Tightly coupling guide line

2021/04/09

郭　暁亮

0. Foreword

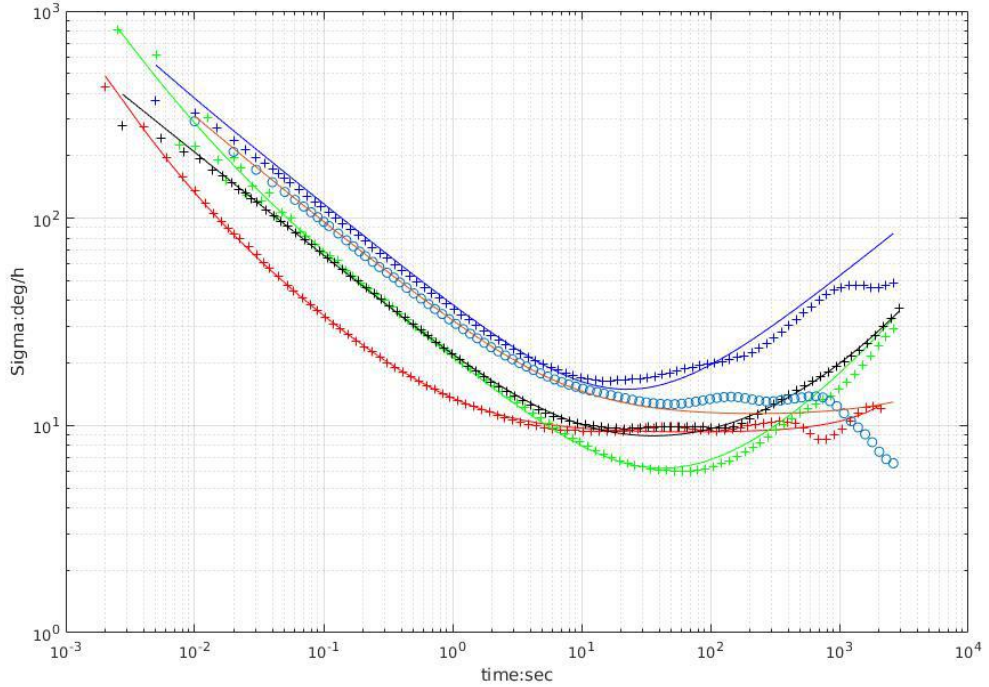
This PPT mainly talk about:

- 1. The open source TC code in the GitHub;
- 2. Calibration of internal parameters;
- 3. Initial alignment;
- 4. Mechanization;
- 5. GNSS clock error and clock rate;
- 6. Kalman filter in TC;
- 7. Usage of TC program.

1. The open source TC code

URL	Language	Environment	Note
https://github.com/marcoamm/gnssins	C 95%, Fortran 4.8%	Linux	On going work on a tightly coupled multi-gnss integration based on Groves (2013)
https://github.com/Erensu/ignav	C 64.9%, C++ 34.5%	Linux	Include LC, TC, Odometer assist, Magnetometer assisted, motion constraints, Doppler assisted INS / GNSS, visual information aided positioning, etc.
https://github.com/benzenemo/TightlyCoupledINSGNSS	MATLAB 100%	Windows	A tightly coupled based on Groves (2013)

2. Calibration of internal parameters



Result of CMG-100

Noise	X	Y	Z	Unit
Quantization	0.151	0.135	0.158	Q/arcsec
Random Walk	0.138	0.133	0.146	$N/(°)/h^{1/2}$
Bias Instability	1.096	1.323	1.089	B/(°)/h
Rate Random Walk	21.253	20.077	22.199	$K/(°)/h/h^{1/2}$
Rate Ramp	15.027	14.351	15.718	$R/(°)/h^2$

*** URL: https://github.com/gaowenliang/imu_utils**

2. Calibration of internal parameters

There are three kinds of errors of IMU:

1. Bias:

Constant value offset;

2. Scale factor error:

A scale factor, multiplied the A/D conversion value multiplied;

3. Installation error:

gyro nonorthogonality, and The coordinate axes do not coincide

*** URL: https://github.com/Kyle-ak/imu_tk**

2. Calibration of internal parameters

2.1 Bias

Error explanation:

Constant deviation in the output of gyroscope or accelerometer, commonly known as bias.

Error characteristics:

Due to the instability of the bias, the bias is not fixed.

Solution:

In practice, it can only be approximately constant for a period of time.

Acc bias: $b_a = [b_{ax} \quad b_{ay} \quad b_{az}]$

Gyro bias: $b_g = [b_{gx} \quad b_{gy} \quad b_{gz}]$

2. Calibration of internal parameters

2.2 Scale factor error

Error interpretation:

A scale factor, multiplied the A/D conversion value multiplied.

Error characteristic:

It is not necessarily a constant value, it will change with the input size, which is the nonlinearity of scale factor.

Solution:

If the degree of nonlinearity is relatively large, the nonlinear curve needs to be compensated to be linear before calibration.

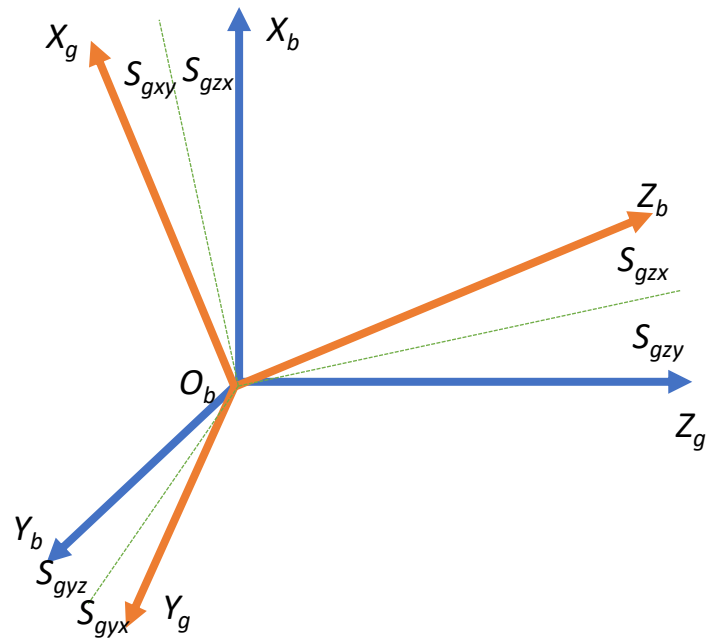
Acc Scale factor: $K_a = \begin{bmatrix} K_{ax} & & \\ & K_{ay} & \\ & & K_{az} \end{bmatrix}$ **Gyro Scale factor:** $K_g = \begin{bmatrix} K_{gx} & & \\ & K_{gy} & \\ & & K_{gz} \end{bmatrix}$

2. Calibration of internal parameters

2.3 Installation error

Error explanation:

gyro nonorthogonality, and The coordinate axes do not coincide.



Error characteristics:

Due to the instability of the bias, the bias is not fixed.

Solution:

In practice, it can only be approximately constant for a period of time.

Acc installation error : $S_a = \begin{bmatrix} 0 & S_{axy} & S_{axz} \\ S_{ayx} & 0 & S_{ayz} \\ S_{azx} & S_{azy} & 0 \end{bmatrix}$ **Gyro bias:** $S_g = \begin{bmatrix} 0 & S_{gxy} & S_{gxz} \\ S_{gyx} & 0 & S_{gyz} \\ S_{gzx} & S_{gzy} & 0 \end{bmatrix}$

2. Calibration of internal parameters

Error model:

$$W = K_g(I + S_g)\omega + b_g \approx (K_g + S_g)\omega + b_g$$

Acc error:

$$\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \begin{bmatrix} K_{ax} & S_{axy} & S_{axz} \\ S_{ayx} & K_{ay} & S_{ayz} \\ S_{azx} & S_{azy} & K_{az} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \begin{bmatrix} b_{ax} \\ b_{ay} \\ b_{gz} \end{bmatrix}$$

Gyro error:

$$\begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix} = \begin{bmatrix} K_{gx} & S_{gxy} & S_{gxz} \\ S_{gyx} & K_{gy} & S_{gyz} \\ S_{gzx} & S_{gzy} & K_{gz} \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} b_{gx} \\ b_{gy} \\ b_{gz} \end{bmatrix}$$

*** The local g vector is easy to know, so we can solve the Acc error!**

2. Calibration of internal parameters

• G vector by MATLAB:

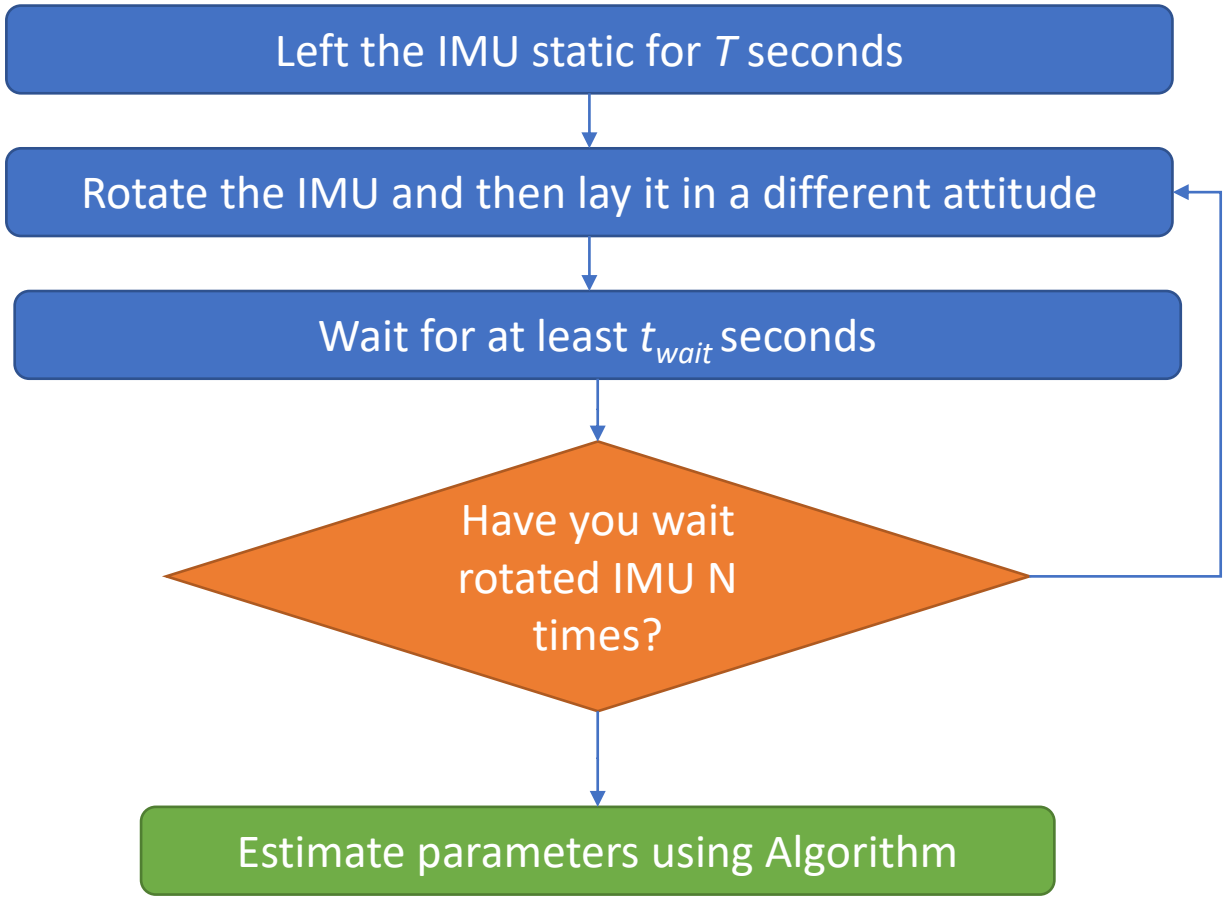
```
function [gVector] = normalGravity(lat,height)
RE_WGS84  =  6378137.0;          %  earth semimajor axis (WGS84) (m)
FE_WGS84  =  (1.0/298.257223563); %  earth flattening (WGS84) */
BE_WGS84    =RE_WGS84*(1.0-FE_WGS84);   %/* earth semiminor axis (WGS84) (m) */
GAMMA_A=       9.7803267715;       %/* gravitational acceleration at semimajor axis (m/s^2) */
GAMMA_B=       9.8321863685;       %/* gravitational acceleration at semiminor axis (m/s^2) */
OMGE=       7.2921151467E-5;     %/* earth angular velocity (IS-GPS) (rad/s) */
GM  =       3.986005E14 ;       %/* earth gravitational constant WGS84 (m^3/s^2) */

% Calculate normal gravity at ellipsoidal surface according to Somigliana
m = ((GAMMA_A*GAMMA_A) / GM)*OMGE*OMGE*BE_WGS84;
dGamma = (RE_WGS84*GAMMA_A*power(cos(lat), 2.0) + BE_WGS84 * GAMMA_B*power(sin(lat), 2.0)) /sqrt(power(RE_WGS84 * cos(lat), 2.0) + power(BE_WGS84 * sin(lat), 2.0));
% Propagate gravity to current height and set g vector
gVector(1) = -8.08E-9 * height * sin(2.0 * lat);
gVector(2) = 0.0;
gVector(3) = dGamma * (1.0 - (2.0 / RE_WGS84) * (1.0 + FE_WGS84 + m - 2.0*FE_WGS84*power(sin(lat), 2.0))*height + 3.0 * power(height/ RE_WGS84, 2.0));
end
```

