### Smooth Position and Velocity Sub-Algorithm

A Kalman smoothing filter is used to smooth the ECI position and velocity vectors to accomplish step 1.e. above. The Kalman filter assumes a random process that can be modeled as follows:



1. where:

[*X*]K = (n x 1) state vector at time tK

[]K = (n x n) matrix relating XK to XK+1

[*q*]K = (n x 1) process noise at time tK

[*f* ]k = (n x 1) forcing function at time tK

Measurements of the process are modeled as follows:



1. where:

[*Z*]K = (m x 1) measurement vector at time tk

[*H*]K = (m x n) relates the state vector at time tk to the measurement

[*v*]K = (m x 1) measurement error at time tk

As noted below, in this application, the measurements are direct observations of the state vector, so n = m.

To smooth the ephemeris data, the state vector X is defined as follows:



1. where:

Xp, Yp, Zp = X, Y, Z position

Xv, Yv, Zv = X, Y, Z velocity

The measurement vector [Z] is a 6x1 vector containing the telemetry X, Y, Z positional values along with the telemetry X, Y, Z velocity values. The measurement vector looks like the state vector but contains the measured ephemeris telemetry values for time tk, whereas the state vector contains our estimate of the “true” ephemeris position and velocity values.

The discrete state transition matrix is defined as follows:



where t is the time transition between measurement k and k+1.

The matrix [H] is defined as a 6x6 identity matrix since the measurements directly correspond to the elements of the state vector.

The forcing function, [ *f* ], is equal to the change in acceleration of the satellite due to the Earth’s gravitational potential. The Potential Functions sub-algorithm below describes the forcing function in more detail.

The process noise vector [q]K represents a random forcing function that models the uncertainty in the dynamic model as a zero-mean random process with covariance [Q]. The process noise controls how strictly the filtered states will conform to the dynamic model.

The process noise covariance matrix is defined as follows:



where:

σx=standard deviation in X positional element/value

σy=standard deviation in Y positional element/value

σz=standard deviation in Z positional element/value

σxv=standard deviation in X velocity element/value

σyv=standard deviation in Y velocity element/value

σzv=standard deviation in Z velocity element/value

The measurement error matrix is 6x6 diagonal matrix:



where:

mp = variance of error in positional measurement

mv = variance of error in velocity measurement

The Kalman filter is used to produce a set of filtered and predicted state vectors, along with estimated and predicted covariance state error matrices. These values are then used to produce a smoothed state vector. This smoothed state vector will represent the smoothed position and velocity ephemeris data.

Prediction equations:





Filter equations:







where:

[*I*] is the identity matrix

[*P*] is the error covariance matrix

[*Q*] = E[qtqt]

[R] = E[vtvt]

[*X*]PK = estimate of [*X*] given measurements through tK

[*P*]PK = error covariance associated with estimate [*X*]K

[*X*]K = filtered estimate at tK

[*P*]K = filtered estimate at tK

Note that the filtering step is skipped for points flagged as outliers so that:

[*X*]K = [*X*]PK

[*P*]K = [*P*]PK

Using the definitions above, a new notation can be written:

[*X*]K|K-1 = [*X*]PK

[*P*]K|K-1 = [*P*]PK

[*X*]K|K = [*X*]K

[*P*]K|K = [*P*]K

The smoothing equations are then:

For n=number of points-1,…,0







The Kalman filter is initialized with a state vector:



1. where:

Px(0) = first available X positional value

Py(0) = first available Y positional value

Pz(0) = first available Z positional value

Vx(0) = first available X velocity value

Vy(0) = first available Y velocity value

Vz(0) = first available Z velocity value

The initial error covariance matrix is defined as follows:



1. where:

*σx* = initial standard deviation in X position

*σy* = initial standard deviation in Y position

*σz* = initial standard deviation in Z position

*σxv* = initial standard deviation in X velocity

*σyv* = initial standard deviation in Y velocity

*σzv* = initial standard deviation in Z velocity

* Initialize the state vector, error covariance matrix, and measurement error matrix
* Loop on ephemeris points
	+ Calculate Δt (time difference between sample time i+1 and i)
	+ Calculate process noise matrix
	+ Calculate Kalman gain
	+ Filter state vector and error covariance matrix
	+ Predict error covariance error matrix
	+ Calculate force (acceleration) of Earth’s mass
	+ Predict state
* Initialize Δt to nominal delta time (1 sec)
* Loop on ephemeris (reverse order for smoothing)
	+ Calculate smoothed gain
	+ Calculate smoothed state
	+ Calculate Δt (time difference between sample time i+1 and i)

The resulting [*X*]K|N are the smoothed ephemeris state vectors.

