鈴鹿経営科学研究所

Basic Toolkit パッケージ

Version: 1.33

1. 本書について

本書は鈴鹿経営科学研究所(SMSL)が提供する RTMaps パッケージの使用方法説明書です。

パッケージ名: SMSL_BasicToolkit.pck

著者: 伊田 祥吾 / 鈴鹿経営科学研究所 所長

問い合わせ先 email: support@suzuka-msl.jp

2. 免責事項

本書および関連する RTMaps パッケージは、そのままの姿で提供され、一切の明示的または黙示的な保証はされません。本パッケージに含まれるコンポーネントの正確性と機能を保証するためにあらゆる努力をしておりますが、著者および配布者は、本ソフトウェアの使用または誤用に起因する直接的、間接的、偶発的、または結果的な損害について一切の責任を負いません。

このソフトウェアは RTMaps（Real-Time Multisensor applications）環境で動作するように設計されており、特定のハードウェア、ドライバ、またはサードパーティ製ライブラリに依存する場合があります。安全性が重視されるシステムや実稼働システムに本ソフトウェアを導入する前に、システムの互換性を確認し、適切なテストを実施することは使用者の責任となります。

性能は基盤となるシステム構成によって異なり、リアルタイムシステムにおける使用には、導入する組織による適切な検証と認証が必要であることにご注意ください。

このパッケージにはサードパーティ製のコンポーネントが含まれている場合や、サードパーティ製のコンポーネントとインターフェース接続している場合があります。使用者は、関連するサードパーティ製ソフトウェアのライセンス条項を遵守する必要があります。

本パッケージに含まれるオープンソースソフトウェア（GeographicLib、YDLidar-SDK）とそのライセンスは、パッケージに同梱の Release フォルダの LICENSE.txt に記載しています。

本出版物およびその内容は、予告なく変更される場合があります。

RTMaps は Intempora S.A.S.の登録商標です。

その他のブランド名または製品名は、それぞれの会社または組織の商標または登録商標です。

Table of contents

1. 本書について 2

2. 免責事項 3

3. SMSL_BasicToolkit パッケージ 6

3.1 パッケージ概要 6

3.2 ジェネレータ コンポーネント 7

SineCosineGenerator 7

GaussianNoiseGenerator 8

StaticImageGenerator 11

3.3 GNSS コンポーネント 13

GeoDistance 13

NMEA_GNS_Decoder 16

NMEA_RMC_Decoder 19

3.4 センサードライバコンポーネント 22

YDLidarX2_Driver 22

WitMotion_WT901B 25

Hi_Link_LD2410C 28

3.5 信号処理コンポーネント 32

UnitDelay 32

SkipEarlyFrames 36

SampleHistoryVectorizer 38

MaxMinMeanforArray 40

MaxMinMeanforMultipleSignals 42

NumericalTextWriter 44
bits2Byte 45
Byte2bits 46
imu_to_pose6D 47
imu_orientation_estimator 49
3.6 ハードウェア仕様コンポーネント 53
RobotSpecManager 53
4. マクロコンポーネント 56
SineCosineGenMacro 57
GaussianNoiseGenMacro 59
PresenceAlarm 61
5. デモダイヤグラム 63
DSB_AM_demo 63
MaxMinMean_demo 65
GeoDistance_demo 66
YDLidarDataAcquisition_demo 67
WitMotion_WT901B_imu_demo 67
Hi_Link_LD2410C_demo 68
6. 結論 70
7. 更新履歴 70

3. SMSL_BasicToolkit パッケージ

3.1 パッケージ概要

本パッケージには、SMSL により製作された RTMaps4で動作する基本的なアプリケーションコンポーネントが含まれます。SMSL は本パッケージで包括的な信号発生器、GNSS データ処理機能、およびいくつかの信号処理機能を提供します。

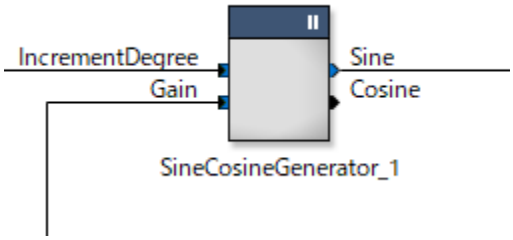
本パッケージは以下の OS で評価されています:

