RTKLIB Manual

ver. 1.0　2007/01/23

目　　次

1　はじめに

1.1　RTKLIB の概要

RTKLIB は RTK (Realtime Kinematic) - GPS 測位アルゴリズムの評価確認のため開発した、C 言語により記述された簡潔で可搬性の高い RTK-GPS 測位演算ライブラリです。RTKLIB は以下の機能を有しています。

(1) 行列・ベクトル演算
(2) 時刻・文字列処理
(3) 座標系変換
(4) RINEX 観測データ、航法メッセージファイル読み込み
(5) OTF 整数 Ambiguity 決定
(6) 航法演算処理
(7) 対流圏モデル、電離層モデル計算
(8) 単独測位演算、RTK-GPS 相対測位演算

RTKLIB には RTKLIB ライブラリを使用して後処理基線解析を行うアプリケーションプログラム RNX2RTKP が含まれています。

2 仕様

2.1 実行環境

RTKLIB の構築及び実行には ANSI 標準 C プログラムがコンパイル・リンクできる環境が必要になります。外部の行列演算ライブラリを使用する場合、BLAS/LAPACK または互換ライブラリがインストールされている必要があります (内蔵ルーチンを使うことも出来ます)。実行確認済み実行環境は以下の通りです。

(1) Windows XP、Cygwin, gcc + BLAS/LAPACK
(2) Windows XP、MS Visual Studio.NET 2002＋MSC/C++ (BLAS/LAPACK なし)
(3) Windows XP、MS Visual Studio.NET 2002+ Intel C/C++ 8.0 + Intel MKL 7.2

2.2 機能

(1) 行列・ベクトル演算
・行列メモリ確保、ゼロ行列、単位行列、内積、ノルム、行列印刷
(2) 時刻・文字列処理
・文字列数値変換、文字列時刻変換、カレンダー時刻変換、GPS 時刻変換、GPST-UTC 変換
(3) 座標系変換
・ECEF-緯度経度高度変換、ECEF-局地座標系変換、ジオイド高
(4) RINEX 観測データ、航法メッセージファイル読み込み
(5) OTF 整数 Ambiguity 決定
・LAMBDA/MLAMBDA 整数不定性決定
(6) 航法演算処理
・暦計算、衛星位置・時計バイアス計算、幾何学距離計算、衛星方位仰角計算
(7) 対流圏モデル、電離層モデル計算
・対流圏モデル：Saastamoinen モデル＋標準大気、・電離層モデル：Klobucher モデル
(8) 単独測位演算
(9) RTK-GPS 相対測位演算

3 操作説明

3.1 RTKLIB のインストール

(1) パッケージ **rtklib_<ver>.tar.gz** または **rtklib_<ver>.zip** を適当なディレクトリの下に解凍して下さい。

(2) パッケージのディレクトリ構成は以下の通りです。

rtklib /src	: ソースプログラム
/gcc	: gcc 用メイク環境 (+BLAS/LAPACK)
/msc	: MS Visual Studio + MS C 用プロジェクト (BLAS/LAPACK なし)
/icc	: MS Visual Studio + Intel C/C++用プロジェクト (+Intel MKL)
/util	: ユーティリティ
/test	: 試験用データ、プログラム
/doc	: 文書ファイル

3.2 RTKLIB ライブラリ及び AP の構築

(1) gcc を使用する場合、以下のコマンドを入力してください。**rtklib/gcc** 下にオブジェクトファイル及び RTK ライブラリ **rtklib.a** 及びアプリケーションプログラム **rnx2rtkp(.exe), pos2kml(.exe)**が生成されます。

```
> cd rtklib/gcc
> make
```

(2) Visual Studio+MSC や Intel C/C++を使用する場合 **msc/msc.sln** または **icc/icc.sln** ファイルをダブルクリックしソリューションを起動してからメニュー「ビルド」－「ソリューションのリビルド」でビルドして下さい。RTK ライブラリ **rtklib.lib** 及びアプリケーションプログラム **rnx2rtkp.exe, pos2kml.exe** が生成されます。

(3) 組み込みの行列演算ルーチンを使用し、BLAS/LAPACK を使用しない場合、C/C++のコンパイルオプションとしてマクロ **NOLAPACK** 定義を追加して下さい。

(4) BLAS/LAPACK の代わりに Intel MKL を使用する場合、C/C++のコンパイルオプションとし

てマクロ **MKL** 定義を追加して下さい。またリンクオプションとして Intel MKL を入力に追加してください。詳細は Intel MKL のマニュアルを参照下さい。

3.3 RTKLIB ライブラリの利用

(1) RTKLIB の関数をアプリケーションプログラムから利用するためには、アプリケーションプログラム中で以下のヘッダファイルをインクルードして下さい。

rtklib/src/rtklib.h

(2) アプリケーションプログラムのリンクオプションとして RTK ライブラリ **rtklib/gcc/rtklib.a** または **rtklib/msc/Release/rtklib.lib** を入力として追加して下さい。

(3) アプリケーションプログラムから使用できる RTKLIB の関数仕様については付録 A を参照して下さい。

3.4 RNX2RTKP による後処理基線解析

(1) 3.1 で構築したアプリケーションプログラム **rnx2rtkp(.exe)**をパスの通ったディレクトリにコピーして下さい。

(2) 後処理基線解析にはローバ及び基準局の RINEX OBS 形式の GPS 観測データ、RINEX NAV 形式の航法メッセージファイルが必要になります。この例ではそれぞれのファイル名が以下であるとします。

rov0010.07o ref0010.07o ref0010.07n

(3) コンソール(コマンドプロンプト)で以下のコマンドを入力して下さい。

```
> rnx2rtkp rov001.07o ref0010.07o ref0010.07n > output.pos
```

(4) プログラムの実行が終わったらエディタ等で出力ファイル **output.pos** の内容を確認して

下さい。先頭**%**の行がコメント、それ以降にローバの測位解が 1 行に 1 エポック分、時間、座標、Q (品質)フラグ、有効衛星数、推定値標準偏差の形式で格納されます。このうち Q フラグの意味は以下の通りです。