##### Gravitational Potential Functions

This sub-algorithm calculates the gravitational potential of the Earth, represented as acceleration (x,y,z). One way to model the Earth’s gravitational potential is by:



1. where:

*Jn* = Spherical Harmonics determined by experimentation

μ = Earth’s gravitational parameter

re = equatorial radius of Earth

Pn = Legendre Polynomials

L = geocentric latitude

sin(L) = z/r

Taking the partial derivatives of the potential function with respect to x, y, and z gives the forcing functions needed for each axis.



 

 



 

 



 

 

 

The heritage implementation uses the following six functions to invoke the potential calculations:

p1x - first derivative of X (velocity)

p1y - first derivative of Y (velocity)

p1z - first derivative of Z (velocity)

p2x - second derivative of X (acceleration)

p2y - second derivative of Y (acceleration)

p2z - second derivative of Z (acceleration)

These functions are used to populate the six elements of the forcing function used to Kalman smooth the ephemeris data.

##### Attitude Data Preprocessing

This sub-algorithm validates the quaternion attitude estimates and converts their spacecraft time codes to accomplish steps 2.b., 2.c., and 2.d. above.

1. Extract the attitude quaternion data records from the ancillary data.
2. Search the ancillary data attitude records and find the first and last valid attitude records in the interval. Extract the time tags for these records.
3. Adjust the attitude data window as necessary to ensure that it fits entirely within the ephemeris data window.
4. Set the start and stop indexes for the attitude to be stored in the model to the indexes found in c).
5. Set the attitude epoch to the time associated with the start index found in step c) converted to UTC (see Convert Spacecraft Time Code to UTC sub-algorithm above). Retain the corresponding epoch spacecraft time, as it will be subtracted from the other attitude samples.
6. Loop on attitude records starting at and ending at indexes found in c).
7. Set attitude sample time to the spacecraft time code minus the attitude start time, i.e., convert times to offsets from attitude epoch.
8. Compute the magnitude of the attitude quaternion:

Mag = sqrt( q1\*q1 + q2\*q2 + q3\*q3 + q4\*q4 )

1. Check the magnitude against the nominal value of 1:
	1. If the magnitude is between 1- and 1+ then store the value for processing. The quaternion normalization tolerance value, , is nominally 1e-06 (1 part per million) and stored in the CPF.
	2. If the magnitude is outside the allowable range, then flag the value as an outlier.

If SIRU processing is required, complete the following steps:

1. Extract the IMU (SIRU) data records from the ancillary data.
2. Process the SIRU clock data to construct spacecraft time codes for each SIRU sample. This is step 2.e. above and is described in the “Process SIRU Time” sub-algorithm below. If this step fails, all subsequent SIRU processing is suppressed by setting a “SIRU\_Valid” flag to FALSE.
3. Examine the SIRU status words, flagging any invalid points as outliers.
4. Convert the SIRU counts to angular rates. This is step 2.f. above and is described in the “Process SIRU Counts” sub-algorithm below.
5. Repeat steps b) through e) above for the SIRU data.
6. Use the SIRU epoch as the combined attitude data time epoch. If the SIRU data are not used, the attitude quaternion times are used instead.

##### Process SIRU Time Sub-Algorithm

This sub-algorithm analyzes the SIRU clock readouts accompanying each SIRU data sample and uses these in conjunction with the SIRU clock sync offset values and SIRU sync spacecraft time codes included in the Level 0R IMU data records to construct spacecraft time codes for each SIRU data sample.