* G vector in campus is 9.7978 m/s*s

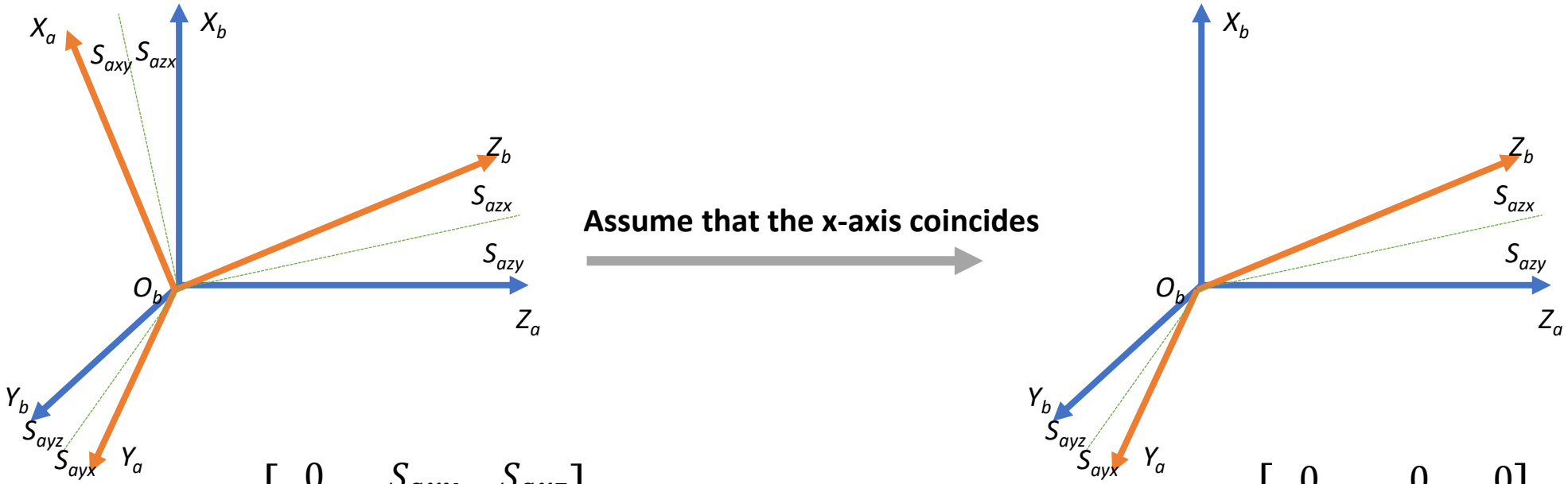
2. Calibration of internal parameters

Calibration without turntable



2. Calibration of internal parameters

Error model:



$$S_a = \begin{bmatrix} 0 & S_{axy} & S_{axz} \\ S_{ayx} & 0 & S_{ayz} \\ S_{azx} & S_{azy} & 0 \end{bmatrix} \rightarrow S_a = \begin{bmatrix} 0 & 0 & 0 \\ S_{ayx} & 0 & 0 \\ S_{azx} & S_{azy} & 0 \end{bmatrix}$$

- **Reference:**

A Robust and Easy to Implement Method for IMU Calibration without External Equipments

2. Calibration of internal parameters

Error model:

$$\boldsymbol{\theta}^{acc} = [S_{ayx}, S_{azx}, S_{azy}, K_{ax}, K_{ay}, K_{az}, b_{gx}, b_{gy}, b_{gz}]$$

$$A = K_a(I + S_a)a + b_a$$

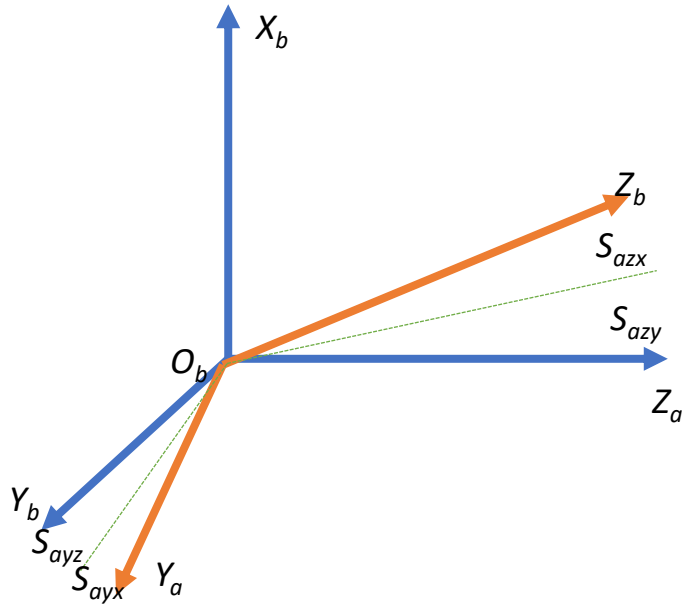
$$a = (I + S_a)^{-1}K_a^{-1}(A - b_a)$$

g vector is:

g = [0, 0, g]

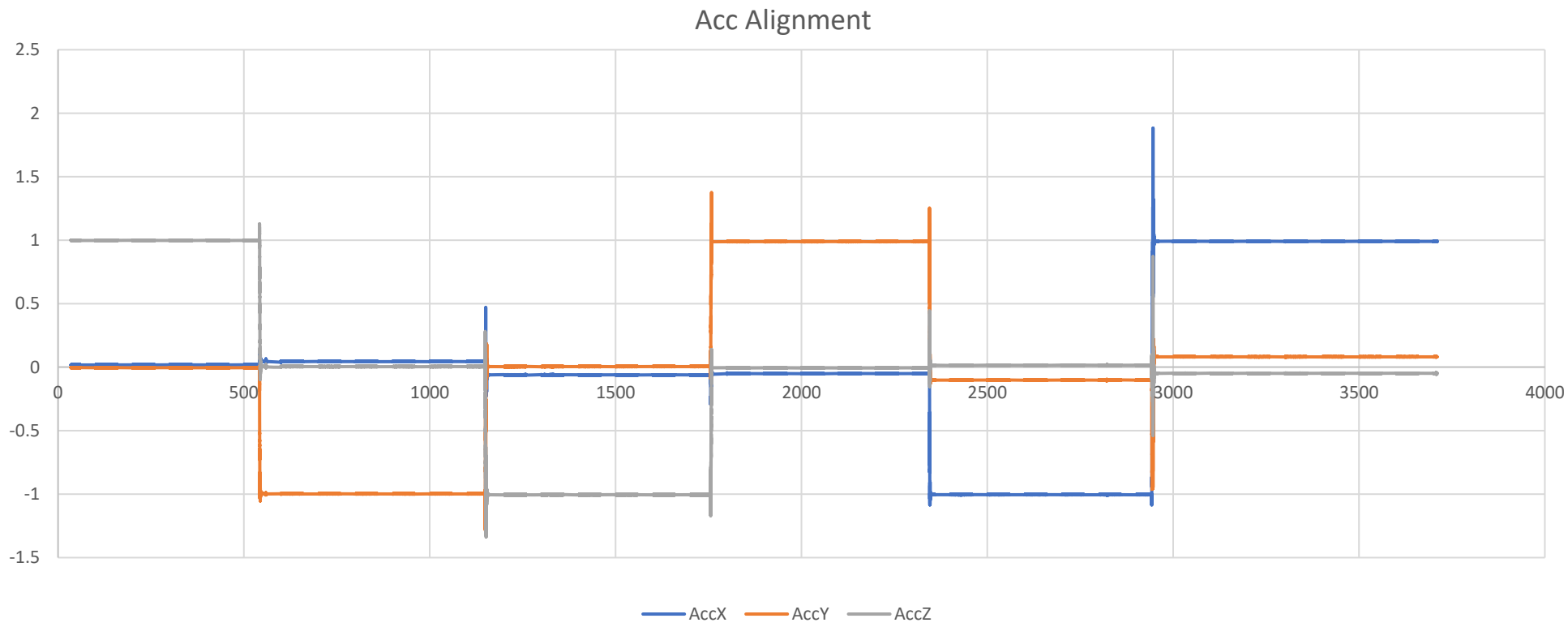
When the internal parameters is correct:

||g||^2 = ||a||^2



2. Calibration of internal parameters

residual function is ||g||^2 - ||a||^2



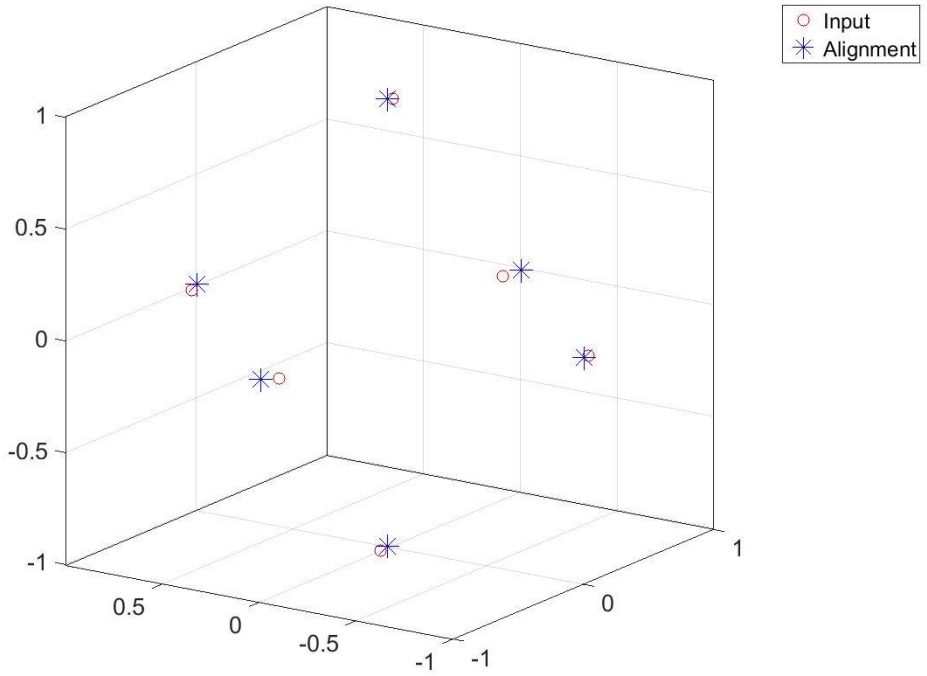
2. Calibration of internal parameters

2.1 Acc error of K

	x		y		z	
1	Back	0.018153	Right	-0.00443	Down	0.998284
2	Back	0.043775	Down	-0.99834	Left	0.004385
3	Back	-0.06068	Left	0.004725	Up	-1.00496
4	Back	-0.05057	Up	0.988496	Right	-0.0064
5	Down	-1.00449	Back	-0.10214	Right	0.015054
6	Up	0.990415	Front	0.081077	Right	-0.04803