- Windows11 64bit 版.
- Ubuntu 24.04 LTS 64bit 版

3.2 ジェネレータ コンポーネント

これらのジェネレータコンポーネントは、カルマンフィルタコンポーネントの検証、位相信号を生成するのに有用です。

SineCosineGenerator



可変ゲイン・初期角度・DC オフセット付きの正弦波と余弦波（位相差 90°）を生成します。

処理内容

- IncrementDegree のサンプルごとに内部角度を進めます: θ += IncrementDegree [deg]（初期値 InitAngleBias）。

- Sine = Gain·sin(θ) + DC_bias、 Cosine = Gain·cos(θ) + DC_bias。

使い方

- 主な入力: number_generator.o_float64 → IncrementDegree、 number_generator.o_float64 → Gain、 SineCosineGenerator.Sine → Gain。

- 主な出力: Sine → Oscilloscope.inputA、 Sine → SineCosineGenerator.Gain。

データ形式

- 入力: スカラー。出力: Float64 スカラー。

注意事項

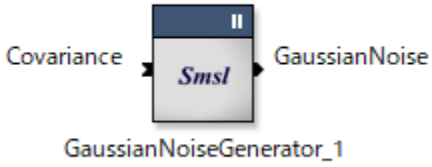
- 出力周波数は (IncrementDegree / 360) × トリガレートです。

入力ポート	データ型	説明
IncrementDegree	Float64	1 サンプルあたりの角度増分 [deg]。1 サンプルごとに 1 回出力します。
Gain	Float64	出力波形の振幅。

出力ポート	データ型	説明
Sine	Float64	Gain·sin(θ) + DC_bias。
Cosine	Float64	Gain·cos(θ) + DC_bias。

プロパティ	データ型	説明
InitAngleBias	Float64	初期角度 θ [deg]。（既定値: 0.0）
DC_bias	Float64	両出力に加えるオフセット。（既定値: 0.0）

GaussianNoiseGenerator



指定した共分散（1 サンプルごとの分散）を持つガウス分布（正規分布）のランダムノイズを、入力 1 サンプルごとに 1 サンプル生成します。

処理内容

- Covariance のサンプルが届くたびに、ノイズを 1 サンプル出力します（Covariance がトリガ入力です）。したがって、出力のレートは Covariance の入力のレートになります。

- 出力 = sqrt(|Covariance|) * n です。n は標準正規分布の乱数です（MT19937 の一様乱数と polar 法で生成します）。各サンプルの分散は、サンプリングレートによらず |Covariance| です。

データ形式

- 入力: スカラー（ベクトルサイズ 1。それ以外のときはエラーで停止します）。出力: Covariance 入力のタイムスタンプを持つ Float64 スカラー。

注意事項

- 目的の出力レートで、周期的な信号源（number generator など）から Covariance を与えてください。

- カルマンフィルタの観測ノイズを模擬するときは、Covariance にフィルタの観測ノイズ共分散 R と同じ値を与えてください。

- 乱数の系列は起動のたびに同じシードから始まるので、毎回同じノイズになります。Windows と Linux でも同じです。

参考文献

- [1] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, pp. 3-30, 1998. Used through std::mt19937 of the C++ standard library. https://doi.org/10.1145/272991.272995 — 使っている機能：一様乱数（MT19937）。

- [2] G. Marsaglia and T. A. Bray, "A Convenient Method for Generating Normal Variables," SIAM Review, vol. 6, no. 3, pp. 260-264, 1964. https://doi.org/10.1137/1006063 — 使っている機能：正規乱数（polar 法）。