SIRU sample timing is driven by a clock internal to the SIRU unit. This SIRU clock is a 16-bit counter that increments every 4 microseconds, and rolls over when the 16-bit counter reaches its 64K limit. The SIRU clock is periodically (every 10 seconds or so) synchronized with the spacecraft clock when flight software sends a reset command. Flight software records the spacecraft time code associated with this reset, and the SIRU records the offset between the clock counter value at the time of the reset and the clock counter value at the time of the current data sample in its clock sync field. These offsets are scaled to units of 1/3 of a microsecond (1/12 of the SIRU clock resolution). The clock counter value is not changed by the reset operation, so successfully locating and processing a single reset event is sufficient to establish the timing relationship between the spacecraft and SIRU clocks. The spacecraft time code associated with the reset is captured in the ancillary data, as is the SIRU sync field. Several additional considerations regarding the SIRU timing data include the following:

1. The SIRU sync field is only populated during the 100 Hz cycle while it is being updated (i.e., while the SIRU is being resynchronized). Otherwise, this field will contain fill. A fill value of -32768 is used for this purpose.
2. One implication of the 100 Hz SIRU cycle is that some sync values will go unrecorded by the 50 Hz L8/9 IMU telemetry. The syncs will be timed such that alternate values will be sampled by and therefore present in the ancillary data. This raises the question of how these unrecorded sync values will be detected and recovered. The SIRU sync spacecraft time code value that accompanies each IMU record will change for the record containing the sync. That would make it possible to use the SIRU sample clock data to recover the missing sync values, though it is probably not important to do so. The sync events that are represented in the data should be sufficient to establish the SIRU/spacecraft timing relationship.
3. Ancillary test data acquired during spacecraft comprehensive performance tests demonstrated that the SIRU clock is not perfectly synchronized to the spacecraft clock. This was manifested as timing jitter in both the SIRU and attitude estimates (which apparently take their times from the corresponding SIRU samples) in which adjacent samples, nominally separated by 0.02 seconds, are sometimes 0.01 and sometimes 0.03 seconds apart, indicating timing drift across the 100 Hz sampling sequence. This has the following effects:
	1. The SIRU rates must be computed using the actual time differences rather than the nominal time difference.
	2. The assignment of times to samples prior to the first SIRU clock sync must use the actual SIRU clock values, not the nominal timing offset.
	3. The SIRU clock syncs are not always visible every 20 seconds (1000 samples apart). The separation is sometimes 500 samples and sometimes 1500 samples due to 100 Hz sampling cycle slippage. This (partly) motivated the inclusion of logic to validate SIRU clock sync events against the previously established timing, to ensure that timing gaps are not introduced into the SIRU data. It also means that intervals shorter than 30 seconds may not contain any valid SIRU clock sync events, leading to the failure of this algorithm and the suppression of further SIRU processing for the interval. This is unlikely in normal (Earth-view, lunar, and stellar) acquisitions, but likely in solar calibration data.
4. The scaling and use of the SIRU latency estimate telemetry in the spacecraft ancillary data stream is not entirely clear. Ancillary data sets from the spacecraft Comprehensive Performance Test (CPT) indicate that the latency is the time offset, in seconds, between the SIRU (and 50 Hz quaternion) data and the flight software 1 Hz cycle times (i.e., the times at which ephemeris and attitude filter outputs are generated). Since this offset is reflected in the time codes that accompany these data elements, the latency estimates are somewhat redundant. The baseline algorithm does not apply a latency correction.

Three items of SIRU timing telemetry will be used in the following algorithm: the SIRU clock value at the sample time (one per SIRU sample), the SIRU sync reference field (one per sample time, but only valid during resynchronization cycles), and the SIRU sync spacecraft time code (one per IMU record). These will be referred to as clock value (clock), clock sync (sync), and sync time code (time), respectively.

For each IMU record:

Compute the spacecraft time of last sync from the time code seconds and microseconds: time = seconds + microseconds/1e6

For each SIRU sample (i):

If the clock sync field is not fill:

* 1. Record the current sample clock and time (above) values as base\_clock and base\_time. Set base\_sync equal to base\_clock.
	2. Compute the SIRU sync offset from the SIRU sync word using the 1/3 microsecond per count scaling factor. The sync word scaling is represented as a ratio relative to the SIRU clock scaling (12 sync counts per clock count):

sync\_time = SIRU sync word \* siru\_time\_scale / SIRU sync ratio

* 1. Add the SIRU sync offset to the base\_time. This the time (in seconds from spacecraft epoch) corresponding to the base\_clock SIRU clock value.
	2. Initialize the current offset, 16-bit rollover counter and previous offset value:

siru\_offset = 0

excess\_offset = 0

last\_offset = siru\_offset

* 1. Compute the time of the current sample:

gtime[i] = base\_time + siru\_offset \* siru\_time\_scale (from CPF)