1: FIX 解、**2**:FLOAT 解、**4**:DGPS 解、**5**:単独測位解

なおデフォルトでは 2 周波キネマティック、仰角マスク 10 度、緯度・経度・高度出力となっています。またデフォルトでは基準局座標として単独測位解の平均値を使用します。

表 1　測位解出力例

```
% program   : rnx2rtkp ver.1.0
% inputs    : rov0010.07o ref0010.07o ref0010.07n
% obs start : 2007/01/01 00:00:00.0 GPST (gpsweek1316 518400.0s)
% obs end   : 2007/01/01 23:59:30.0 GPST (gpsweek1316 604770.0s)
% mode/obsv : kinematic/L1+L2
% elev mask : 10.0 deg
% ref pos   : 35.132062716  139.624305669   72.3338
%
% (time=GPST, lat/lon/hight=WGS84/ellipsoidal, Q=1:fix,2:float,4:dgps,5:single,ns=# of
sats)
%  time     latitude(deg) longitude(deg)   hight(m)   Q  ns   sdn(m)   sde(m)   sdu(m)
 518400.000   35.160871612  139.613842087    66.8062   1   7   0.0072   0.0054   0.0164
 518430.000   35.160871607  139.613842115    66.7987   1   7   0.0072   0.0054   0.0164
 518460.000   35.160871593  139.613842110    66.7999   1   7   0.0072   0.0054   0.0163
 518490.000   35.160871583  139.613842093    66.8118   1   7   0.0072   0.0053   0.0163
 518520.000   35.160871627  139.613842143    66.8086   1   7   0.0072   0.0053   0.0163
 518550.000   35.160871616  139.613842127    66.8061   1   7   0.0072   0.0053   0.0162
 518580.000   35.160871596  139.613842108    66.8191   1   7   0.0072   0.0053   0.0162
 518610.000   35.160871586  139.613842086    66.8229   1   7   0.0072   0.0053   0.0161
 518640.000   35.160871615  139.613842077    66.8212   1   7   0.0072   0.0053   0.0161
 518670.000   35.160871628  139.613842154    66.8044   1   7   0.0072   0.0053   0.0160
...
```

(5) コマンドオプションを指定することにより、測位モード、周波数、仰角マスク、出力形式、基準局座標、等を変更することが出来ます。**RNX2KML** コマンドの詳細は付録 B を参照下さい。

3.4　POS2KML による測位解の変換

(1) 3.3 で出力したローバの位置軌跡を付属ユーティリティプログラム **pos2kml** を使うことにより Google Earth 上で表示できる KML ファイルに変換することができます。例えば 3.3 の例で出力した測位解**output.pos** を KML ファイルに変換するためには以下のコマンドを入力して下さい。デフォルトでは拡張子を**.kml** に変更したファイルに変換結果が出力されます。

```
> pos2kml output.pos
```

(2) **POS2KML** コマンドの詳細については付録 B を参照下さい。

3.5 RTKLIB ライブラリを使った RTK-GPS プログラムの開発

(1) RTKLIB ライブラリをユーザアプリケーションプログラムに組み込むことにより RTK-GPS プログラムを構築することができます。

(2) RTK-GPS プログラムを構築する場合、1 エポック分の観測データ、航法メッセージ入力コールバック関数 **input()**及び測位解出力コールバック関数 **output()**をユーザが記述する必要があります。各関数のプロトタイプについては付録 A **rtkpos()**関数の説明を参照してください。

(3) ユーザアプリケーションプログラム中で、コールバック関数 **input()**及び **output()**のポインタを引数にして RTKLIB の **rtkpos()**関数を呼び出して下さい。**rtkpos()**内部でまず **input()**を呼び出し、1 エポック分の観測データ及び航法メッセージを入力して、RTK-GPS 測位演算処理を行い、その結果を **output()**を呼び出して出力し、再度 **input()**を呼び出して次のエポックの処理を行うという処理ループに入ります。**rtkpos()**内部の処理ループから抜けるためにはコールバック関数 **input()**の戻り値として 0 以下の値を返してください。

(4) RTKLIB を使用した RTK-GPS プログラムの例を表 2 に示します。**rtkpos()**関数の詳細は付録 A を参照して下さい。

表 2　RTKLIB による RTK-GPS プログラム例

```
#include "rtklib.h"

/* 観測データ、航法メッセージ入力コールバック関数 */
int input(obsd_t * obs, nav_t *nav)
{
    int n;

    /* 1エポック分のローバ、基準局観測データ及び航法メッセージ読み込み処理 */

    ...

    return n; /* 観測データ個数、n<=0: 終了 */
}

/* 測位解出力コールバック関数 */
void output(gtime_t t, sol_t sol, sol_t solf, int stat, int ns)
{
    char str[64];
    time2str(t,str);
    printf("%s : %12.3f %12.3f %12.3f %d¥n",str,solf.rr[0],solf.rr[1],
           solf.rr[2],stat);
}

/* RTKプログラムメイン */
int main(void)
{
    /* reference position (ECEF) (m) */
    double rb[]={
        -3978241.958,3382840.234,3649900.853
    };

    /* RTK-GPS processing loop (exit if input() returns <=0) */
    rtkpos(input,output,2,2,rb,15.0*D2R,3.0,0);

    return 0;
}
```