2. Calibration of internal parameters

2.1 Acc error of K



URL: https://github.com/yandld/nav_matlab

2. Calibration of internal parameters

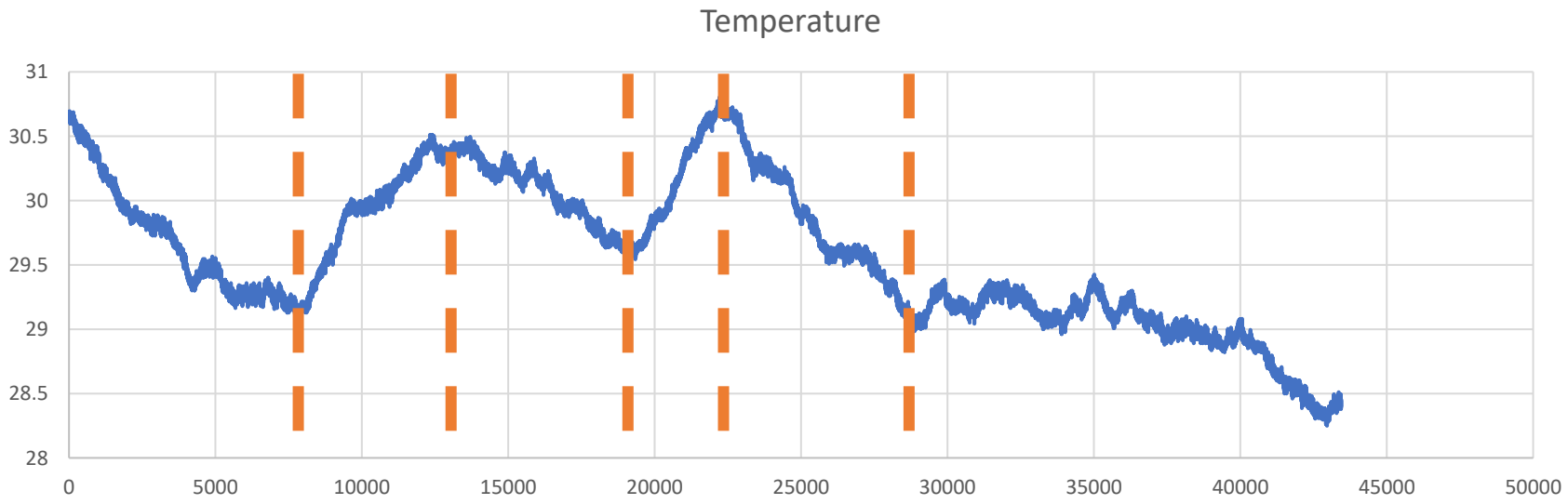
Temperature compensation:

Find the relation between bias and temperature, temperature change rate.

$$B = a * T^3 + b * T^2 + c * T + e * \Delta T^2 + f * \Delta T + g * T * \Delta T + h$$

However, the temperature is not regular, it is better to divide the data to several parts,

And calculate the model. //TODO



3. Initial alignment

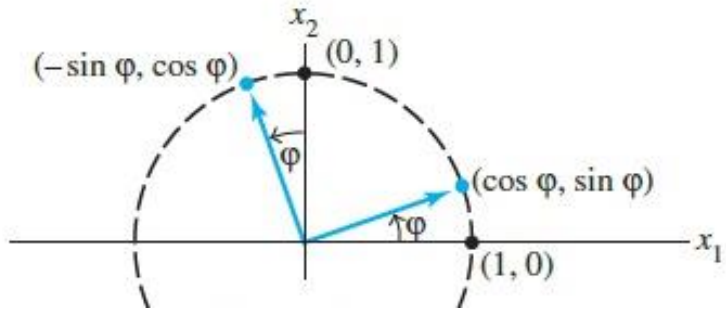
Before introduce alignment,

we need to know the transform matrix.

As for the 2D example:

X (1,0) is the start point, φ is the rotation angle, Y is the result.

- $R = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$



Y = R * X.

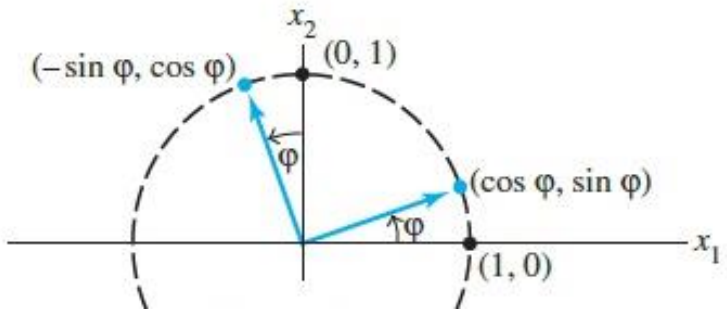
If we call the old coordination is b, new coordination is I,

We usually name the function as C_b_i or C_b^i.

3. Initial alignment

From 2D to 3D, we need to expand the R, from 2 by 2 to 3 by 3, C_b_n.

- $R = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$



1. Rotation γ around Z axis

$$C_b^i = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Rotation β around Y axis

$$C_b^i = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

3. Rotation α around X axis

$$C_b^i == \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

3. Initial alignment

When we describe the 3D rotation,

We can multiply the three rotation matrixes.

So, x, y and z, which axis rotates first?

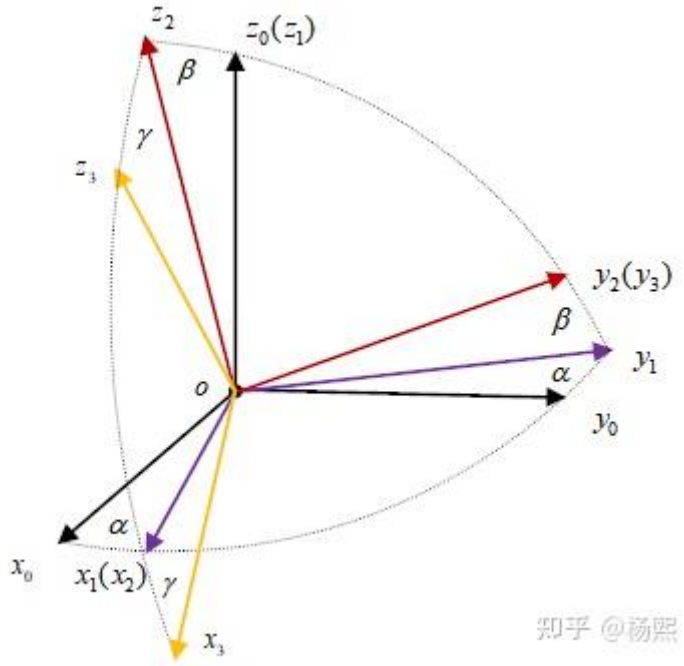
If we rotate the z for α, then rotate the x for β,

last rotate the y for γ, the fig is like this.

Usually call it (+3) - (+1) - (+2).

“+” means right hand coordinate;

“-” means left hand coordinate.



Reference:

Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors

3. Initial alignment

Due to the Gimbal lock, the Euler angle in navigation only have two kinds of ways, (+3) - (+2) - (+1) and (+3) - (+1) - (+2).

So, x, y and z, which axis rotates first?

If we rotate the z for 30 deg, then rotate the y for 40 deg, last rotate the z for 50 deg, the result is like this.

Cb2n_321 =

0.4924	-0.4568	0.7408
0.5868	0.8029	0.1050
-0.6428	0.3830	0.6634

Cb2n_312 =

0.2462	-0.6634	0.7066
0.7934	0.5567	0.2462
-0.5567	0.5000	0.6634

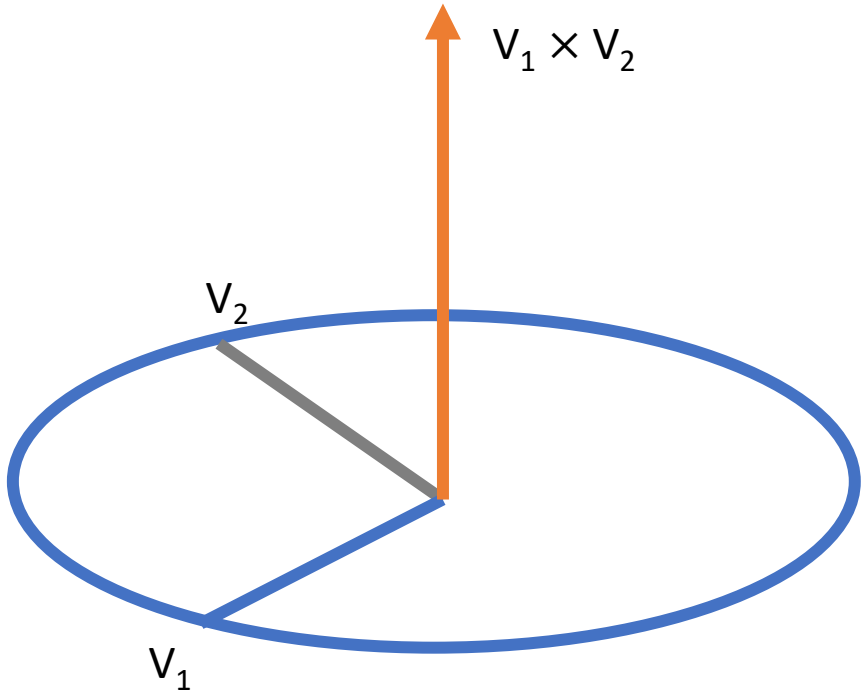
URL: https://www.youtube.com/watch?v=zc8b2Jo7mno

3. Initial alignment

Antisymmetric matrix (skew):

Usually, the 3D vector cross can be written as:

- $\boldsymbol{V}_1 = [V_{1x} \quad V_{1y} \quad V_{1z}]^{\mathrm{T}}$ and $V_2 = [V_{2x} \quad V_{2y} \quad V_{2z}]^{T}$
- $\boldsymbol{V}_1 \times \boldsymbol{V}_2 = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ V_{1x} & V_{1y} & V_{1z} \\ V_{2x} & V_{2y} & V_{2z} \end{vmatrix} = \begin{bmatrix} V_{1y}V_{2z} - V_{1z}V_{2y} \\ -(V_{1x}V_{2z} - V_{1z}V_{2x}) \\ V_{1x}V_{2y} - V_{1y}V_{2x} \end{bmatrix}$
- $\begin{bmatrix} 0 & -V_{1z} & V_{1y} \\ V_{1z} & 0 & -V_{1x} \\ -V_{1y} & V_{1x} & 0 \end{bmatrix} \begin{bmatrix} V_{2x} \\ V_{2y} \\ V_{2z} \end{bmatrix} = \begin{bmatrix} V_{1y}V_{2z} - V_{1z}V_{2y} \\ -(V_{1x}V_{2z} - V_{1z}V_{2x}) \\ V_{1x}V_{2y} - V_{1y}V_{2x} \end{bmatrix}$
- $V_1 \times V_2 = (V_1 \times)V_2$ is equal to $(\boldsymbol{V} \times) = -(\boldsymbol{V} \times)^{\mathrm{T}}$