* 1. If this is the first valid (non-fill) value, calculate all previous times by working back through the previous SIRU clock samples, subtracting each from the previous time:

for ( j = i to 1 )

 clock = MOD(siru\_clock[j]–siru\_clock[j-1]+64K, 64K)

 gtime[j-1] = gtime[j] – clock\*siru\_time\_scale

* 1. Make sure the new clock sync is consistent with previous time codes:

If abs(gtime[i]-gtime[i-1]-0.02) > 0.02

 gtime[i] = gtime[i-1] + 0.02

 base\_time = gtime[i] – siru\_offset\*siru\_time\_scale

This coarse test ensures that the sync update does not introduce a timing adjustment of more than a full 0.02 second sample time. A warning message is generated if this adjustment is made. This test ensures that the SIRU time codes are consistent, and at a minimum, are based on the first sync time in the interval.

Otherwise:

If no valid sync fields have been found, go to the next point.

Otherwise:

Note: All arithmetic involving clock and sync variables is modulo 64K.

Check for a sync that was not sampled:

* + 1. If time > base\_time + 0.02 (a resync occurred) AND

 time < gtime[i-1] + 0.02 (it occurred before this sample):

* + - 1. Reconstruct the sync time:

sync\_time = gtime[i-1] + 0.02 - time

* + - 1. Update the base\_time:

base\_time = time + sync\_time

* + - 1. Reset the other sync cycle variables:

base\_clock = clock

base\_sync = clock

siru\_offset = 0

excess\_offset = 0

* + 1. Otherwise:
			1. Calculate the sample time offset based on the current sync variables:

siru\_offset = clock – base\_sync (modulo 64K)

* + - 1. Correct for previous 16-bit rollover:

siru\_offset += excess\_offset

* + - 1. See if 16-bit rollover occurred on this sample and increment rollover and offset variables; if so:

if (siru\_offset < last\_offset)

excess\_offset += 0x010000

siru\_offset += 0x010000

* + 1. Compute the time of the current sample:

gtime[i] = base\_time + siru\_offset \* siru\_time\_scale

* + 1. Set the last offset value to the current value (used to detect missed sync resets):

last\_offset = siru\_offset

If no valid sync values were detected, return an error.

This procedure performs step 2.e. above.

##### Process SIRU Counts Sub-Algorithm

This sub-algorithm converts the raw SIRU data counts to angular rates.

For each SIRU sample:

1. If the sample’s SIRU validity flags are not set:
	1. Mark the point as an outlier.
	2. If this is the first point, set the angular rates to zero.
	3. Otherwise, set the angular rates to the previous sample values.
2. For valid SIRU samples:
	1. If this is not the first point, compute the difference between the current integrated angle reading and the previous reading for each of the 4 SIRU axes.
	2. If this is the first point, compute the difference between the next integrated angle reading and the current reading for each of the 4 SIRU axes. If the next point is invalid, mark the current point as an outlier and set the angular rates to zero.
	3. Check for SIRU reset/rollover on each axis:
		1. If the value of the angle difference is > 32K, subtract 64K.
		2. If the value of the angle difference is < -32K, add 64K.
	4. Scale the counts to radians using the SIRU scale factor from the CPF.
	5. The SIRU delta angle measurements are converted to rates by dividing by the delta time computed from the SIRU sample time codes. This could also be done after the four SIRU axis measurements are converted to roll-pitch-yaw measurements using the method described in the next section.

This procedure performs step 2.f. above. For lunar and stellar intervals, all SIRU samples are flagged as outliers so they will be deweighted by the attitude Kalman filter.

##### Rotate SIRU Sub-Algorithm

This sub-algorithm rotates the SIRU data to the ACS coordinate frame to accomplish step 2.g. above. Note that this sub-algorithm will only be used if SIRU data processing is required.