A RTKLIB Functions

Function	Description	Reference
	Matrix and vector functions	
mat()	New martix	A.1
zeros()	New zero matrix	A.2
eye()	New identity marix	A.3
dot()	Inner Product	A.4
norm()	Euclid norm	A.5
matmul()	Multiply matrix	A.6
matinv()	Inverse of matic	A.7
solve()	Solve linear equation	A.8
lsg()	Least square estimation	A.9
filter()	Kalman filter state update	A.10
matprint()	Print matrix	A.11
	Time and string funtions	
str2num()	String to number	A.12
str2time()	String to time	A.13
epoch2time()	Calender day/time to time	A.14
time2epoch()	Time to calender day/time	A.15
gpst2time()	GPSTIME to time	A.16
time2gpst()	Time to GPSTIME	A.17
timeadd()	Add time	A.18
timediff()	Time difference	A.19
gpst2utc()	GPSTIME to UTC	A.20
utc2gpst()	UTC to GPSTIME	A.21
time2str()	Time to string	A.22
	Coordinates transformations	
ecef2pos()	ECEF to geodetic position	A.23
pos2ecef()	Geodetic to ECEF position	A.24
ecef2enu()	ECEF to local coordinates	A.25
enu2ecef()	Local to ECEF coordinates	A.26
covenu()	Covariance in local coordinates	A.27
geoidh()	Geoid height	A.28
	File Input/Output functions	
readrnx()	Read RINEX files	A.29
	OTF Integer Ambiguity resolution	
lambda()	LAMBDA/MLAMBDA integer least-square estimation	A.30
	Navigation functions	
eph2pos()	Satelilte ephemeris to satellite position/clock-bias	A.31
satpos()	Satellite positions/clock-biases	A.32
geodist()	Geometric distance	A.33
satazel()	Satellite azimuth/elevation angle	A.34
	Ionospheric/Tropospheric models	
ionmodel()	Ionospheric model	A.35
tropmodel()	Tropospheric model	A.36
	Positioning solutions	
pntpos()	Single point positioning	A.37
rtkpos()	RTK positioning	A.38

A.1 New Matrix

mat()

[Prototype]

```
#include "rtklib.h"
double *mat(int n, int m);
```

[Args]

int	n,m	I	Number of rows and columns of new matrix

[Return]

Matrix pointer (if `n<=0` or `m<=0`, return NULL)

[Description]

Allocate memory for new matrix.

[Notes]

If memory allocation error, print error message and exit program.

A.2 Zero Matrix

zeros()

[Prototype]

```
#include "rtklib.h"
double *zeros(int n, int m);
```

[Args]

int	n,m	I	Number of rows and columns of matrix

[Return]

Matrix pointer (if `n<=0` or `m<=0`, return NULL)

[Description]

Generate new zero matrix.

[Notes]

If memory allocation error, print error message and exit program.

A.3 New Identity Matrix

eye()

[Prototype]

```
#include "rtklib.h"
double *eye(int n);
```

[Args]

int	n	I	Number of rows and columns of matrix

[Return]

Matrix pointer (if `n<=0`, return NULL)

[Description]

Generate new identity matrix.

[Notes]

If memory allocation error, print error message and exit program.

A.4 Inner Product

dot()

[Prototype]

```
#include "rtklib.h"
double dot(const double *a, const double *b, int n);
```

[Args]

`double`	`a,b`	`I`	Vector ***a***, ***b*** (`n x 1`)
`int`	`n`	`I`	Size of vector ***a***, ***b***

[Return]

Inner product of vector ***a***, ***b*** ($\boldsymbol{a}^T\boldsymbol{b}$)

[Description]

Inner product of vectors.

[Notes]

A.5 Euclid Norm

norm()

[Prototype]

```
#include "rtklib.h"
double norm(const double *a, int n);
```

[Args]

`double *a`	`I`	vector $\boldsymbol{a}$ (`n x 1`)
`int n`	`I`	Size of vector $\boldsymbol{a}$

[Return]

Euclid norm of vector $\boldsymbol{a}$ ($\| \boldsymbol{a} \|$)

[Description]

Euclid norm of vector.

[Notes]

A.6 Multiply Matrix

matmul()

[Prototype]

```
#include "rtklib.h"
void matmul(const char *tr, int n, int k, int m, double alpha,
       const double *A, const double *B, double beta, double *C);
```

[Args]

`char`	`*tr`	`I`	Transpose flags of matrix ***A***, ***B*** (`"N"`: normal,`"T"`: transpose)
`int`	`n,k,m`	`I`	Size of (transposed) matrix ***A***, ***B***
`double`	`alpha`	`I`	*alpha*
`double`	`*A,*B`	`I`	(Transposed) Matrix ***A*** (`n x m`), ***B*** (`m x k`)
`double`	`beta`	`I`	*beta*
`double`	`*C`	`IO`	Matrix ***C*** (`n x k`)

[Return]

None

[Description]

Multiply matrix by matrix. (***C***=*alpha****A*******B****+*beta****C***)

[Notes]

Lapper of BLAS DGEMM.

A.7 Inverse of Matrix

`matinv()`

[Prototype]

```
#include "rtklib.h"
int matinv(double *A, int n);
```

[Args]

`double`	`*A`	`IO`	Matrix ***A*** (`n x n`)
`int`	`n`	`I`	Size of matrix ***A***

[Return]

Status (`0`:OK, `0>`:Error)

[Description]

Inverse of matrix ($\boldsymbol{A}=\boldsymbol{A}^{-1}$).

[Notes]

A.8 Solve Linear Equation

solve()

[Prototype]

```
#include "rtklib.h"
int solve(const char *tr, const double *A, const double *Y, int n, int m,
          double *X);
```

[Args]

`char`	`*tr`	`I`	Transpose flag of matrix $\boldsymbol{A}$ (`"N"`: normal,`"T"`: transpose)
`double`	`*A`	`I`	Matrix $\boldsymbol{A}$ (`n x n`)
`double`	`*Y`	`I`	Matrix $\boldsymbol{Y}$ (`n x m`)
`int`	`n,m`	`I`	Size of matrix $\boldsymbol{Y}$, $\boldsymbol{X}$
`double`	`*X`	`O`	Solution $\boldsymbol{X}$ of linear equation $\boldsymbol{AX}=\boldsymbol{Y}$ or $\boldsymbol{A}^T\boldsymbol{X}=\boldsymbol{Y}$ (`n x m`)

[Return]

Status (0:OK, 0>:Error)

[Description]

Solve linear equation ($\boldsymbol{AX}=\boldsymbol{Y}$ or $\boldsymbol{A}^T\boldsymbol{X}=\boldsymbol{Y}$).

[Notes]

Matirix stored by column-major order (FORTRAN convention).

`X` can be same as `Y`.