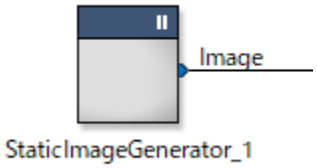
入力ポート	データ型	説明

Covariance	Float64	出力ノイズの共分散（1 サンプルごとの分散）。1 サンプルごとに出力が 1 サンプル生成されます。

出力ポート	データ型	説明
GaussianNoise	Float64	ガウスノイズのサンプル（平均 0、分散 |Covariance|）。

プロパティ	データ型	説明
-		

StaticImageGenerator



一様（単色）のグレースケールまたは RGB 画像を、1 回だけ、または周期的に出力します。オーバーレイ用の下地画像などに使います。

処理内容

- X_pixels × Y_pixels の画像を Brightness（GRAY）または (Red, Green, Blue)（RGB）で塗りつぶし、pPeriod マイクロ秒ごとに、output_mode = Once では起動時に 1 回だけ出力します。Once では、出したあとのプロパティの変更は出力に反映されません。

データ形式

- 出力: IplImage（8 ビット GRAY または RGB）。

注意事項

- レーザースキャンのオーバーレイ（DrawingObjects など）の地図背景として使います。

Input Ports	Data type	Description
–		

Output Ports	Data type	Description
Image	IplImage	生成した画像。

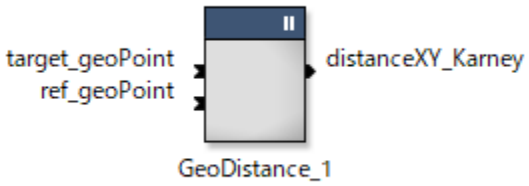
Properties	Data type	Description
X_pixels	Integer	画像の幅 [pixel]。（既定値: 640）
Y_pixels	Integer	画像の高さ [pixel]。（既定値: 480）

output_mode	ENUM	Once：起動時に 1 回だけ出す。Repeat：pPeriod ごとに出す。保存済みのダイアグラムの動きを変えないよう、既定は Repeat です。選択肢: Once, Repeat。（既定値: Repeat）
image_format	ENUM	画像のピクセル形式。選択肢: GRAY, RGB。（既定値: GRAY）
pPeriod	Integer	出力周期 [μs]（1 以上）。output_mode = Repeat のときだけあります。（既定値: 100000）
Brightness	Integer	GRAY モードの輝度（0〜255）。（既定値: 0）
Red	Integer	RGB モードの赤成分（0〜255）。（既定値: 0）
Green	Integer	RGB モードの緑成分（0〜255）。（既定値: 0）
Blue	Integer	RGB モードの青成分（0〜255）。（既定値: 0）

3.3 GNSS コンポーネント

これらの GNSS コンポーネントは、GNSS から得られる物理信号を計算するのに活用できます。

GeoDistance



2 地点の地理座標（緯度・経度）間の距離を、WGS84 楕円体上で Karney のアルゴリズム（GeographicLib）により計算します。オプションで Haversine 式による距離も出力します。

処理内容

- distanceXY_Karney: GeographicLib による測地線距離（ナノメートル精度）。

- distanceXY_Haversine（オプション）: 球面上の大円距離（高速だが精度は低い）。

使い方

- 主な入力: UnitDelay.output → target_geoPoint、data_vectorizer.o_data → ref_geoPoint。

- 主な出力: distanceXY_Karney → SkipEarlyFrames.input_0、distanceXY_Haversine → SkipEarlyFrames.input_1。

データ形式

- 入力: [緯度, 経度]（度）。出力: 距離 [m]。

注意事項

- https://geographiclib.sourceforge.io/ を参照してください。

参考文献

- [1] C. F. F. Karney, "Algorithms for Geodesics," Journal of Geodesy, vol. 87, no. 1, pp. 43-55, 2013. https://doi.org/10.1007/s00190-012-0578-z ー 使っている機能：WGS84 楕円体上の測地線距離（distanceXY_Karney）。

- [2] C. F. F. Karney, GeographicLib (bundled in 3rdparty/GeographicLib). https://github.com/geographiclib/geographiclib-cpp ー 使っている機能：Karney のアルゴリズムの実装。

- [3] R. W. Sinnott, "Virtues of the Haversine," Sky and Telescope, vol. 68, no. 2, p. 159, 1984. ー 使っている機能：Haversine 式の距離（distanceXY_Haversine）。

Karney 法はきわめて高生堂な計算を実現しますが、本コンポーネントは Property のチェックを入れることで、Haversine 近似計算も出力でき、コンポーネントの出力と外部の計算手法の結果を比較するのに使えます。