Construct the roll, pitch, and yaw rotational matrices from SIRU measurements and rotate angles to the ACS coordinate system. Note that rather than reporting roll, pitch, and yaw rotations directly, the SIRU reports rotations about four non-orthogonal axes oriented in an octahedral tetrad. These four correlated measurements must first be reduced to rotations about the three orthogonal X-Y-Z axes using the SIRU axis vectors from the CPF. These vectors define the orientations of the four SIRU axes, and are nominally:

 SIRU A: [ +0.57735027 +0.57735027 +0.57735027 ]

 SIRU B: [ +0.57735027 -0.57735027 +0.57735027 ]

 SIRU C: [ -0.57735027 -0.57735027 +0.57735027 ]

 SIRU D: [ -0.57735027 +0.57735027 +0.57735027 ]

Any one of these SIRU axes can be lost and the rotations about the X-Y-Z axes can still be recovered. The vector for a failed SIRU axis should be set to zero.

Convert SIRU angles to roll-pitch-yaw:

Construct the [S] matrix where the columns of this 3 by 4 matrix contain the four SIRU axis vectors:



Construct the 3 by 4 SIRU to roll-pitch-yaw conversion matrix:



Construct the 4 by 1 SIRU observation vector:

 where, θn = angle for SIRU axis n

Convert the four SIRU observations to three roll-pitch-yaw angles in the SIRU coordinate system:



Construct the perturbation matrix using the SIRU roll-pitch-yaw angles:



Where [IRU to ACS] is the SIRU to attitude matrix found in the CPF and [ACS to IRU] is its inverse.

The SIRU measured roll, pitch, and yaw in ACS coordinates are then:







##### Correct for Orbital Motion in SIRU Data Sub-Algorithm

The spacecraft SIRU senses rotations relative to inertial space, so it will measure the orbital pitch used to maintain spacecraft pointing, as well as any deviations from that nominal alignment with the orbital coordinate system. In using the SIRU data to densify or repair the spacecraft attitude estimates, they are blended in the orbital coordinate system. Before this can be done, it is necessary to correct the SIRU data for the time-varying orientation of the orbital frame relative to inertial space. This is step 2.h. above.

To account for off-nadir viewing and yaw steering effects, a reference attitude vector representing the mean roll-pitch-yaw for the scene is also provided. This is necessary because the orbital pitch effect will show up in more than just the spacecraft pitch axis if the spacecraft body is not aligned with the orbital coordinate system. The reference attitude is calculated as the average of the roll-pitch-yaw values derived from the quaternion data for the attitude interval.

Note that this sub-algorithm will only be used if SIRU data processing is required.

If the acquisition type is Earth-viewing, then:

Loop on all SIRU values:

1. Calculate the ECI position and velocity of satellite at time t0 using Lagrange interpolation. The l8\_movesat unit does this. That sub-algorithm is included in the LOS Model Creation algorithm.
2. Calculate the transformation matrix from the satellite orbit system to the spacecraft body/ACS coordinate system (ORB2ACS) using the input reference mean attitude. This is the transpose of the ACS2ORB matrix shown in the “Convert Roll, Pitch, Yaw to Quaternion” section below.
3. Calculate the transformation matrix from ECI to satellite orbit system for time t0 and tn (the inverse of the ORB2ECI matrix presented in the next section).

Using the satellite position and velocity at times t0 and tn, the following matrix transformations can be calculated:





Calculate the transformation from the Orbit system to ECI for time tn using the ephemeris state vector at time tn.

1. Use the ORB2ACS matrix to compute the ECI2ACS matrices from the ECI2ORB matrices:

[eci2acs]t0 = [orb2acs] [eci2orb]t0

[eci2acs]tn = [orb2acs] [eci2orb]tn

Since the eci2acs matrix is orthogonal, acs2eci can be calculated as:



5. Calculate the amount of roll, pitch, and yaw due to the satellite’s orbit.

The roll, pitch, and yaw due to the orbital motion of the satellite can be found by looking at the matrix transformation from spacecraft frame reference at time tn to spacecraft frame reference t0.









time = tn – t0

This formulation computes the orbital attitude rate correction and assumes that the SIRU data are, or have been converted to, rates.

1. Remove orbital motion attitude delta from original values.



The sign is swapped to convert the SIRU angles/rates from body-to-orbit to orbit-to-body.

##### Convert to Spacecraft Roll, Pitch, and Yaw Sub-Algorithm

The attitude data are given as quaternions in the ECI reference frame (ECI2ACS). The quaternions are converted to roll, pitch, and yaw values in the ACS reference frame per step 2.i. above.