A.9 Least Square Estimation

lsq()

[Prototype]

```
#include "rtklib.h"
int lsq(const double *A, const double *y, int n, int m, double *x,
        double *Q);
```

[Args]

`double *A`		I	Transpose of (weighted) design matrix $\boldsymbol{A}$ (`n x m`)
`double *y`		I	(Weighted) Measurements $\boldsymbol{y}$ (`m x 1`)
`int`	`n,m`	I	Number of parameters and measurements (`n<=m`)
`double *x`		O	Estimated parameters $\boldsymbol{x}$ (`n x 1`)
`double *Q`		O	Esimated parameters covariance matrix $\boldsymbol{Q}_x$ (`n x n`)

[Return]

Status (`0`:OK, `0>`:error)

[Description]

Least square estimation by solving normal equation ($\boldsymbol{x}=(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{y}$).

[Notes]

For weighted least square, replace $\boldsymbol{A}$ and $\boldsymbol{y}$ to $\boldsymbol{A}\boldsymbol{w}$ and $\boldsymbol{w}\boldsymbol{y}$ ($\boldsymbol{w}=\boldsymbol{W}^{1/2}$).

Matirix stored by column-major order (FORTRAN convention).

A.10 Kalman Filter State Update

filter()

[Prototype]

```
#include "rtklib.h"
int filter(const double *x, const double *P, double *H, double *v,
           const double *R, int n, int m, double *xp, double *Pp);
```

[Args]

`double *x`		I	States vector $\boldsymbol{x}$ (`n x 1`)
`double *P`		I	Covariance matrix of states $\boldsymbol{P}$ (`n x n`)
`double *H`		I	Transpose of design matrix $\boldsymbol{H}$ (`n x m`)
`double *v`		I	Innovation (measurement - model) $\boldsymbol{v}$ (`m x 1`)
`double *R`		I	Covariance matrix of measurement error $\boldsymbol{R}$ (`m x m`)
`int`	`n,m`	I	Number of states and measurements
`double *xp`		O	Updated states vector $\boldsymbol{x}^+$ (`n x 1`)
`double *Pp`		O	Updated covariance matrix of states $\boldsymbol{P}^+$ (`n x n`)

[Return]

Status (`0`:OK,`0>`:Error)

[Description]

Kalman filter states update by measurements ($\boldsymbol{K}=\boldsymbol{PH}\ (\boldsymbol{H}^T\boldsymbol{PH}+\boldsymbol{R})^{-1}$, $\boldsymbol{x}^+=\boldsymbol{x}+\boldsymbol{Kv}$, $\boldsymbol{P}^+=(\boldsymbol{I}-\boldsymbol{KH}^T)\boldsymbol{P}$).

[Notes]

Matirix stored by column-major order (FORTRAN convention).

A.11 Print Matrix

matprint()

[Prototype]

```
#include "rtklib.h"
void matprint(const double *A, int n, int m, int p, int q);
```

[Args]

`double`	`*A`	I	Matrix ***A*** (`n x m`)
`int`	`n,m`	I	Number of rows and columns of ***A***
`int`	`p,q`	I	Print format total columns and columns under decimal point.

[Return]

None

[Description]

Print matrix to STDOUT.

[Notes]

Matirix stored by column-major order (FORTRAN convention).

A.12 String to Number

`str2num()`

[Prototype]

```
#include "rtklib.h"
double str2num(const char *s, int i, int n);
```

[Args]

`char`	`*s`	`I`	String (`"... nnn.nnn ..."`)
`int`	`i,n`	`I`	Substring position and width.

[Return]

Converted number (`0.0`: Error)

[Description]

Convert substring in string to number.

[Notes]

A.13 String to Time

str2time()

[Prototype]

```
#include "rtklib.h"
int str2time(const char *s, int i, int n, gtime_t *t);
```

[Args]

`char`	`*s`	I	String (`"... yyyy mm dd hh mm ss ..."`)
`int`	`i,n`	I	Substring position and width
`gtime_t`	`*t`	O	RTKLIB time struct

[Return]

Status (`0`:OK,`0>`:Error)

[Description]

Convert substring in string to RTKLIB time struct.

[Notes]

A.14 Calendar Day/Time to Time

epoch2time()

[Prototype]

```
#include "rtklib.h"
gtime_t epoch2time(const double *ep);
```

[Args]

`double *ep` I day/time `{year,month,day,hour,min,sec}`

[Return]

RTKLIB time struct.

[Description]

Convert calendar day/time to RTKLIB time struct.

[Notes]

A.15 Time to Calendar Day/Time

time2epoch()

[Prototype]

```
#include "rtklib.h"
void time2epoch(gtime_t t, double *ep);
```

[Args]

`gtime_t t`	`I`	RTKLIB time struct
`double *ep`	`O`	Day/time `{year,month,day,hour,min,sec}`

[Return]

None

[Description]

Convert RTKLIB time struct to calendar day/time

[Notes]

A.16 GPSTIME to Time

gpst2time()

[Prototype]

```
#include "rtklib.h"
gtime_t gpst2time(int week, double sec);
```

[Args]

int	week	I	GPS week number
double	sec	I	GPSTIME (time of week) (sec)

[Return]

RTKLIB time struct.

[Description]

Convert GPS week and time of week to RTKLIB time struct.

[Notes]

A.17 Time to GPSTIME

time2gpst()

[Prototype]

```
#include "rtklib.h"
double time2gpst(gtime_t t, int *week);
```

[Args]

`gtime_t`	`t`	`I`	RTKLIB time struct
`int`	`*week`	`O`	GPS week number

[Return]

GPSTIME (time of week) (sec)

[Description]

Convert RTKLIB time struct to GPS week and time of week.

[Notes]

A.18 Add Time

timeadd()

[Prototype]

```
#include "rtklib.h"
gtime_t timeadd(gtime_t t, double sec);
```

[Args]

`gtime_t t`	`I`	RTKLIB time struct.
`double sec`	`I`	Time to add (sec)

[Return]

RTKLIB time struct (`t+sec`)

[Description]

Add time to RTKLIB time struct.

[Notes]

A.19 Time Difference

timediff()

[Prototype]

```
#include "rtklib.h"
double timediff(gtime_t t1, gtime_t t2);
```

[Args]

`gtime_t t1,t2 I` RTKLIB time structs

[Return]

Time difference (`t1-t2`) (sec)

[Description]

Difference between RTKLIB time struct.