3. Initial alignment

According to the Cayley–Hamilton theorem theory, the 3D vector can be expanded a matrix

$$e^{(V\times)} = \boldsymbol{I} + \frac{\sin v}{v}(\boldsymbol{V}\times) + \frac{1-\cos v}{v}(\boldsymbol{V}\times)^2$$

When *V* = [0.6096, 0.5747, 0.3260]

R = $e^{(V\times)}$ **=**

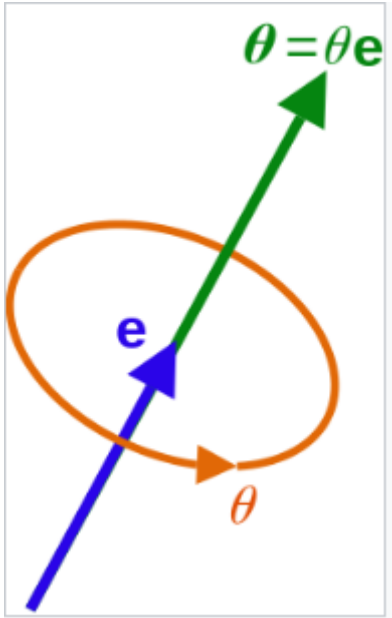
$$\begin{matrix} 0.7960 & -0.1201 & 0.5932 \\ 0.4475 & 0.7767 & -0.4432 \\ -0.4075 & 0.6183 & 0.6721 \end{matrix}$$

3. Initial alignment

Rotation Vector

Mr. Euler said that:

All the rotation can be described as a rotate the unit length axis *e* for *Θ*.



When v = [0, 0, 1], rotates 90 deg, it can be written like:

- $(\text{ axis, angle }) = \left(\begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}, \theta\right) = \left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \frac{\pi}{2}\right)$

- **That is why we see the Rodrigues function (Cayley–Hamilton theorem theory) many times.**

3. Initial alignment

As we know, the rotation matrix can be described by 3*3 matrix.

To be more efficient, there is a method called Quaternion，it only used 4 values.

- $Q = q_0 + q_v = q_0 + q_1 i + q_2 j + q_3 k$
- q_0 is the scalar part, q_1, q_2, q_3 is the vector part.
- // TODO

3. Initial alignment

Just like what we talked before, when we have two vectors in different coordinate, we can calculate the rotation matrix C_b_n.

Body frame vector:

$B = [Acc_{x, y, z}, Gyro_{x, y, z}, cross(Acc_{x, y, z}, Gyro_{x, y, z})]$;

Navigation frame vector:

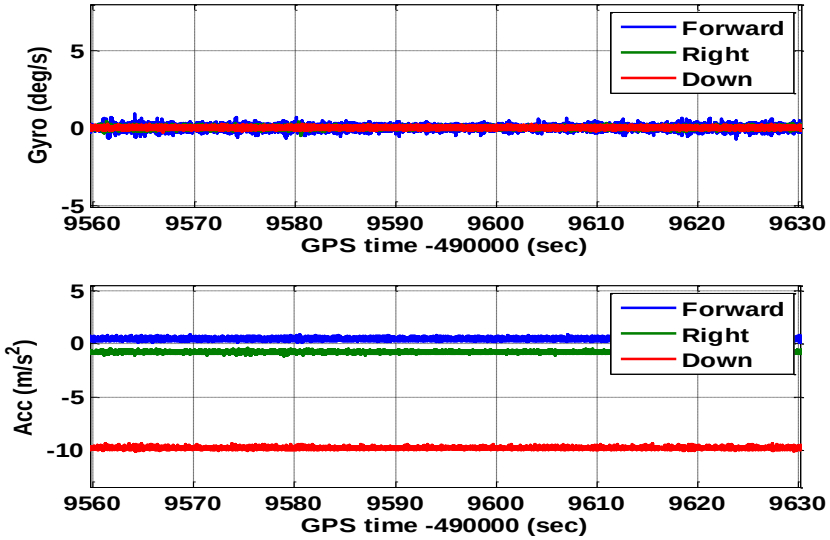
N = [g_vector, earth rotation vector, cross(g_vector, earth rotation vector)];

C_b_n = B * inv(N)

3. Initial alignment

The task is to calculate the body frame vector:

$$B = [Acc_{x,y,z}, Gyro_{x,y,z}, cross(Acc_{x,y,z}, Gyro_{x,y,z})];$$



There are two kinds of ways for IMU alignment:

	Advantages	Disadvantages
Static alignment	Long time stable and reliable	Have to the detect the earth rotation angle rate
Moving alignment	Short time	Speed rely on GPS; Few epochs for alignment maybe not reliable

3. Initial alignment

$B = [Acc_{x,y,z}, Gyro_{x,y,z}, cross(Acc_{x,y,z}, Gyro_{x,y,z})];$

	Euler angle(1)	Euler angle(2)	Euler angle(3)
Static alignment	cross(g, earth rotation)	cross(acc, gyro)	cross(Euler angle(1) , Euler angle(2))
Moving alignment	atan2(acc_y, acc_z)	arctan2(acc_x/g)	atan2(v_E, v_N)

4. Mechanization

To describe the mechanization equitation more clearly, I will show does it realized by program:

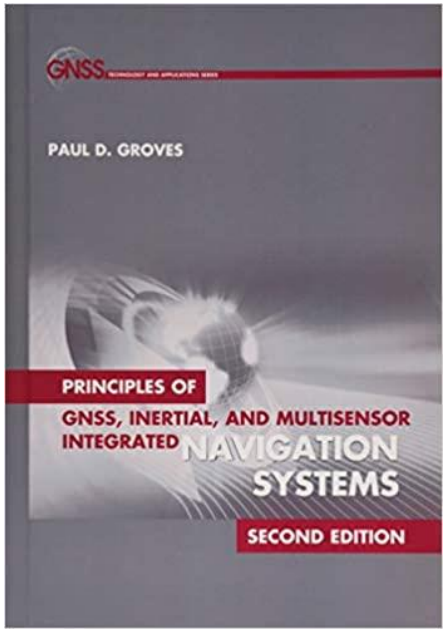
Program function name: Nav_equations_ECEF.m;

Language: MTALAB;

From: Mr. Paul D Groves;

IMU body frame: Forward, Right, Down

Unit: gyro (rad/s), acc(m/s*s)



*** Reference: Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems**

4. Mechanization

Input:

- % tor_i time interval between epochs (s)
- % old_r_eb_e previous Cartesian position of body frame w.r.t. ECEF frame, resolved along ECEF-frame axes (m)
- % old_C_b_e previous body-to-ECEF-frame coordinate transformation matrix
- % old_v_eb_e previous velocity of body frame w.r.t. ECEF frame, resolved along ECEF-frame axes (m/s)
- % f_ib_b specific force of body frame w.r.t. ECEF frame, resolved along body-frame axes, averaged over time interval (m/s^2)
- % omega_ib_b angular rate of body frame w.r.t. ECEF frame, resolved about body-frame axes, averaged over time interval (rad/s)

Parameters:

```
omega_ie = 7.292115E-5;  % Earth rotation rate (rad/s)
```

Output:

- % r_eb_e Cartesian position of body frame w.r.t. ECEF frame, resolved along ECEF-frame axes (m)
- % v_eb_e velocity of body frame w.r.t. ECEF frame, resolved along ECEF-frame axes (m/s)
- % C_b_e body-to-ECEF-frame coordinate transformation matrix

4. Mechanization

0. Coordination

Difference in ECI and ECEF

ECI*(Earth-centered inertial)*	inertial, not accelerated, fixed w.r.t stars; useful to describe motion of celestial bodies and spacecraft.
ECEF (*earth-centered,earth-fixed*)	not inertial, accelerated, rotating w.r.t stars; useful to describe motion of objects on Earth surface.

The IMU output is ECI measurement, and the satellite position is in ECEF, it necessary to convert the motion.

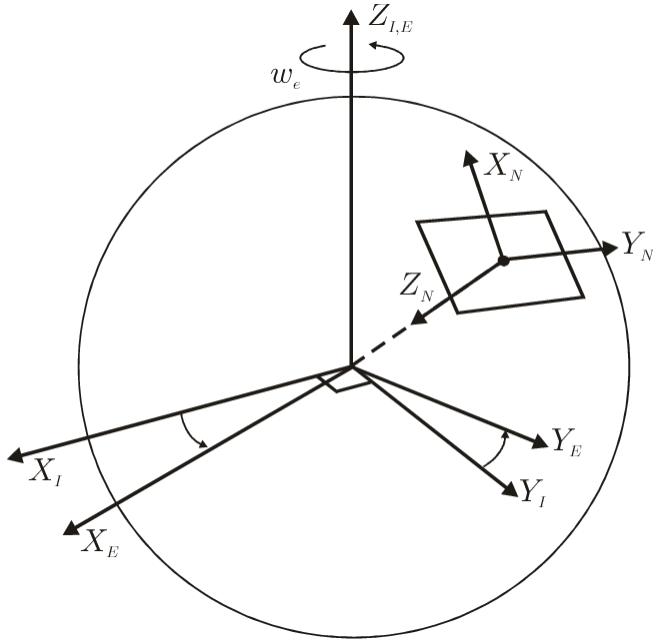
*** URL: https://en.wikipedia.org/wiki/Earth-centered_inertial**

4. Mechanization

1. Attitude update

1.1 From ECI to ECEF

The origin and z axis of the ECI coordinate system and the ECEF coordinate system are coaxial.