We first take the conjugate of the incoming ECI2ACS quaternion (**q**) to calculate the corresponding ACS2ECI quaternion (**q’**).

 q’1 = -q1

 q’2 = -q2

 q’3 = -q3

 q’4 = q4

The direction cosines (transformation) matrix from the ACS reference axis to the ECI reference system (ACS2ECI) is constructed from the ACS2ECI quaternion, **q’**, as:

ACS2ECI =



The ACS2ECI transformation matrix can also be defined as the product of the inverse of the spacecraft's attitude perturbation matrix **P** and the transformation matrix from the orbital reference system to the ECI reference system (ORB2ECI)

The relationship between the orbital and ECI coordinate systems is based on the spacecraft's instantaneous ECI position and velocity vectors. The rotation matrix to convert from orbital to ECI can be constructed by forming the orbital coordinate system axes in ECI coordinates:



where:

*p* = spacecraft position vector in ECI

*v* = spacecraft velocity vector in ECI

*n* = nadir vector direction

*h* = negative of angular momentum vector direction

*cv* = circular velocity vector direction

[ORB2ECI] = rotation matrix from orbital to ECI

The transformation from orbital to ECI coordinates is the inverse of the ECI to orbital transformation matrix. Since the ECI to orbital matrix is orthogonal, the inverse is also equal to the transpose of the matrix.



ACS2ECI = **[**ORB2ECI**][P -1]**

The orbital reference system to ECI matrix must be defined at the same time as the

spacecraft's attitude matrix.

The roll, pitch, and yaw values are contained in the **P**-1 matrix; thus:

**P**-1 = **[**ORB2ECI**] -1[**ACS2ECI**]**

The spacecraft attitude is then:



For lunar and stellar intervals, the rotation to the orbital coordinate system is not performed, so the resulting roll, pitch, and yaw values are relative to the ECI system. An additional check is performed on these roll, pitch, yaw values to make sure that there are no crossings of the +/- radians boundary, with 2 being added or subtracted as necessary to keep the attitude sequence continuous. This check is performed on all intervals, but is only necessary for lunar/stellar data.

##### Smooth Euler and SIRU Sub-Algorithm

A Kalman smoothing filter is used to combine the attitude and SIRU data into one data stream and/or replace attitude estimates flagged as outliers per step 2.j. above. Note that this sub-algorithm will only be used if SIRU data processing is required and if the SIRU data are not suppressed.

Lagrange interpolation is used to synchronize the SIRU and quaternion information at the SIRU sampling interval relative to the attitude epoch time. This is necessary because the quaternion and SIRU data sample times are not necessarily uniformly spaced in the original spacecraft ancillary data. The formulation shown here assumes that the SIRU is reporting attitude rate data rather than integrated angles. Due to the increased potential for noise in rate measurements, an additional step is required to synchronize the SIRU data. Specifically, the SIRU rate measurements are integrated to form angles, the angles are Lagrange interpolated to synchronize the times, then the interpolated angles are converted back to rates. The rate to angle integration is performed as follows (the roll, pitch, and yaw axes are each processed separately using this method):

 SIRU\_angle[0] = SIRU\_rate[0]\*nominal\_SIRU\_time

 For k = 1 to NSIRU-1

SIRU\_angle[k] = SIRU\_angle[k-1]

 + SIRU\_rate[k]\*(SIRU\_time[k] – SIRU\_time[k-1])

Performing the time regularizing interpolation in angle space suppresses any rate noise present in the SIRU data. The interpolated angles are turned back into rates, suitable for use in the Kalman smoother, as follows:

 For k = NSIRU-1 to 1

 SIRU\_rate[k] = (SIRU\_angle[k] – SIRU\_angle[k-1])/nominal\_SIRU\_time

 SIRU\_rate[0] = SIRU\_angle[0] / nominal\_SIRU\_time

The state vector is defined as follows:



where:

attitude = smoothed attitude state

iru = attitude rate state associated with SIRU

drift = slow linear drift error in IRU

The measurement matrix [Z] is a 2x1 matrix containing the Euler and SIRU attitude data for time tk.



1. where:

epa = Euler attitude value at time tk

iru = SIRU attitude value at time tk

The state transition matrix is defined as follows:



where:

dt = sample timing of SIRU

The matrix [H] is defined as follows:



The process noise covariance matrix is defined as follows:



where:

σiru=standard deviation of SIRU process

σdrift=standard deviation of drift process

The measurement noise covariance matrix is defined as a 2x2 diagonal matrix:



1. where:

*meuler* = observation standard deviation noise in Euler measurement

*miru* = observation standard deviation noise in SIRU measurement

Samples flagged as outliers are deweighted by multiplying the measurement standard deviation by 100 for that point.