[Notes]

A.20 GPSTIME to UTC

gpst2utc()

[Prototype]

```
#include "rtklib.h"
gtime_t gpst2utc(gtime_t t);
```

[Args]

gtime_t t	I	Time expressed in GPSTIME

[Return]

Time expressed in UTC.

[Description]

Convert time expressed in GPSTIME to UTC considering leap seconds

[Notes]

A.21 UTC to GPSTIME

utc2gpst()

[Prototype]

```
#include "rtklib.h"
gtime_t utc2gpst(gtime_t t);
```

[Args]

gtime_t t	I	Time expressed in UTC

[Return]

Time expressed in GPSTIME.

[Description]

Convert time expressed in UTC to GPSTIME considering leap seconds.

[Notes]

A.22 Time to String

time2str()

[Prototype]

```
#include "rtklib.h"
void time2str(gtime_t t, char *s, int n);
```

[Args]

`gtime_t t`		I	RTKLIB time struct
`char`	`*s`	O	String in the form of `"yyyy/mm/dd hh:mm:ss.s..."`
`int`	`n`	I	Columns of sec under decimal point

[Return]

None

[Description]

Convert RTKLIB time struct to string.

[Notes]

A.23 ECEF to Geodetic Position

ecef2pos()

[Prototype]

```
#include "rtklib.h"
void ecef2pos(const double *r, double *pos);
```

[Args]

`double *r`	`I`	ECEF position `{x,y,z}` (m)
`double *pos`	`O`	Geodetic position `{lat,lon,h}` (rad,m)

[Return]

None

[Description]

Transform ECEF position to geodetic position (latitulde, longitude and height).

[Notes]

WGS84, ellipsoidal height.

A.24 Geodetic to ECEF Position

pos2ecef()

[Prototype]

```
#include "rtklib.h"
void pos2ecef(const double *pos, double *r);
```

[Args]

`double *pos`	`I`	Geodetic position `{lat,lon,h}` (rad,m)
`double *r`	`O`	ECEF position `{x,y,z}` (m)

[Return]

None

[Description]

Transform geodetic position (latitude, longitude and height) to ECEF position.

[Notes]

WGS84, ellipsoidal height.

A.25 ECEF to Local Coordinates

ecef2enu()

[Prototype]

```
#include "rtklib.h"
void ecef2enu(const double *pos, const double *r, double *e);
```

[Args]

`double *pos`	`I`	Geodetic position `{lat,lon}` (rad)
`double *r`	`I`	Vector in ECEF coordinates `{x,y,z}`
`double *e`	`O`	Vector in local tangental coordinates `{e,n,u}`

[Return]

None

[Description]

Transform ECEF vector to local tangental coordinates.

[Notes]

A.26 Local to ECEF Coordinates

enu2ecef()

[Prototype]

```
#include "rtklib.h"
void enu2ecef(const double *pos, const double *e, double *r);
```

[Args]

`double *pos`	`I`	Geodetic position `{lat,lon}` (rad)
`double *e`	`I`	Vector in local tangental coordinates `{e,n,u}`
`double *r`	`O`	Vector in ECEF coordinates `{x,y,z}`

[Return]

None

[Description]

Transform local tangental coordinates vector to ECEF.

[Notes]

A.27 Covariance in Local Coordinates

covenu()

[Prototype]

```
#include "rtklib.h"
void covenu(const double *pos, const double *P, double *Q);
```

[Args]

`double *pos`	`I`	Geodetic position `{lat,lon}` (rad)
`double *P`	`I`	Covariance in ECEF coordinates
`double *Q`	`O`	Covariance in local tangental coordinates

[Return]

None

[Description]

Transform ECEF covariance to local tangental coordinates.

[Notes]

A.28 Geoid Height

geoidh()

[Prototype]

```
#include "rtklib.h"
double geoidh(const double *pos);
```

[Args]

`double *pos` `I` Geodetic position `{lat,lon}` (rad)

[Return]

Geoid height (m) (`0.0`: Error/out of range)

[Description]

Get geoid height from geoid model.

[Notes]

Geoid model is derived from EGM96. Only supports Japan area (longitude=120-155deg, latitude=20-50 deg).

A.29 Read RINEX Files

readrnx()

[Prototype]

```
#include "rtklib.h"
int readrnx(char **files, int n, obs_t *obs, nav_t *nav);
```

[Args]

char	**files	I	RINEX files
int	n	I	Number of RINEX files (0: Input from STDIN)
obs_t	*obs	O	Observation data
nav_t	*nav	O	Navigation messages

[Return]

Number of epochs (0: No data)

[Description]

Read RINEX observation data (OBS) and navigation message (NAV) files.

[Notes]

Observation data are sorted by time, receiver and satellite number. Navigation messages are sorted by TOE. Dupulicated ephemerides are deleted. Observation data type `obs_t` and Navigation message type `nav_t` are defined as follows. Observation data receiver number `obs->data[i].rcv` is started as 1 and inclemented observation data file by file. Supports RINEX 2.10 but only GPS.

```
typedef struct {        /* observation data record */
    gtime_t time;       /* receiver sampling time */
    int sat,rcv;/* satellite/receiver number */
    double L[NFREQ];    /* observation data carrier-phase (cycle) */
    double P[NFREQ];    /* observation data pseudorange (m) */
    short LLI[NFREQ];   /* loss of lock indicator */
} obsd_t;

typedef struct {        /* observation data */
    int n;              /* number of obervation data records */
    obsd_t *data;       /* obervation data records */
} obs_t;

typedef struct {        /* satellite ephemeris and clock parameters */
    int sat;            /* satellite number */
    int iode,iodc;      /* IODE,IODC */
    int sva,svh;        /* sv accuracy, sv health */
```

```
    gtime_t toe,toc,ttr; /* Toe,Toc,T_trans */
    double A,e,i0,OMG0,omg,M0,deln,OMGd,idot, crc,crs,cuc,cus,cic,
           cis,toes;    /* sv ephemeris parameters */
    double f0,f1,f2,tgd; /* sv clock parameters */
} eph_t;

typedef struct {        /* navigation messages */
    int n;              /* number of ephemeris and clock parameters */
    eph_t *eph;         /* satellite ephemeris and clock parameters */
    double ion[8];      /* iono model params {a0,a1,a2,a3,b0,b1,b2,b3} */
    double utc[4];      /* delta-utc parameters */
} nav_t;
```

A.30 LAMBDA/MLAMBDA Integer Least-Square Estimation

lambda()

[Prototype]

```
#include "rtklib.h"
int lambda(int n, int m, const double *a, const double *Q, double *F,
           double *s);
```

[Args]

int	n	I	Number of float parameters
int	m	I	Number of fixed solutions
double *a		I	Float parameters (n x 1)
double *Q		I	Covariance matrix of float parameters (n x n)
double *F		O	Fixed solutions (n x m)
double *s		O	Sum of squared residulas of fixed solutions (1 x m)

[Return]

Status (0:OK, other:Error)

[Description]

Integer least-square estimation.