- alpha_ie = omega_ie * tor_i; % Earth rotation
- C_Earth = [cos(alpha_ie), sin(alpha_ie), 0;...
- -sin(alpha_ie), cos(alpha_ie), 0;...
- 0, 0, 1];

1.2 Attitude increment

- alpha_ib_b = omega_ib_b * tor_i; % angle increment
- mag_alpha = sqrt(alpha_ib_b' * alpha_ib_b); % angle increment length
- Alpha_ib_b = Skew_symmetric(alpha_ib_b); % skew-symmetric matrix

1. Attitude update

1.3 Obtain coordinate transformation matrix from Rodrigues' formula

- if mag_alpha>1.E-8 %if length of angle changed
- C_new_old = eye(3) + sin(mag_alpha) / mag_alpha * Alpha_ib_b +...
- (1 - cos(mag_alpha)) / mag_alpha^2 * Alpha_ib_b * Alpha_ib_b;
- Else %if |$\alpha\_ib\hat{}b$ | very small, using the approximate form
- C_new_old = eye(3) + Alpha_ib_b;
- end
- % Update attitude using (5.75)
- C_b_e = C_Earth * old_C_b_e * C_new_old;

$$\boldsymbol{C}_{b+}^{b-} = \boldsymbol{I}_3 + \frac{\sin\left|\boldsymbol{\alpha}_{ib}^{b}\right|}{\left|\boldsymbol{\alpha}_{ib}^{b}\right|}\left[\boldsymbol{\alpha}_{ib}^{b}\ \wedge\right] + \frac{1-\cos\left|\boldsymbol{\alpha}_{ib}^{b}\right|}{\left|\boldsymbol{\alpha}_{ib}^{b}\right|^2}\left[\boldsymbol{\alpha}_{ib}^{b}\ \wedge\right]^2$$

4. Mechanization

1. Attitude update

1.4 SPECIFIC FORCE FRAME TRANSFORMATION

```
if mag_alpha>1.E-8
    ave_C_b_e = old_C_b_e * (eye(3) + (1 - cos(mag_alpha)) / mag_alpha^2 ...
        * Alpha_ib_b + (1 - sin(mag_alpha) / mag_alpha) / mag_alpha^2 ...
        * Alpha_ib_b * Alpha_ib_b) - 0.5 * Skew_symmetric([0;0;alpha_ie])...
        * old_C_b_e;
else
    ave_C_b_e = old_C_b_e - 0.5 * Skew_symmetric([0;0;alpha_ie]) *...
        old_C_b_e;
end
```

$$C_b^{b-} = I_3 + \frac{1-\cos\left|\alpha_{ib}^b\right|}{\left|\alpha_{ib}^b\right|^2}\left[\alpha_{ib}^b\ \wedge\right] + \frac{1}{\left|\alpha_{ib}^b\right|^2}\left(1-\frac{\sin\left|\alpha_{ib}^b\right|}{\left|\alpha_{ib}^b\right|}\right)\left[\alpha_{ib}^b\ \wedge\right]^2$$

1.5 Transform specific force to ECEF-frame resolving axes

```
f_ib_e = ave_C_b_e * f_ib_b;
```

$$\bar{C}_b^e = \boldsymbol{C}_b^e(-)\boldsymbol{C}_b^{b-} - \frac{1}{2}\boldsymbol{\Omega}_i^e\boldsymbol{C}_b^e(-)\tau_i$$

4. Mechanization

2. UPDATE VELOCITY

- v_eb_e = old_v_eb_e + tor_i * (f_ib_e + Gravity_ECEF(old_r_eb_e) -...
- 2 * Skew_symmetric([0;0;omega_ie]) * old_v_eb_e);

- $\boldsymbol{v}_{eb}^{e}(+) = \boldsymbol{v}_{eb}^{e}(-) + \boldsymbol{v}_{ib}^{e} + (\boldsymbol{g}_{b}^{e}(\boldsymbol{r}_{eb}^{e}(-)) - 2\boldsymbol{\Omega}_{ie}^{e}\boldsymbol{v}_{eb}^{e}(-))\tau_i$

- Space force have to remove the gravity and minis the earth rotation;
- Coriolis acceleration doen't consider in MEMS IMU.

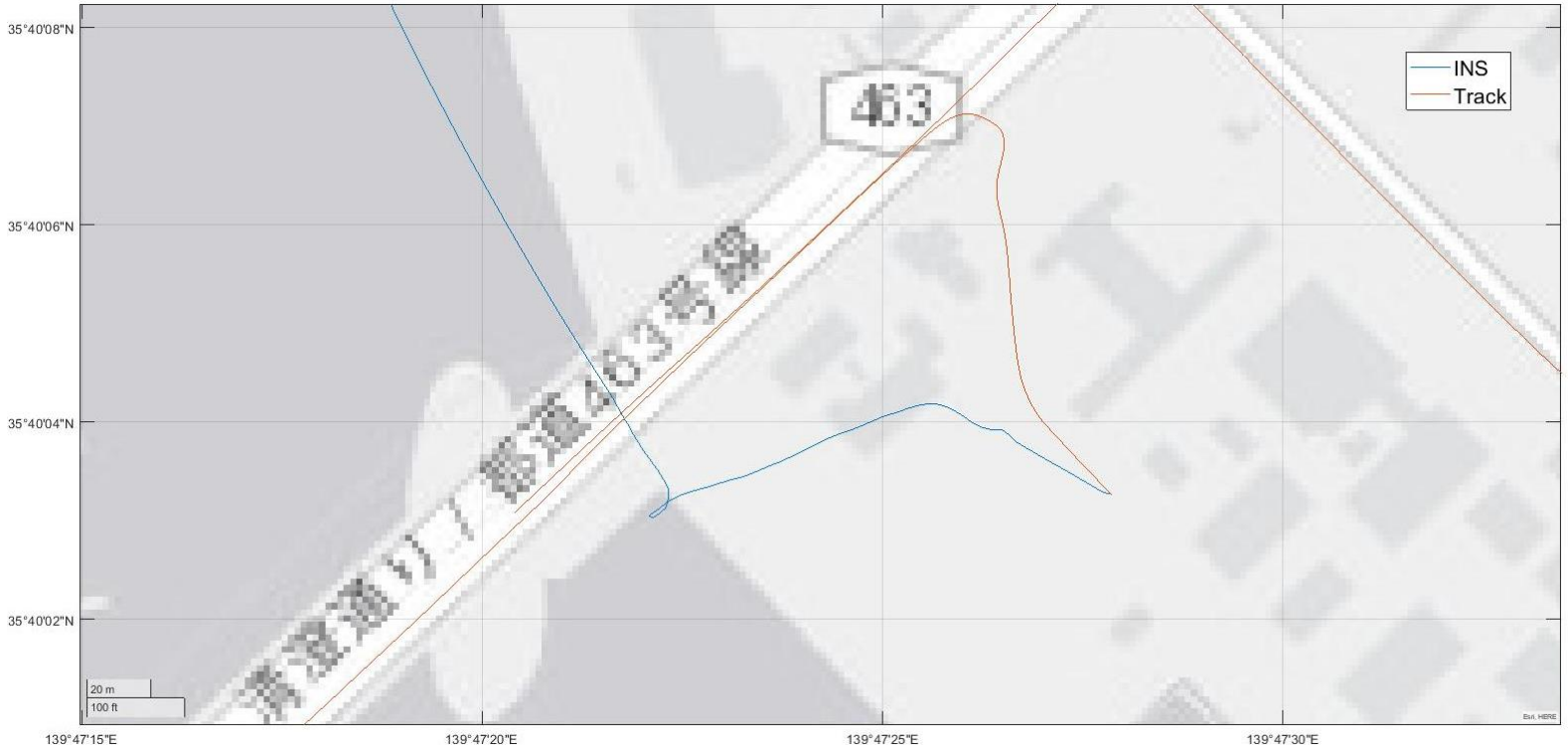
3. UPDATE CARTESIAN POSITION

- r_eb_e = old_r_eb_e + (v_eb_e + old_v_eb_e) * 0.5 * tor_i;

- $r_{eb}^{e}(+) = r_{eb}^{e}(-) + \left(v_{eb}^{e}(-) + v_{eb}^{e}(+)\right)\frac{\tau_i}{2}$

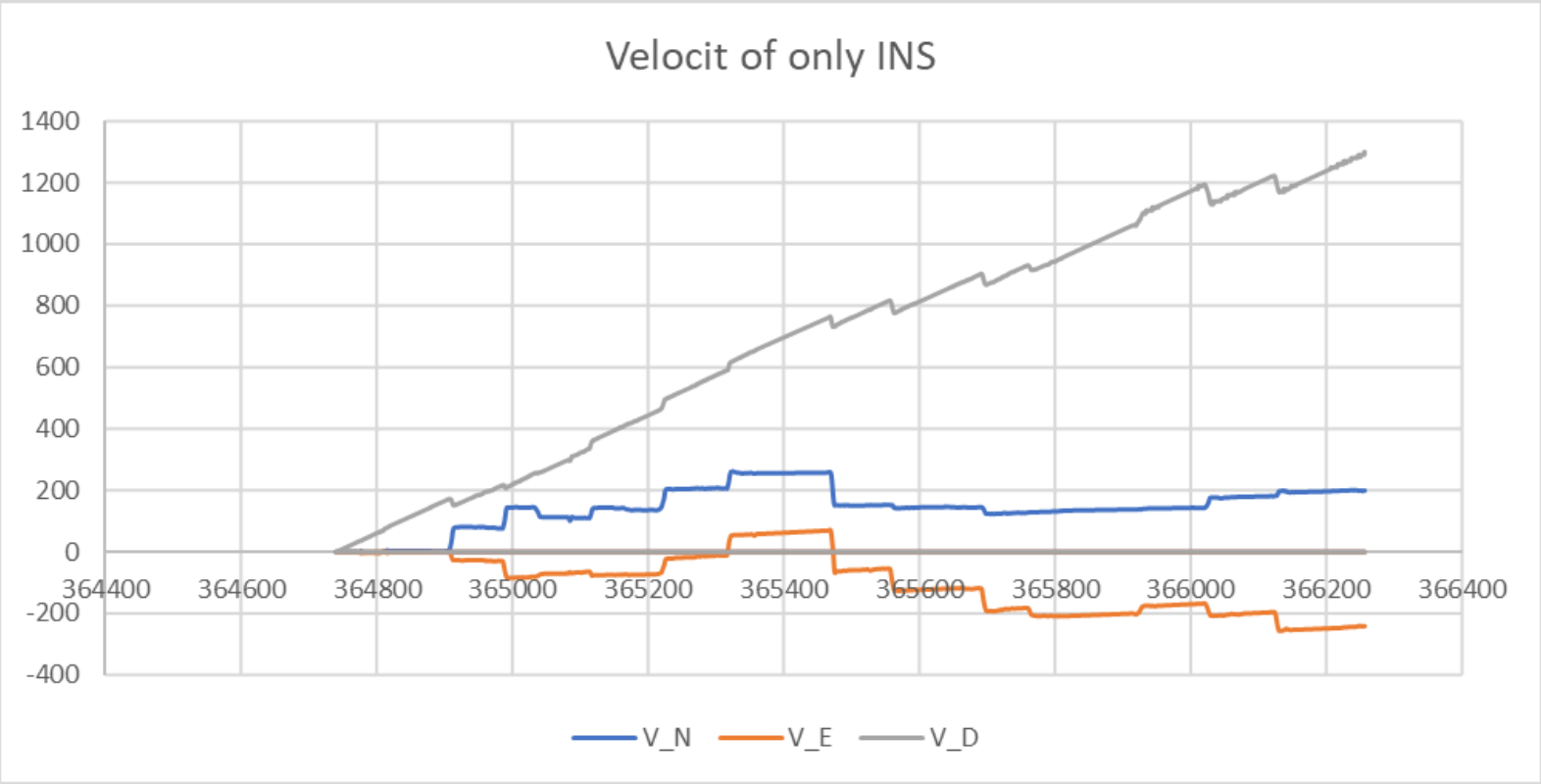
- Position update in the ECEF coordination, doesn't need rotation.
- It is assumed that the velocity changes linearly during the integration period.

4. Result of only INS



AS we see, the MEMS is hard to detect the earth rotation,
so the alignment angle has a huge error, can not dead-rocking for a long time.

4. Result of only INS



AS we see, it is hard to use only MEMS for navigation.

5. GNSS clock error and clock rate

Now, we already finished the IMU part.
let's talking about the TC input data of GNSS!

Program function name: Main_PP_n.m;

Language: MTALAB;

Function: Input the "rover.obs" **and "**base.nav" **files, convert the raw data to** "ObsHead", "ObsData", "NavData"**, then calculate each epoch satellites' position, velocity, receiver position, velocity, clock error and drift etc.**

GNSS version: RINEX 3.02;

Measurement used: C1C, D1C;

- **URL:** https://github.com/dzd9798/READ_GNSS;
  https://github.com/YizeZhang/Net_Diff;

5. GNSS clock error and clock rate

1. generateGnssData.m

This function input are “*.obs” and “*.nav” files.