Each axis is treated as an independent data stream. The Kalman filter is used to produce a set of filtered and predicted state vectors, along with estimated and predicted covariance state error matrices. These values are then used to produce a smoothed state vector. The smoothed vector attitude will represent an overall satellite attitude, or a combination of the Euler and SIRU measurements.

The Kalman filter has an initial state vector of:



where:

epa(0) = first measured quaternion

iru(0) = first measured SIRU

The initial covariance error matrix is defined as the following:



1. where:

σepa = initial standard deviation in Euler

σiru = initial standard deviation in SIRU

σdrift = initial standard deviation in drift

Initialize the state vector, error covariance matrix, measurement error matrix, and dt.

Loop on attitude points

* Calculate process noise matrix
* Calculate Kalman gain
* Filter state vector and error covariance matrix
* Predict error covariance error matrix
* Predict state

Loop on attitude points (reverse order for smoothing)

* Calculate smoothed gain
* Calculate smoothed state

The Kalman filtering machinery used here is the same as described in the Smooth Position and Velocity Sub-Algorithm above.

If SIRU data processing is not performed, this sub-algorithm is replaced by a simple attitude outlier replacement algorithm that replaces estimates flagged as outliers above, by linearly interpolating new roll, pitch, and yaw values from the neighboring samples.

##### Convert Roll, Pitch, Yaw to Quaternion

The roll pitch and yaw sequences computed above are converted to ECI quaternions and to ECEF quaternions per steps 2.k. and 2.l. above. The conversion algorithm is the same in both cases, the only difference being whether the algorithm is provided with ECI ephemeris data or ECEF ephemeris data. See note 7.

Complete the following steps for each attitude sample:

1. Compute the net roll-pitch-yaw by adding the bias value.
2. Use Lagrange interpolation to compute the ephemeris position and velocity at the time of the roll, pitch, yaw attitude sample.
3. Compute the rotation matrix corresponding to the roll-pitch-yaw values:

[ACS2ORB] =



1. Construct the rotation matrix to convert from orbital to ECI/ECEF by forming the orbital coordinate system axes in ECI/ECEF coordinates:



where:

*p* = spacecraft position vector in ECI/ECEF

*v* = spacecraft velocity vector in ECI/ECEF

*n* = nadir vector direction

*h* = negative of angular momentum vector direction

*cv* = circular velocity vector direction

[ORB2EC] = rotation matrix from orbital to ECI/ECEF

1. Compute the ACS2EC rotation matrix:

 **[**ACS2EC] = **[**ORB2EC**][**ACS2ORB**]**

1. Construct the corresponding EC2ACS quaternion:

First, noting that the ACS2EC matrix computed above can be expressed in terms of the corresponding quaternion components as the following:

ACS2EC =



We can derive the following set of equations to compute the quaternion components from the elements of ACS2EC:

1. Compute the four quantities:

d1 = 1 + ACS2EC11 – ACS2EC22 – ACS2EC33

d2 = 1 – ACS2EC11 + ACS2EC22 – ACS2EC33

d3 = 1 – ACS2EC11 – ACS2EC22 + ACS2EC33

d4 = 1 + ACS2EC11 + ACS2EC22 + ACS2EC33

1. Find the largest of these four quantities and use the corresponding equations to compute the quaternion:









This method avoids the numerical danger of dividing by a small number.

We then take the conjugate of the resulting ACS2EC quaternion, **q**, to yield the output EC2ACS quaternion, **q’**:

 q’1 = -q1

 q’2 = -q2

 q’3 = -q3

 q’4 = q4

#### Maturity

The ancillary data preprocessing algorithm includes new features (e.g., SIRU processing) but reuses many heritage components (e.g., coordinate system transformations). Some notable modifications to the heritage logic include the following:

1. The inertial to Earth-fixed coordinate transformation logic was upgraded to include leap seconds (table in CPF to convert from TAI) and light travel time effects (for LOS projection).
2. The Landsat and EO-1 heritage algorithm for converting between Earth-fixed and inertial coordinates performs simple rotation from the ECI to the ECEF system (or vice versa) for any input vector. This has the effect of rotating the inertial velocity vector to the ECEF frame without incorporating the Earth rotation effect in the velocity. The GPS-derived L8/9 ECEF ephemeris includes Earth rotation effects. As noted in the ADD above, a new velocity conversion unit was required to implement the velocity equations shown in figure A.1 of DMA TR8350.2-A:

Position: **X**ECEF = [ ABCD ] **X**ECI

Velocity: **V**ECEF = [ AB’CD ] **X**ECI + [ ABCD ] **V**ECI

Where: B’ is the time derivative of the B matrix.