Reduction is performed by LAMBDA (ref.[1]), and search by MLAMBDA (ref.[2]).

[Notes]

Matrix stored by column-major order (FORTRAN convension).

References

[1] P.J.G.Teunissen, The least-square ambiguity decorrelation adjustment: a method for fast GPS ambiguity estimation, *J.Geodesy*, Vol.70, 65-82, 1995

[2] X.-W.Chang, X.Yang, T.Zhou, MLAMBDA: A modified LAMBDA method for integer least-squares estimation, *J.Geodesy*, Vol.79, 552-565, 2005

A.31 Satellite Ephemeris to Satellite Position/Clock-bias

eph2pos()

[Prototype]

```
#include "rtklib.h"
void eph2pos(gtime_t t, const eph_t *eph, double pr, double *rs,
             double *dts);
```

[Args]

`gtime_t t`	`I`	Time (GPSTIME)
`eph_t *eph`	`I`	Satellite ephemeris and clock parameter
`double pr`	`I`	Pseudorange measurement (m)
`double *rs`	`O`	Satellilte ECEF position at signal transmission `{x,y,z}` (m)
`double *dts`	`O`	Satellilte clock-bias at signal transmission (sec)

[Return]

None

[Description]

Compute satellite position and clock-bias at signal transmission from satellite ephemeris and clock parameters.

[Notes]

Satellite position is expressed in ECEF coordinates at signal reception time.

For type definition of `eph_t` see `readrnx()`.

A.32 Satellite Positions/Clock-biases

satpos()

[Prototype]

```
#include "rtklib.h"
void satpos(const obsd_t *obs, int n, const nav_t *nav, double *rs,
            double *dts);
```

[Args]

`obsd_t`	`*obs`	`I`	Observation data records
`int`	`n`	`I`	Number of observation data records
`nav_t`	`*nav`	`I`	Navigation messages
`double`	`*rs`	`O`	Satellilte ECEF positions at signal transmission `{x,y,z}` (m)
`double`	`*dts`	`O`	Satellilte clock-biases at signal transmission (sec)

[Return]

None

[Description]

Compute satellite positions and clock-biases at signal transmission from navigation messages.

[Notes]

`rs[(0:2)+i*3],dts[i]` = `obs[i]` satellite position/clock-bias.

If no ephemeris and clock parameter, set `0.0` to `rs[]`, `dts[]`.

Satellite positions are expressed in ECEF coordinates at signal reception time.

For type definition of `obsd_t` and `nav_t`, see `readrnx()`.

A.33 Geometric Distance

geodist()

[Prototype]

```
#include "rtklib.h"
double geodist(const double *rs, const double *rr, double *e);
```

[Args]

`double *rs`	`I`	Satellilte ECEF position at signal transmition `{x,y,z}` (m)
`double *rr`	`I`	Receiver ECEF position `{x,y,z}` (m)
`double *e`	`O`	Receiver-to-satellilte unit vector `{x,y,z}`

[Return]

Geometric distance (m) (0>: Error/no satellite position)

[Description]

Compute geometric distance between satellite and receiver, and receiver-to-satellite unit vector.

[Notes]

Satellite position and receiver-to-satellite unit vector are expressed in ECEF coordinates at signal reception time.

A.34 Satellite Azimuth/Elevation Angle

satazel()

[Prototype]

```
#include "rtklib.h"
void satazel(const double *pos, const double *e, double *azel);
```

[Args]

`double *pos`	`I`	Receiver geodetic position `{lat,lon,h}` (rad, m)
`double *e`	`I`	Receiver-to-satellilte unit vevtor `{x,y,z}`
`double *azel`	`O`	Satellite Azimuth/elevation angle `{az,el}` (rad)

[Return]

None

[Description]

Compute satellite azimuth/elevation angle from receiver.

[Notes]

Receiver-to-satellite unit vector are expressed in ECEF coordinates at signal reception time.

A.35 Ionospheric Model

ionmodel()

[Prototype]

```
#include "rtklib.h"
double ionmodel(gtime_t t, const double *ion, const double *pos,
                const double *azel);
```

[Args]

`gtime_t t`	I	RTKLIB time struct
`double *ion`	I	Ionospheric model parameters `{a0,a1,a2,a3,b0,b1,b2,b3}`
`double *pos`	I	Receiver geodetic position `{lat,lon,h}` (rad, m)
`double *azel`	I	Satellite azimuth/elevation angle `{az,el}` (rad)

[Return]

Ionospheric delay (L1) (m)

[Description]

Compute ionospheric delay by broadcast ionosphere model (Klobuchar model).

[Notes]

A.36 Tropospheric Model

tropmodel()

[Prototype]

```
#include "rtklib.h"
double tropmodel(const double *pos, const double *azel);
```

[Args]

`double *pos`	I	Receiver geodetic position `{lat,lon,h}` (rad,m)
`double *azel`	I	Satellite azimuth/elevation angle `{az,el}` (rad)

[Return]

Tropospheric delay (m)

[Description]

Compute tropospheric delay by Standard Atmosphere and Saastamoinen model.