It can calculate the raw data to “ObsHead”, “ObsData”, “NavData”.

In order to save time for second time use, you and saved it to “.mat” format.

ObsHead includes:

File version, interval and the satellite system be used.

ObsData includes:

GPStime, number of satellites, PRN, measurement, satellite position, etc.

NavData includes:

Different satellites correction information.

5. GNSS clock error and clock rate

2. Initial setting

- % Receiver start position and velocity (From RTK result)
- ApproCoor = [-3961909.5034 ,3348998.9979 ,3698217.0898]; %init receiver position in ecef
- ApproCoorV_ = [0.064726, -0.051345, -0.021533]; %init receiver velocity in ecef

- % Velocity of light (m/s)
- c = 299792458;

- %Frequency of L1 (Hz)
- F1 = 1575420000.0;

- % Order from RINEX 3.02
- C1C=1; % C/A code pseudo-range of L1
- D1C=3; % C/A code Doppler frequency of L1
- S1C=4; % C/A code SNR of L1

5. GNSS clock error and clock rate

3. Signal point position

This program was changed from a SPP program:

- `for epoch=1:` Max epoch
- `for num=1:` Number of satellites
- if satellite less than 4
- continue;
- end
- Calculate satellite positions and velocity;
- Store data for TC;
- build matrix for least square;
- Calculate the receiver position and velocity;
- end
- end

- GNSS time;
- Satellite PRN;
- Signal frequency pseudo range removed Ionosphere, troposphere, relativistic delay and satellite clock error;
- Satellite position in X, Y, Z;
- Doppler rate from satellite measurement;
- satellite clock rate;
- Satellite velocity in X, Y, Z;
- Satellite elevation (degree);
- SNR form satellite measurement;
- Receiver clock drift correction.

5. GNSS clock error and clock rate

4. satellite clock and satellite clock rate

- tempNav: Nav data in this epoch;
- af0: Satellite clock deviation (s);
- af1: Satellite clock drift (s/s);
- af2: Drift velocity of satellite clock (s/s*s) ;
- tk: Time difference between the time when the satellite transmits the signal and the reference time;
- tgd1: Consider the tgd correction when only use L1 frequency.

- sat_clk=tempNav.af0+tk*tempNav.af1+tk^2*tempNav.af2 - tempNav.tgd1;

- rate_clock = tempNav.af1 + tempNav.af2 * tk;

5. Store result

In the end, we can save the result:

GNSSTCData: Input data for GNSS/IMU tightly coupling;

Rec_Clk: receiver clock error;

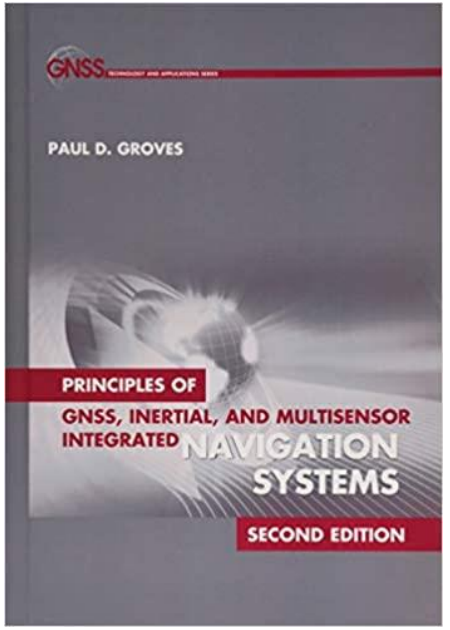
Rec_Clk_rate: receiver clock rate;

Obs_Coor: receiver ECEF position in X, Y and Z;

Obs_CoorV_: receiver ECEF velocity in X, Y and Z;

6. Kalman filter in TC

To describe the mechanization equitation more clearly, I will show does it realized by program:



Program function name: TC_KF_Epoch.m;

Language: MTALAB;

From: Mr. Paul D Groves and

https://github.com/benzenemo/TightlyCoupledINSGNSS.

IMU body frame: Forward, Right, Down

Unit: gyro (rad/s), acc(m/s*s)

*** Reference: Principles of GNSS, Inertial, and Multisensory Integrated Navigation Systems**

6. Kalman filter in TC

0 Input:

- % GNSS measurement data:

Pseudo-range measurements (m) , Pseudo-range rate measurements (m/s), Satellite ECEF position (m), Satellite ECEF velocity (m/s) , Elevation angle (deg)

- % no_meas Number of satellites for which measurements are supplied
- % IMU measurement data:

Position and velocity in ECEF, bias, Kalman filter state estimates, measured specific force, previous latitude solution

- % tor_s propagation interval (s)
- % P_matrix_old previous Kalman filter error covariance matrix
- % TC_KF_config
- % Clock_Reset_ estimated receiver clock offset (m) , estimated receiver clock drift (m/s)
- % P_matrix_new updated Kalman filter error covariance matrix
- % R_matrix measurement noise covariance matrix

Parameters:

- omega_ie = 7.292115E-5; % Earth rotation rate in rad/s
- R_0 = 6378137; %WGS84 Equatorial radius in meters
- e = 0.0818191908425; %WGS84 eccentricity

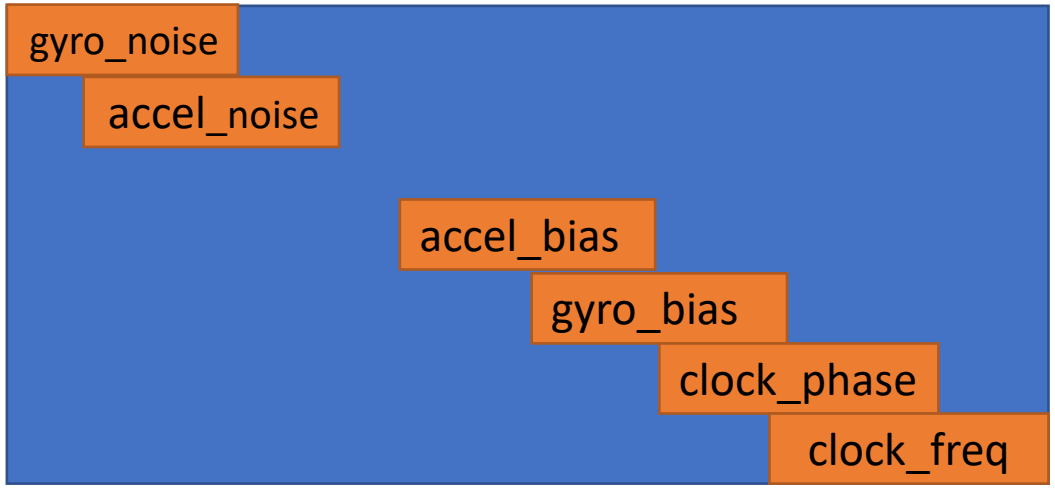
6. Kalman filter in TC

1 Build P matrix:

- $\boldsymbol{\Phi}^{e}_{\mathrm{INS}} \approx \begin{bmatrix} \boldsymbol{I}_3 - \boldsymbol{\Omega}^{e}_{ie}\tau_s & 0_3 & 0_3 & 0_3 & \widehat{\boldsymbol{C}}^{e}_{b}\tau_s \\ \boldsymbol{F}^{e}_{21}\tau_s & I_3 - 2\boldsymbol{\Omega}^{e}_{ie}\tau_s & \boldsymbol{F}^{e}_{23}\tau_s & \boldsymbol{C}^{e}_{b}\tau_s & 0_3 \\ 0_3 & I_3\tau_s & I_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & I_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & I_3 \end{bmatrix}$
- Phi_matrix = eye(17);
- Phi_matrix(1:3,1:3) = Phi_matrix(1:3,1:3) - Omega_ie * tor_s;
- Phi_matrix(1:3,13:15) = est_C_b_e_old * tor_s;
- Phi_matrix(4:6,1:3) = -tor_s * Skew_symmetric(est_C_b_e_old * meas_f_ib_b);
- Phi_matrix(4:6,4:6) = Phi_matrix(4:6,4:6) - 2 * Omega_ie * tor_s;
- geocentric_radius = R_0 / sqrt(1 - (e * sin(est_L_b_old))^2) * sqrt(cos(est_L_b_old)^2 + (1 - e^2)^2 * sin(est_L_b_old)^2); Phi_matrix(4:6,7:9) = -tor_s * 2 * Gravity_ECEF(est_r_eb_e_old) / geocentric_radius * est_r_eb_e_old' / sqrt (est_r_eb_e_old’ * est_r_eb_e_old);
- Phi_matrix(4:6,10:12) = est_C_b_e_old * tor_s;
- Phi_matrix(7:9,4:6) = eye(3) * tor_s;
- Phi_matrix(16,17) = tor_s;

6. Kalman filter in TC

2 Build Q matrix：



- Q_prime_matrix = zeros(17);
- Q_prime_matrix(1:3,1:3) = eye(3) * TC_KF_config.gyro_noise_PSD * tor_s;
- Q_prime_matrix(4:6,4:6) = eye(3) * TC_KF_config.accel_noise_PSD * tor_s;
- Q_prime_matrix(10:12,10:12) = eye(3) * TC_KF_config.accel_bias_PSD * tor_s;
- Q_prime_matrix(13:15,13:15) = eye(3) * TC_KF_config.gyro_bias_PSD * tor_s;
- Q_prime_matrix(16,16) = TC_KF_config.clock_phase_PSD * tor_s;
- Q_prime_matrix(17,17) = TC_KF_config.clock_freq_PSD * tor_s;

*PSD power spectral density

6. Kalman filter in TC

3 Clock jump repair:

```
% GNSS clock skip repair identification, for the repaired clock, the residual clock error that has not been repaired is 0

% x_est_propagated is the  state estimates

x_est_propagated = zeros(17,0);

• if Clock_Reset_Flag==1
•     x_est_propagated(16,1)=0;
•     x_est_propagated(17,1)=0;
• else
•     x_est_propagated(16,1) = est_clock_old(1) + est_clock_old(2) * tor_s;
•     x_est_propagated(17,1) = est_clock_old(2);
• end
```

6. Kalman filter in TC

4.1 Update P matrix:

- P_matrix_propagated = Phi_matrix * (P_matrix_old + 0.5 * Q_prime_matrix) *...
- Phi_matrix' + 0.5 * Q_prime_matrix;
- $\boldsymbol{P}_k^- \approx \boldsymbol{\Phi}_{k-1}\left(\boldsymbol{P}_{k-1}^+ + \frac{1}{2}\boldsymbol{Q}'_{k-1}\right)\boldsymbol{\Phi}_{k-1}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{Q}'_{k-1}$