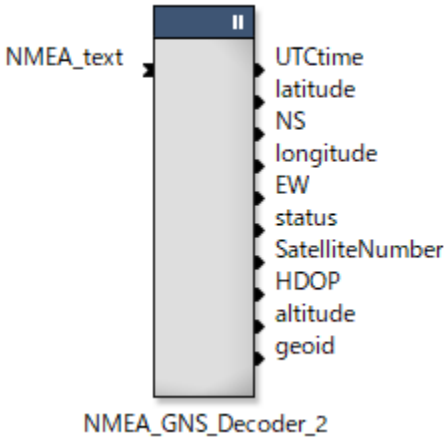
入力ポート	データ型	説明
target_geoPoint	Float64	対象地点 [緯度, 経度]（度）。
ref_geoPoint	Float64	基準地点 [緯度, 経度]（度）。

出力ポート	データ型	説明
distanceXY_Karney	Float64	測地線距離 [m]（Karney）。
distanceXY_Haversine	Float64	大円距離 [m]（Haversine）。 Output_Haversine が true のときに存在します。

プロパティ	データ型	説明

Output_Haversine	Bool	Haversine 距離の出力を追加します。（既定値: false）

NMEA_GNS_Decoder



NMEA 0183 テキストストリームの GNS センテンスを物理信号にデコードします（FURUNO PT-G1 で動作確認）。

処理内容

- 受信した GNS センテンスを解析し、各フィールドをテキストのタイムスタンプ付きで個別のポートに出力します。

データ形式

- 文字のフィールド（N/S、E/W、モード）は ASCII コードで出力します（例: N = 78、S = 83、E = 69、W = 87）。

注意事項

- NMEA シリアル／GPS コンポーネントのテキスト出力を接続してください。

参考文献

- [1] National Marine Electronics Association, NMEA 0183 Standard for Interfacing Marine Electronic Devices (GNS and RMC sentences). https://www.nmea.org/ ー 使っている機能：GNS センテンスの形式。

詳細は古野電気の web サイト参照:

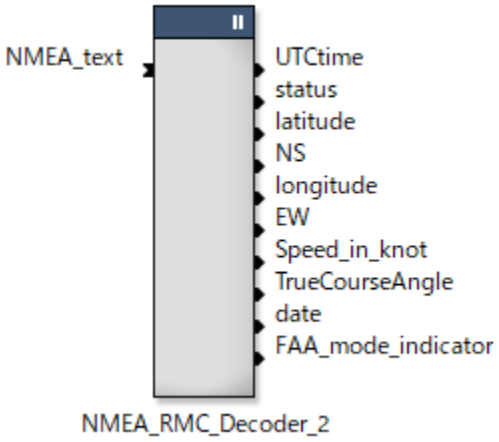
https://www.furuno.com/jp/gnss/datadownload/#PT-G1

入力ポート	データ型	説明
NMEA_text	Text	NMEA 0183 テキストストリーム。

出力ポート	データ型	説明
UTCtime	Float64	UTC 時刻（hhmmss.ss）。
latitude	Float64	緯度（10 進度。NMEA の ddmm.mmmm から変換）。
NS	Integer32	北緯/南緯の識別子（ASCII コード: 78 = N、83 = S）。
longitude	Float64	経度（10 進度。NMEA の dddmm.mmmm から変換）。
EW	Integer32	東経/西経の識別子（ASCII コード: 69 = E、87 = W）。
status	Integer32 vector	モード識別子の ASCII コード（例: 65 = A 単独測位、68 = D ディファレンシャル、78 = N 非測位）。衛星システムごとに 1 つ。
SatelliteNumber	Integer32	使用衛星数（0～24）。
HDOP	Float64	水平精度低下率 HDOP（0.0～50.0）。
altitude	Float64	標高 [m]。
geoid	Float64	ジオイド高 [m]。

プロパティ	データ型	説明
DebugOut	Bool	true の場合、受信した NMEA テキストをデバッグ用にコンソールへ出力します。（既定値: false）