[Notes]

A.37 Single Point Positioning

pntpos()

[Prototype]

```
#include "rtklib.h"
int pntpos(const obsd_t *obs, int n, const double *rs, const double *dts,
           const double *ion, double elmin, double *rr, double *Qr,
           double *dtr, double *azel);
```

[Args]

`obsd_t`	`*obs`	I	Observation data records for an epoch
`int`	`n`	I	Number of observation data records
`double`	`*rs`	I	Satellilte positions at signal transmission `{x,y,z}` (m)
`double`	`*dts`	I	Satellilte clock-biases at signal transmission (sec)
`double`	`*ion`	I	Ionosphere model parameters
`double`	`elmin`	I	Elevation cutoff angle (rad)
`double`	`*rr`	O	Estimated receiver position `{x,y,z}` (m)
`double`	`*Qr`	O	Estimated receiver position covarience (3 x 3)
`double`	`*dtr`	O	Estimated receiver clock-bias (sec)
`double`	`*azel`	O	Satellite azimuth/elevation `{az,el,...}` (rad)

[Return]

Number of valid satellites (`0>`: Error)

[Description]

Compute receiver position and clock-bias by Single Point Positioning.

[Notes]

`rs[(0:2)+i*3],dts[i]` = `obs[i]` Satellite position/clock-bias

`azel[(0:1)+i*2]` = `obs[i]` Satellite azimuth/elevation angle

Satellite positions are expressed in ECEF coordinates at signal reception time.

For type definition of `obs_t` see `readrnx()`.

A.38 RTK positioning

rtkpos()

[Prototype]

```
#include "rtklib.h"
void rtkpos(infunc_t input, outfunc_t output, int mode, int nf,
            const double *rb, double elmin, double thres, int opt);
```

[Args]

`infunc_t`	`input`	I	Input callback function
`outfunc_t`	`output`	I	Output callback function
`int`	`mode`	I	Positioning Mode (`0`: Single, `1`: DGPS, `2`: Kinematic, `3`: Static)
`int`	`nf`	I	Number of frequencies (`1`: L1, `2`: L1+L2)
`double`	`*rb`	I	Reference (base) station position (ECEF) `{x,y,z}` (m)
`double`	`elmin`	I	Elevation mask angle (rad)
`double`	`thres`	I	Validation threshold of ambiguity (`1>`: No AR)
`int`	`opt`	I	Options (`1`: Instantaneous AR)

[Return]

None

[Description]

Input observation data and navigation message, compute rover position and output solutions. If callback function `input()` return 0, exit the function.

[Notes]

Input and output callback functions shall comply the following prototypes.

```
int input(obsd_t *obs, nav_t *nav);
```

`obsd_t`	`*obs`	I	Observation data records for an epoch `obs[i].rcv`=`1`: Rover, `2`: Reference station, sorted by receivers and satellites
`nav_t`	`*nav`	I	Navigation messages
`return`			Number of observation data records

```
void output(gtime_t t, sol_t sol, sol_t solf, int stat, int ns);
```

gtime_t	t	I	Time (Receiver sampling GPSTIME)
sol_t	sol	I	FLOAT, DGPS, or SINGLE solution
sol_t	solf	I	FIX solution
int	stat	I	Solution status (1: FIX, 2: FLOAT, 4: DGPS, 5: SINGLE)
int	ns	I	Number of valid satellites

The type `sol_t` is defined as follows. For type definition of `obs_t` and `nav_t`, see `readrnx()`.

```
typedef struct {       /* positioning solution */
   double rr[3];      /* estimated receiver position (ecef) (m) */
   double Qr[9];      /* estimated receiver positoin covariance */
} sol_t;
```

B Commands

Command	Description	Reference
rnx2rtkp	Baseline analysis by precise relative positioning	B.1
pos2kml	Convert positions to Google Earth KML file	B.2

B.1 Baseline Analysis by Precise Relative Positioning

rnx2rtkp

SYNOPSIS

```
rnx2rtkp [option ...] file file [...]
```

DESCRIPTION

Read RINEX OBS/NAV files, compute receiver (rover) positions and output position solutions. The first RINEX OBS file shall contain receiver (rover) observations. For the relative mode, the second RINEX OBS file shall contain reference (base) receiver observations. At least one RINEX NAV file shall be included in input files. Command options are as follows. ([]:default).

OPTIONS

`-h`	Print help
`-o output`	Output file [STDOUT]
`-p mode`	Positioning Mode (`0`: SINGLE, `1`: DGPS, `2`: KINEMATIC, `3`: STATIC) [`2`]
`-m mask`	Elevation mask angle (deg) [`10`]
`-f freq`	Number of frequencies for relative mode (`1`: L1, `2`: L1+L2) [`2`]
`-v thres`	Validation threshold for Integer Ambiguity Resolution (`0`: No AR) [`3`]
`-b`	Backward analysis solutions [off]
`-c`	Forward/backward combined solutions [off]
`-i`	Instantaneous Integer Ambiguity Resolution [off]
`-e`	Output X/Y/Z-ECEF position [Latitude/Longitude/Height]
`-n`	Output NMEA-0183 GGA sentence [off]
`-g`	Output latitude/longitude in the form of `ddd mm ss.ss` [`ddd.ddd`]
`-t`	Output time in the form of `yyyy/mm/dd hh:mm:ss.ss` [`ssssss.ss`]
`-u`	Output time in UTC [GPST]
`-d col`	Columns of time under decimal point [`3`]
`-s sep`	Field separator [`' '`]
`-r x y z`	Reference (base) receiver ECEF position (m) [Average of single pos]

EXAMPLES

Example 1. Kinematic Positioning, L1+L2, output Latitude/Longitude/Height to STDOUT.

command

```
> rnx2rtkp 07590920.05o 30400920.05o 30400920.05n
```

result

```
% program   : rnx2rtkp ver.1.0
% inputs    : 07590920.05o 30400920.05o 30400920.05n
% obs start : 2005/04/02 00:00:00.0 GPST (gpsweek1316 518400.0s)
% obs end   : 2005/04/02 23:59:30.0 GPST (gpsweek1316 604770.0s)
% mode/obsv : kinematic/L1+L2
% elev mask : 10.0 deg
% ref pos   : 35.132062716  139.624305669   72.3338
%
% (time=GPST, lat/lon/hight=WGS84/ellipsoidal, Q=1:fix,2:float,4:dgps,5:single, ns=# of
sats)
%  time     latitude(deg) longitude(deg)   hight(m)   Q  ns   sdn(m)   sde(m)   sdu(m)
 518400.000   35.160871612  139.613842087    66.8062   1   7   0.0072   0.0054   0.0164
 518430.000   35.160871607  139.613842115    66.7987   1   7   0.0072   0.0054   0.0164
 518460.000   35.160871593  139.613842110    66.7999   1   7   0.0072   0.0054   0.0163
 518490.000   35.160871583  139.613842093    66.8118   1   7   0.0072   0.0053   0.0163
...
```

