - T_y \\
{}^{c}P^{w}_{z} &= -\sqrt{r_s^2 - ({}^{c}P^{w}_{x})^2 - ({}^{c}P^{w}_{y})^2} \qquad \text{(E.8)}
\end{aligned}$$

$$\text{with} \quad \xi = \sqrt{r_s^2(x'^2+y'^2+1) - (T_x - x'T_z)^2 - (T_y - y'T_z)^2 - (y'T_x - x'T_y)^2}$$

The following steps outline the procedure for creating the VOS orientation quaternion measurement from $N = 3$ observed 2D pixel frame point locations in an image, also recognizing their associated colours.

1. Get the marker locations relative to the Local sphere frame, ${}^{L}\mathbf{P}_{x,y,z}$, from the associated $N = 3$ marker colours.

2. Determine the $N = 3$ absolute 3D camera frame marker positions, $({}^{c}\mathbf{P}^{w}_{x,y,z})$, from the pixel locations using Equation (E.8).

3. Rotate the camera frame marker positions into the World frame using the camera's transposed extrinsic rotation matrix, $({}^{c}\mathbf{R}^{T}_{w})$, to obtain the $N = 3$ measured/estimated marker positions, $({}^{w}\mathbf{P}_{x,y,z})$, as follows.

$${}^{w}\mathbf{P}_{x,y,z} = {}^{c}\mathbf{R}^{T}_{w}\,{}^{c}\mathbf{P}^{w}_{x,y,z}$$

4. Now that the marker locations relative to both sphere and world frame are known, the rotation matrix which defines the orientation of the sphere can be determined. The rotation matrix ${}^{w}\mathbf{R}_{L}$ is the orientation of the Local sphere frame relative to the World frame, and can be calculated from the markers by

Equation (E.9), where subscripts 1, 2, and 3 refer to each of $N = 3$ 3D markers in the image [33].

$$\begin{aligned}\begin{bmatrix} {}^{w}\mathbf{P}_1 & {}^{w}\mathbf{P}_2 & {}^{w}\mathbf{P}_3 \end{bmatrix} &= {}^{w}\mathbf{R}_L \begin{bmatrix} {}^{L}\mathbf{P}_1 & {}^{L}\mathbf{P}_2 & {}^{L}\mathbf{P}_3 \end{bmatrix} \\ {}^{w}\mathbf{R}_L &= \begin{bmatrix} {}^{w}\mathbf{P}_1 & {}^{w}\mathbf{P}_2 & {}^{w}\mathbf{P}_3 \end{bmatrix} \begin{bmatrix} {}^{L}\mathbf{P}_1 & {}^{L}\mathbf{P}_2 & {}^{L}\mathbf{P}_3 \end{bmatrix}^{-1} \end{aligned} \tag{E.9}$$

5. To get an orthogonal rotation matrix, singular value decomposition is performed. The rotation matrix obtained from the previous step is not orthogonal due to errors present in the calibration process. In order to make the rotation matrix orthogonal, such that $\mathbf{R}^T\mathbf{R} = \mathbf{I}$, singular value decomposition can be applied. Given the measured rotation matrix ${}^{w}\mathbf{R}_L = \mathbf{SDV}^T$, the new orthogonal rotation matrix is calculated to be $\mathbf{R}_{new} = \mathbf{SV}^T$ (matrix D is constrained as an identity matrix) [37].

6. This new rotation matrix, $\mathbf{R}_{new}$ is converted into an orientation quaternion, as per Equation (2.13) in Section 2.5. Note: If $N > 3$ markers are analyzed, a least squares solution may be applied in order to calculate the rotation matrix but has not been implemented.

The entire VOS process model described to this point is proposed as a non-linear observation function for the UKF. A Matlab function, namely '*VOSprocess.m*', has been created for this process and is provided in Appendix E.0.2. For creating a simulated VOS measurement, random noise is added during simulation for two quantities; the measured CMM local sphere marker locations, and the measured pixel locations in the acquired image. For simulation in the VOS process function, a flag input value is used solely to signal when to introduce this added random noise.

It has been observed through simulation and testing with real IMU data that the entire VOS process function can be avoided within the UKF altogether for speed savings. This results due to the fact that when a VOS measurement becomes available it is lagged. The more current IOS measurement is available to act as the expected VOS measurement for the lagged VOS measurement. By doing so, it has been observed that only a slight degradation results with a large computational speed savings. The method works for VOS operational frequencies higher than 15 Hz, otherwise, the current IOS measurement can not be trusted due to significant drift. This application is further described in Section 3.4.