NMEA_RMC_Decoder



NMEA 0183 テキストストリームの RMC センテンスを物理信号にデコードします（FURUNO PT-G1 で動作確認）。

処理内容

- 受信した RMC センテンスを解析し、各フィールドをテキストのタイムスタンプ付きで個別のポートに出力します。

使い方

- 主な入力: GPS_NMEA0183.oNMEAMessages → NMEA_text。

- 主な出力: latitude → data_vectorizer.i_data_1、 longitude → data_vectorizer.i_data_2、latitude → QMLViewer.latitude、 longitude → QMLViewer.longitude。

データ形式

- 文字のフィールドは ASCII コードで出力します（例: A = 65、V = 86、N = 78、S = 83、E = 69、W = 87）。

参考文献

- [1] National Marine Electronics Association, NMEA 0183 Standard for Interfacing Marine Electronic Devices (GNS and RMC sentences). https://www.nmea.org/ ー 使っている機能：RMC センテンスの形式。

詳細は古野電気の web サイト参照:

https://www.furuno.com/jp/gnss/datadownload/#PT-G1

入力ポート	データ型	説明
NMEA_text	Text	NMEA 0183 テキストストリーム。

出力ポート	データ型	説明
UTCtime	Float64	UTC 時刻（hhmmss.ss）。
status	Integer32	ステータスの ASCII コード（65 = A 有効、86 = V 無効）。
latitude	Float64	緯度（10 進度。NMEA の ddmm.mmmm から変換）。
NS	Integer32	北緯/南緯の識別子（ASCII コード: 78 = N、83 = S）。
longitude	Float64	経度（10 進度。NMEA の dddmm.mmmm から変換）。
EW	Integer32	東経/西経の識別子（ASCII コード: 69 = E、87 = W）。
Speed_in_knot	Float64	対地速度 [knot]。
TrueCourseAngle	Float64	対地進行方向 [deg]（0〜360）。
date	Integer32	日付（ddmmyy）。
FAA_mode_indicator	Integer32	FAA モードの ASCII コード（65 = A、68 = D、69 = E、78 = N）。

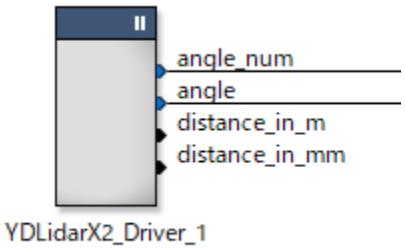
プロパティ	データ型	説明

DebugOut	Bool	true の場合、受信した NMEA テキストをデバッグ用にコンソールへ出力します。（既定値: false）

3.4 センサードライバコンポーネント

よく使用され、低コストなセンサーのドライバコンポーネントです。

YDLidarX2_Driver



USB シリアル接続の低価格 2D LiDAR「YDLIDAR X2」のドライバです。1 回転ごとに 1 スキャンを出力します。

処理内容

- DeviceName に 115200 baud で接続し、YDLIDAR のドライバコード（MIT ライセンス）でスキャンパケットをデコードします。

- Inverted は YDLIDAR のドライバにそのまま渡します。X2 は角度を時計回りに測ります。Inverted = true（初期値）では反時計回りが正の角度に変換し、このツールキットの座標系の約束（ROS REP-103：X 前方、Y 左、角度 0 は +X）に合わせます。変わるのは各点の角度（$2\pi - \theta$）だけで、点は測定順（時刻の順）のままです。そのため、角度は配列の後ろほど小さくなります。生の時計回りの角度が必要なときだけ false にしてください。

- StartAtZeroAngle は、角度 0 に最も近い点が先頭になるように配列を巡回シフトします。点は測定順（時間順）のままで、値による並べ替えはしません。並べ替えが必要なときはソート用のコンポーネントを使ってください。FixedOutputSize は 1 回転あたりの点数を一定にします。

使い方

- 主な出力: angle_num → DataViewer.input_0、angle → DataViewer.input_1、distance_in_m → laser_scan_to_xyz_points.dist_points。

データ形式

- angle（rad または deg、[-π, π)）、distance_in_m（Float64）、distance_in_mm（Integer32）は angle_num 個の要素を持ちます（最大 2048）。