1. The heritage ephemeris time jitter correction and Kalman smoother logic has been included in the baseline algorithm, but may not be necessary, as the spacecraft ephemeris should be cleaner than what we got from EO-1.
2. The heritage IAU 1980 precession and nutation models were replaced with the NOVAS C3.1 implementation of the IAU 2000 models.

#### Notes

Algorithm assumptions and notes, including those embedded in the text above, are as follows:

1. The attitude and position/velocity estimates produced by the spacecraft will be sufficiently accurate to achieve L8/9 geolocation accuracy requirements without definitive processing of the raw attitude sensor and/or GPS data.
2. Ancillary data for the full imaging interval (with 4 seconds of extra data before and after the interval) is available to provide the required geometric support data, a CPF containing the scale factors needed to convert the ancillary data to engineering units is available, and the quality thresholds needed to detect and remove or repair outliers are provided in the CPF.
3. The spacecraft ancillary data will provide attitude estimates (in the form of ECI-to-body quaternions) at the same rate that it provides SIRU data. It remains to be seen whether the spacecraft attitude estimates embody the full SIRU bandwidth. If they are overly smoothed, then the SIRU data will be used to restore the high-frequency information to the sequence of attitude estimates.
4. Spacecraft time codes will be TAI offsets from the J2000 epoch. Since TAI and UTC differ only by leap seconds, the conversion to UTC amounts to a leap second correction. The spacecraft (J2000) epoch is hard coded (in a #define statement) to prevent it from being inadvertently changed.
5. Spacecraft ephemeris data will be provided in ECEF rather than ECI coordinates.
6. Ancillary data will include ephemeris and attitude records that contain time tags/time codes (e.g., seconds and fractions of seconds) that are TAI offsets from the J2000 epoch. Note that J2000 occurred at January 1, 2000, 11:59:27.816 TAI, which corresponds to January 1, 2000, 11:58:55.816 UTC (ref. LDCM Space to Ground Interface Control Document 70-P58230P Rev B). These times reflect the 32.184-second offset between TAI and TDT (the J2000 epoch reference frame) and the 32-second offset between TAI and UTC as of J2000. The TAI-UTC offset at J2000 includes the fixed 10-second TAI-UTC offset as of January 1, 1972, and the 22 accumulated leap seconds between then and J2000.
7. The baseline algorithm retains the heritage roll-pitch-yaw attitude model. At some point in the future, this may be replaced by a reformulated model that uses the quaternion representation directly. The sub-algorithm that converts the roll-pitch-yaw attitude representation to a quaternion may not ultimately be used in such a quaternion-based reformulation of the attitude model, since a part of that reformulation would probably involve directly filtering/smoothing the quaternion sequence rather than working in roll-pitch-yaw coordinates. That said, having the capability, provided by this sub-algorithm, to generate a quaternion attitude data representation that is identical to the roll-pitch-yaw representation would simplify the testing of any future reformulation. Note that by including both roll-pitch-yaw and quaternion representations of the attitude data, the algorithm outputs support either approach.
8. Common mathematical algorithms (e.g., matrix and vector operations, Lagrange interpolation) that can be found in standard references (e.g., Numerical Recipes in C) are cited without being described here.
9. The spacecraft estimate of SIRU latency was a late addition to the spacecraft ancillary data stream. Based on our current understanding of its meaning, it is not needed for SIRU data processing and is not included in the SIRU processing algorithm as of this writing. Should the need for this parameter be established, it should be a straightforward adjustment to the computed SIRU sample times.
10. The terms “IMU,” “IRU,” and “SIRU” are used interchangeably in this document. Inertial Measurement Unit (IMU) is another name for an Inertial Reference Unit (IRU).