6. Kalman filter in TC

4.2 Loop measurement:

for j = 1:no_meas

- %Satellite position already corrected the earth roatation
- delta_r = GNSS_measurements(j,3:5)' - est_r_ea_e_old;
- range = sqrt(delta_r' * delta_r); %Distance between sat and rec
- pred_meas(j,1) = range + x_est_propagated(16);
- u_as_e_T(j,1:3) = delta_r' / range; % Predict line of sight
- % Predict pseudo-range rate using (8.45) ignore Sagnac
- range_rate = u_as_e_T(j,1:3) * (GNSS_measurements(j,6:8)'-est_v_ea_e_old); % (sat_vel – rec_vel)
- pred_meas(j,2) = range_rate + x_est_propagated(17);
- end

$$\begin{cases} \hat{\rho}_{a,c,k}^{j-} = \sqrt{\left[C_c^I\left(\tilde{t}_{st,a,k}^{j}\right)\hat{r}_{ej}^{e}\left(\tilde{t}_{st,a,k}^{j}\right) - \hat{r}_{ea,k}^{e-}\right]^{\mathrm{T}}\left[C_e^l\left(\tilde{t}_{st,a,k}^{j}\right)\hat{r}_{ej}^{e}\left(\tilde{t}_{st,a,k}^{j}\right) - r_{ea,k}^{e-}\right]} + \delta\hat{\rho}_{c,k}^{a-} \\ \dot{\hat{\rho}}_{a,c,k}^{j-} = \hat{u}_{asj}^{e-\mathrm{T}}\left[C_e^I\left(\tilde{t}_{st,a,k}^{j}\right)\left(\hat{v}_{ej}^{e}\left(\tilde{t}_{st,a,k}^{j}\right) + \Omega_{ie}^{e}\hat{r}_{ej}^{e}\left(\tilde{t}_{st,a,k}^{j}\right)\right) - \left(\hat{v}_{ea,k}^{e-} + \Omega_{ie}^{e}\hat{r}_{ea,k}^{e-}\right)\right] + \delta\dot{\hat{\rho}}_{c,k}^{a-} \end{cases}$$

6. Kalman filter in TC

5 build H matrix:

% pseudo-range:

- H_matrix(1:no_meas,7:9) = u_as_e_T(1:no_meas,1:3);
- H_matrix(1:no_meas,16) = ones(no_meas,1);

$$\boldsymbol{H}_{G,k}^{\gamma} \approx \begin{pmatrix} 0_{1,3} & 0_{1,3} & u_{a1}^{\gamma\mathrm{T}} & 0_{1,3} & 0_{1,3} & 1 & 0 \\ 0_{1,3} & 0_{1,3} & u_{a2}^{\gamma\mathrm{T}} & 0_{1,3} & 0_{1,3} & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{1,3} & 0_{1,3} & u_{am}^{\gamma\mathrm{T}} & 0_{1,3} & 0_{1,3} & 1 & 0 \\ \ldots. & \ldots. & \ldots. & \ldots. & \ldots. & \ldots. & \ldots. \\ 0_{1,3} & u_{a1}^{\gamma\mathrm{T}} & 0_{1,3} & 0_{1,3} & 0_{1,3} & 0 & 1 \\ 0_{1,3} & u_{a2}^{\gamma\mathrm{T}} & 0_{1,3} & 0_{1,3} & 0_{1,3} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{1,3} & u_{am}^{\gamma\mathrm{r}} & 0_{1,3} & 0_{1,3} & 0_{1,3} & 0 & 1 \end{pmatrix}_{x=\hat{x}_k} , \gamma \in i, e$$

% pseudo-range rate:

- H_matrix((no_meas + 1):(2 * no_meas),4:6) = u_as_e_T(1:no_meas,1:3);
- H_matrix((no_meas + 1):(2 * no_meas),17) = ones(no_meas,1);

6. Kalman filter in TC

6 Set the R matrix:

- index_LowEle=GNSS_measurements(:,9)<30;
- ElevationGain(index_LowEle)=1./(2*sind(GNSS_measurements(index_LowEle,9)));
- R_matrix(1:no_meas,1:no_meas) = diag(ElevationGain)*...
- TC_KF_config.pseudo_range_SD^2;
-
- R_matrix((no_meas + 1):2*no_meas,(no_meas + 1):2*no_meas) =...
- diag(ElevationGain.*ElevationGain)...
- * TC_KF_config.range_rate_SD^2;

- Adjust the measurement quality by satellite elevation.

7 & 8 Calculate K and Z:

```
K_matrix = P_matrix_propagated * H_matrix' /(H_matrix *...
    P_matrix_propagated * H_matrix' + R_matrix);

% pseudo-range:
delta_z(1:no_meas,1) = GNSS_measurements(1:no_meas,1) -...
    pred_meas(1:no_meas,1);
% pseudo-range rate:
delta_z((no_meas + 1):(2 * no_meas),1) = GNSS_measurements(1:no_meas,2) -...
    pred_meas(1:no_meas,2);
```

6. Kalman filter in TC

9 & 10 Update state estimates and estimation error covariance:

x_est_new = x_est_propagated + K_matrix * delta_z;

- $$\begin{aligned}\hat{\boldsymbol{x}}_k^+ &= \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k^-) \\ &= \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\delta\boldsymbol{z}_k^-\end{aligned}$$

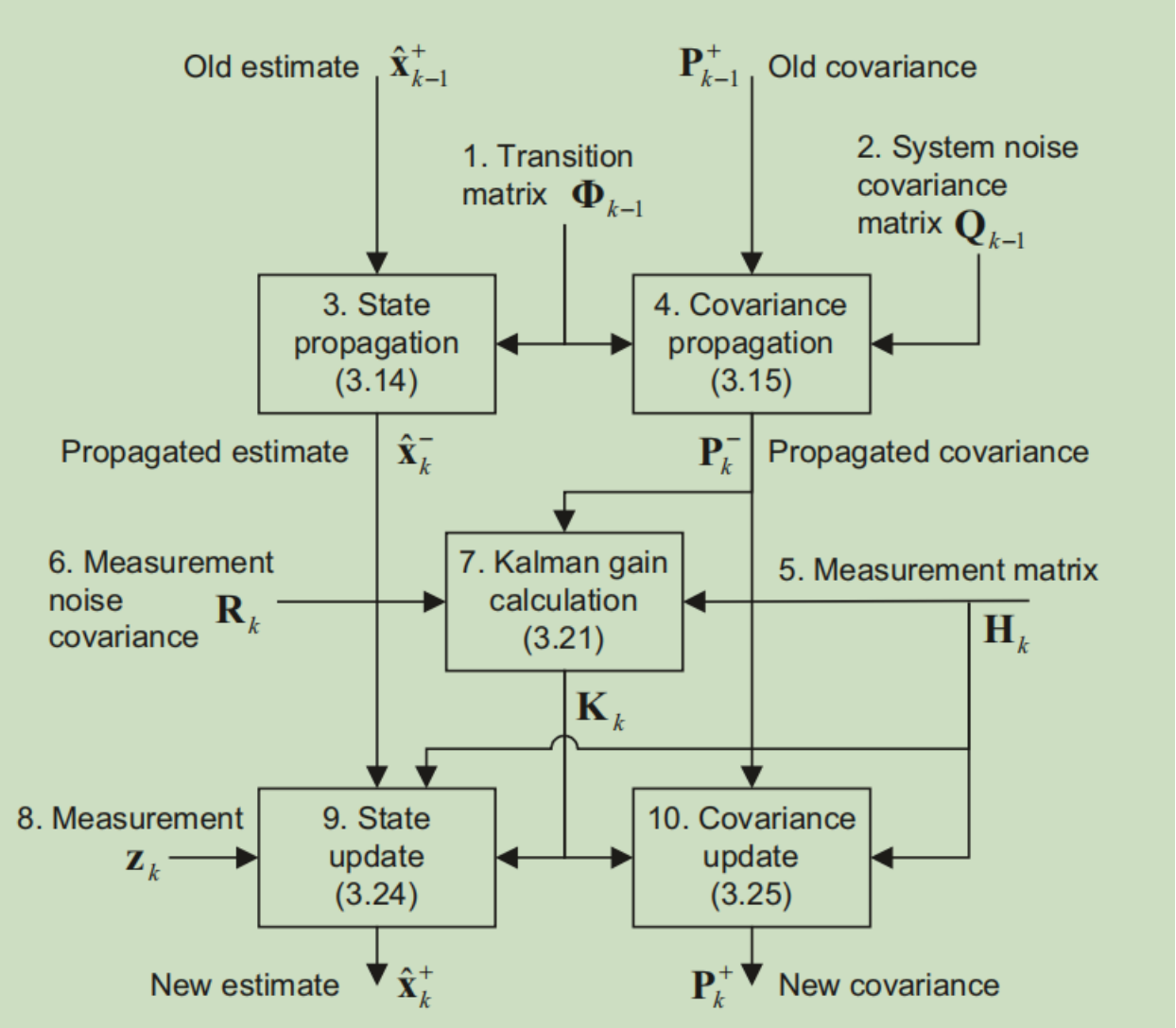
P_matrix_new = (eye(17) - K_matrix * H_matrix) * P_matrix_propagated;

- $\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_k^-$

11 Close loop correction

- % Correct attitude, velocity, and position
- est_C_b_e_new = (eye(3) - Skew_symmetric(x_est_new(1:3))) * est_C_b_e_old;
- est_v_eb_e_new = est_v_eb_e_old - x_est_new(4:6);
- est_r_eb_e_new = est_r_eb_e_old - x_est_new(7:9);
-
- % Update IMU bias and GNSS receiver clock estimates
- est_IMU_bias_new = est_IMU_bias_old + x_est_new(10:15);
- est_clock_new = x_est_new(16:17)';

6. Kalman filter in TC



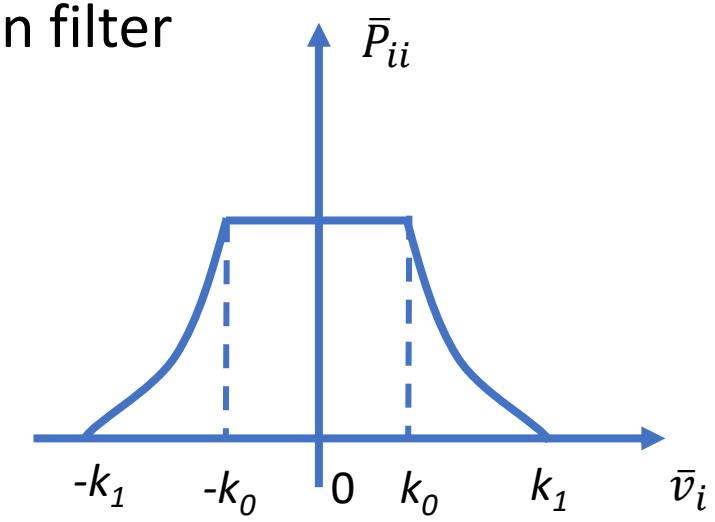
6. Kalman filter in TC

IAE-KF: Innovation-based adaptive estimation Kalman filter

The program set a value MGain to expend the R matrix.

For those satellites which stand innovation are exceeded,

We can reset the weight.



- $StdInn = Z/\sqrt{\mathrm{diag}(H * P * H' + R)}$

- If abs(StdInn > k1), change the *MGain* to a huge value;

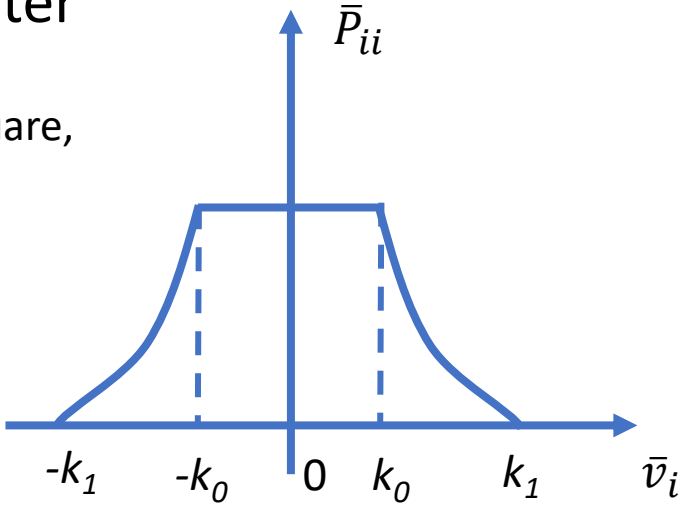
- If abs(StdInn < k1) and abs(StdInn > k0):
- $MGain = \frac{(k_1 - k_0)^2}{k_0} * \mathrm{diag}(\frac{|StdInn|}{(k_1 - |StdInn|)^2})$
- R_matrix=MGain*R_matrix;

$$\bar{p}_i = \begin{cases} p_i & |\tilde{V}_i| \leqslant k_0 \\ p_i \frac{k_0}{|\tilde{V}_i|}\left(\frac{k_1 - |\tilde{V}_i|}{k_1 - k_0}\right)^2 & k_0 < |\tilde{V}_i| \leqslant k_1 \\ 0 & |\tilde{V}_i| > k_1 \end{cases}$$

6. Kalman filter in TC

M-LS-KF: Robost M esitmation least square Kalman filter

M-estimation is to estimate the regression coefficient by iterative least square, and determine the weight of samples according to the size of the previous regression residual.