Example 2. Single Point Positioning, El Mask=15deg, output NMEA GGA to file `out.pos`

command

```
> rnx2rtkp -p 0 -m 15 -n -o out.pos 07590920.05o 30400920.05n
```

result

```
$GPGGA,235947.00,35 9.6524150,N,13936.8296671,E,1,07,,34.318,M,36.181,M,,*42
$GPGGA,000017.00,35 9.6525341,N,13936.8298278,E,1,07,,33.808,M,36.181,M,,*47
$GPGGA,000047.00,35 9.6524354,N,13936.8299014,E,1,07,,33.717,M,36.181,M,,*4F
$GPGGA,000117.00,35 9.6522549,N,13936.8298201,E,1,07,,34.418,M,36.181,M,,*4B
$GPGGA,000147.00,35 9.6522543,N,13936.8298643,E,1,07,,33.317,M,36.181,M,,*49
$GPGGA,000217.00,35 9.6523911,N,13936.8296580,E,1,07,,34.406,M,36.181,M,,*47
$GPGGA,000247.00,35 9.6524503,N,13936.8299040,E,1,07,,34.230,M,36.181,M,,*4F
$GPGGA,000317.00,35 9.6523647,N,13936.8300515,E,1,07,,33.780,M,36.181,M,,*42
...
```

Example 3. Static Positioning, L1, time form `yyyy/mm/dd hh:mm:ss`, output X/Y/Z-ECEF positions

command

```
> rnx2rtkp -p 3 -f 1 -t -e 07590920.05o 30400920.05o 30400920.05n
```

result

```
% program   : rnx2rtkp ver.1.0
% inputs    : 07590920.05o 30400920.05o 30400920.05n
% obs start : 2005/04/02 00:00:00.0 GPST (gpsweek1316 518400.0s)
% obs end   : 2005/04/02 23:59:30.0 GPST (gpsweek1316 604770.0s)
% mode/obsv : static/L1
% elev mask : 10.0 deg
% ref pos   : -3978240.6491   3382839.2297   3649900.4598
%
% (time=GPST, x/y/z-ecef=WGS84, Q=1:fix,2:float,4:dgps,5:single, ns=# of sats)
%  time                   x-ecef(m)     y-ecef(m)     z-ecef(m)   Q  ns   sdx(m)   sdy(m)
sdz(m)
2005/04/02 00:00:00.000  -3976217.9351   3382371.4458   3652511.3843   2   7   1.2124
1.3748   1.0970
2005/04/02 00:00:30.000  -3976217.8724   3382370.5977   3652510.7455   1   7   0.0108
0.0120   0.0094
2005/04/02 00:01:00.000  -3976217.8733   3382370.5987   3652510.7454   1   7   0.0088
0.0098   0.0077
...
```

Example 4. Kinematic Positioning, Instantaneous AR, validation threshold=2, comma separator

command

```
> rnx2rtkp -i -v 2 -s , 07590920.05o 30400920.05o 30400920.05n
```

result

```
% program   : rnx2rtkp ver.1.0
% inputs    : 07590920.05o 30400920.05o 30400920.05n
% obs start : 2005/04/02 00:00:00.0 GPST (gpsweek1316 518400.0s)
% obs end   : 2005/04/02 23:59:30.0 GPST (gpsweek1316 604770.0s)
% mode/obsv : kinematic/L1+L2/instantaneous AR
% elev mask : 10.0 deg
% ref pos   : 35.132062716, 139.624305669,   72.3338
%
% (time=GPST, lat/lon/hight=WGS84/ellipsoidal, Q=1:fix,2:float,4:dgps,5:single, ns=# of
sats)
%  time     latitude(deg) longitude(deg)   hight(m)   Q  ns   sdn(m)   sde(m)   sdu(m)
 518400.000,  35.160871612, 139.613842087,   66.8062,  1,  7,  0.0072,  0.0054,  0.0164
 518430.000,  35.160871607, 139.613842115,   66.7987,  1,  7,  0.0072,  0.0054,  0.0164
 518460.000,  35.160871593, 139.613842110,   66.7999,  1,  7,  0.0072,  0.0054,  0.0163
 518490.000,  35.160871583, 139.613842093,   66.8118,  1,  7,  0.0072,  0.0053,  0.0163
 518520.000,  35.160871627, 139.613842143,   66.8086,  1,  7,  0.0072,  0.0053,  0.0163
...
```

B.2 Convert Positions to Google Earth KML file

pos2kml

SYNOPSIS

```
pos2kml [option ...] file [...]
```

DESCRIPTION

Read position file(s) and convert it to Google Earth KML file. Each line in the input file shall contain fields of time, position fields (Latitude/Longitude/Height or X/Y/Z-ECEF), and Quality flag (option). The line started with `'%'`, `'#'`, `';'` is treated as comment. Command options are as follows. ([]:default)

OPTIONS

`-h`	Print help
`-o file`	Output file [input file `+.kml`]
`-c color`	Track color (`0`:Off, `1`:White, `2`:Red, `3`:Orange, `4`:Green, `5`:Yellow) [`5`]
`-p color`	Point color (`0`:Off, `1`:White, `2`:Red, `3`:Orange, `4`:Green, `5`:by Quality Flag) [`5`]
`-a`	Output altitude information [off]
`-f n e h`	Add North/East/Height offsets to position (m) [0 0 0]
`-e`	Input X/Y/Z-ECEF position [Latitude/Longitude/Height]
`-g`	Input Latitude/longitude in the form of `ddd mm ss.ss` [`ddd.ddd`]
`-s sep`	Field separator [`' '`]