- $Resdiual = (I - H * K) - Z$
- UnitWeightVar = (Z / (R + H * P * H')) * Z/n
- CovResdiual = R – H * P * H'
- StdResdiual = Resdiual / sqrt(UnitWeightVar) / sqrt(diag(CovResdiual));

- If abs(StdResdiual > k1), change the *MGain* to a huge value;

- If abs(StdInn < k1) and abs(StdInn > k0):
- $MGain = \frac{(k_1 - k_0)^2}{k_0} * \text{diag}(\frac{|\text{StdResdiual}|}{(k_1 - |\text{StdResdiual}|)^2})$
- R_matrix=MGain*R_matrix;

$$\bar{p}_i = \begin{cases} p_i & |\tilde{V}_i| \leqslant k_0 \\ p_i \frac{k_0}{|\tilde{V}_i|} \left(\frac{k_1 - |\tilde{V}_i|}{k_1 - k_0} \right)^2 & k_0 < |\tilde{V}_i| \leqslant k_1 \\ 0 & |\tilde{V}_i| > k_1 \end{cases}$$

7. Usage of TC program

There are many parameters for setting:

Epoch interval; %frequency of GNSS data

Mask angle and signal strength ; %for selecting the satellites

TC_KF_config; %there are many PSD values of GNSS and IMU noise

. KFMethod; % ClassicKF ,M-LSKFClassicKF and IAE-KF

Start time:

old_time; %start time of the alignment

old_est_r_ea_e & old_est_v_ea_e; % position and velocity at start time

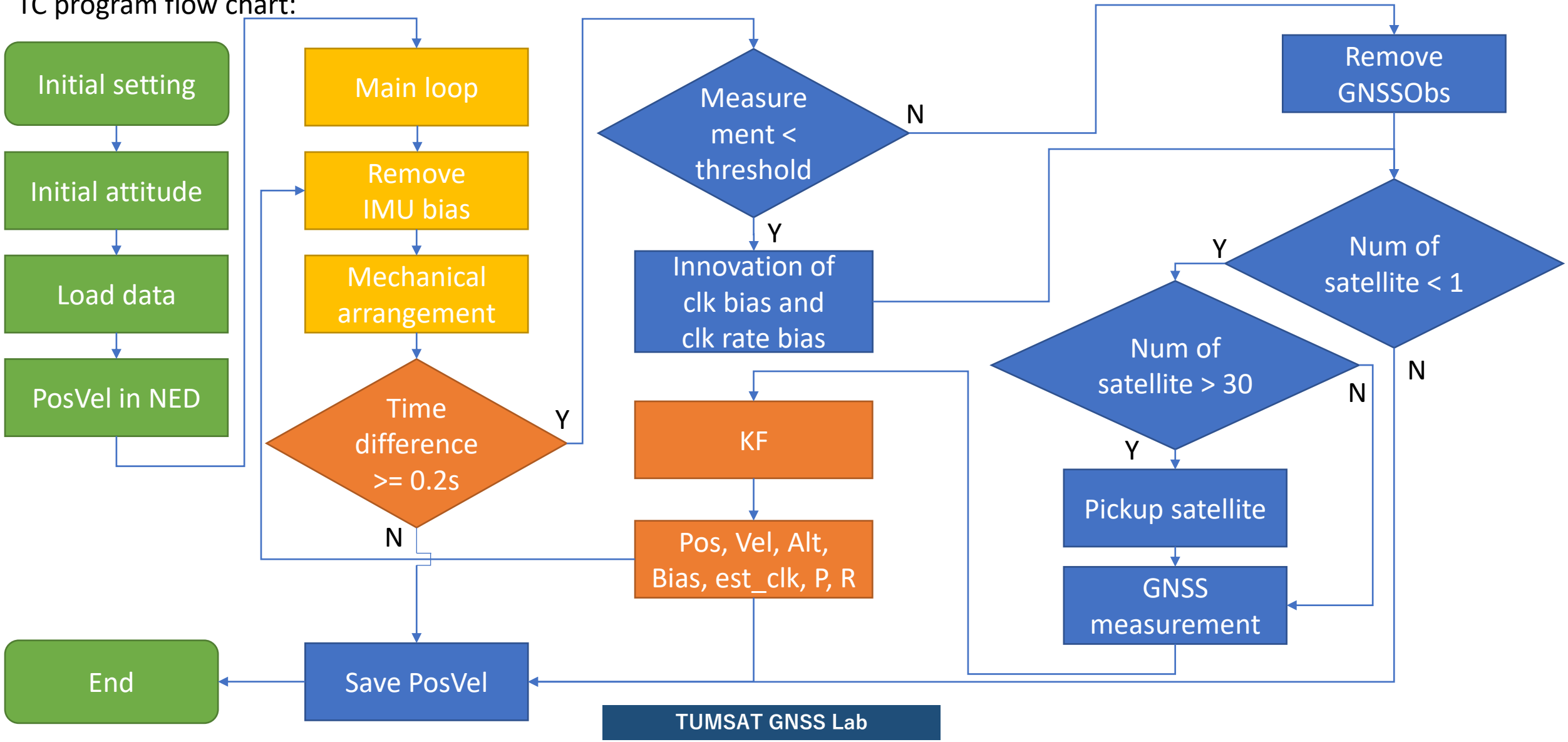
est_clock; %receiver clock error and drift at start time

attitude_ini; %Euler angle (3)-(2)-(1)

L_ba_b; %Lever arm IMU to antenna (F-R-D)

7. Usage of TC program

TC program flow chart:

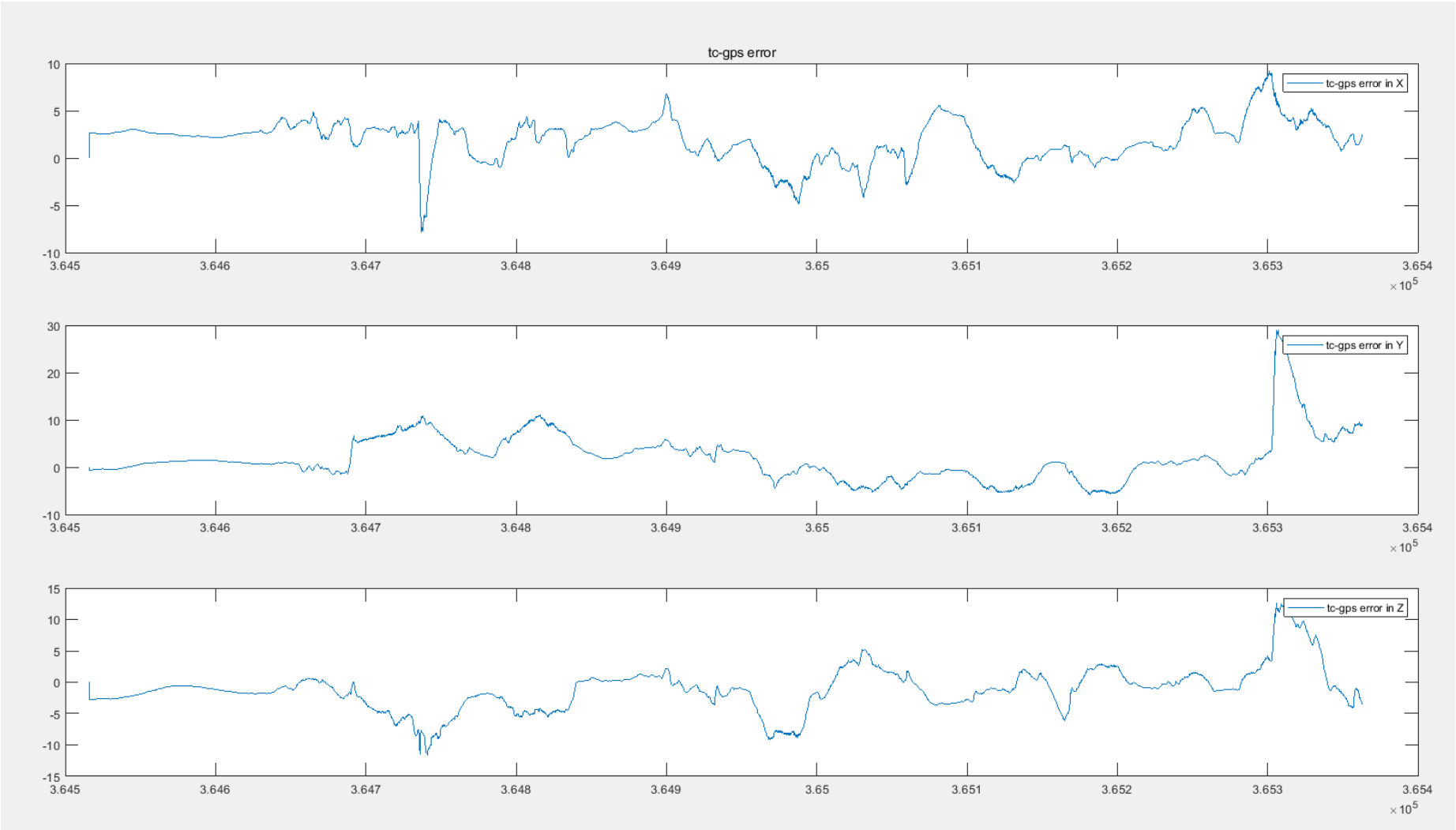


7. Usage of TC program (KF)

7. Usage of TC program (IAE-KF)

7. Usage of TC program (M-LSKF)

7. Usage of TC program (M-LSKF)



7. Usage of TC program

Result of 1105, compare with U-blox NMEA ECEF position:

GPS and QZSS L1 been used, mask angle is 10 deg;

IMU alignment angle is [0;0;-44.09582024712471]; (row, pitch, yaw)

GQ_L1	max-x	max-y	max-z	mean-x	mean-y	mean-z	std-x	std-x	std-z
spp	108.037	115.875	71.277	0.676291	-2.85485	-2.86478	6.863399	5.726574	5.726574
dgps	98.929	46.026	40.329	-0.41125	-2.40729	-2.75875	10.01121	6.710364	6.710364
rtk	98.929	46.974	40.414	-0.37678	-2.44548	-2.87893	10.03366	6.781295	6.781295
lc-rtk-gnss	68.441	36.641	40.068	-0.37771	-2.40806	-2.80642	8.884419	6.180154	6.180154
tc-kf	18.817	31.292	19.1884	4.720501	-1.55786	-3.29365	3.700006	5.934414	5.934414
tc-MLS	11.2218	9.5413	11.7903	3.46035	0.173291	-2.19483	2.766932	3.789179	3.789179
tc-IAE	9.1559	29.0443	12.6545	1.889393	1.939662	-1.1691	2.201533	4.926717	4.926717

We can see that filter methods play a import role in TC, MLS method has the best performance.

For classical KF TC, the mean error is worse than LC, but standard deviation is better than LC.