- 測定範囲外（0.05 m 未満または 64 m 超）の点は距離 0 として残します。すべての出力のタイムスタンプは、スキャンを受信した時刻です。

注意事項

- X2 で動作確認しています。ほかの YDLIDAR 機種は自己責任で使用してください。

- バージョン 1.01：AngleSort を StartAtZeroAngle に改名し、Inverted（初期値 true）を追加しました。バージョン 1.0 は時計回りの角度を出力していたため、1.0 で記録したデータと合わせるときは Inverted = false にしてください。

参考文献

- [1] EAI (YDLIDAR), YDLidar-SDK (bundled in 3rdparty/YDLidar-SDK). https://github.com/YDLIDAR/YDLidar-SDK ー 使っている機能：センサとの接続とスキャンの取得（CYdLidar）。

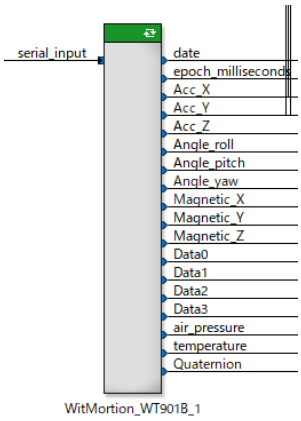
入力ポート	データ型	説明
-		

出力ポート	データ型	説明
angle_num	Integer32	現在の 1 回転の点数。

angle	Float64 vector	各点の角度（AngleUnit で選んだ単位）。Inverted が true（初期値）のとき反時計回りが正です。
distance_in_m	Float64 vector	各点の距離 [m]。
distance_in_mm	Integer32 vector	各点の距離 [mm]。

プロパティ	**データ型**	**説明**
DeviceName	Text	LiDAR のシリアルデバイス（Windows は COMx、Linux は /dev/tty*）。（既定値: COM1）
FixedOutputSize	Bool	true の場合、毎回転を同じ点数で出力します。（既定値: false）
StartAtZeroAngle	Bool	true の場合、角度 0 に最も近い点から出力します（巡回シフト、測定順は保持）。false の場合はドライバの順のまま出力します。（既定値: false）
AngleUnit	ENUM	角度出力の単位。選択肢: radian, degree。（既定値: radian）
Inverted	Bool	YDLIDAR のドライバにそのまま渡します。true（初期値）：X2 は時計回りに測るため、反時計回りが正の角度（ROS REP-103）に変換します。false：X2 の生の時計回りの角度です。どちらでも、点は測定順（時刻の順）のままです。（既定値: true）

WitMotion_WT901B



WitMotion WT901B IMU のシリアルデータ（加速度、角速度、角度、磁場、気圧、温度、クォータニオン）をデコードします。

処理内容

- バイトストリームを WitMotion のパケットに分割し、パケットの種類ごとに対応する出力を更新します。

使い方

- 主な入力: SerialPort.streamOutput → serial_input。

- 主な出力: date → DataViewer.input_0、epoch_milliseconds → DataViewer.input_1、Acc_X → DataViewer.input_2、Acc_Y → DataViewer.input_3。

データ形式

- 入力: UInt8 のシリアルバイト（9600 baud）。出力: スカラーと 4 要素のクォータニオン。

注意事項

- シリアルポートコンポーネントの出力を接続してください。例 WitMotion_WT901B_imu_demo を参照。

参考文献

- [1] WitMotion Shenzhen Co., Ltd., WT901 series serial communication protocol (data packet format used by WT901B). https://www.wit-motion.com/ ー 使っている機能：シリアルのデータパケットの形式。

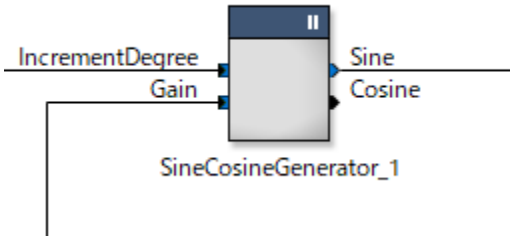
入力ポート	データ型	説明
serial_input	UInt8	IMU のシリアルポートからの生バイト列。

出力ポート	データ型	説明
date	UInt32	IMU 起動からの日付（20YYMMDD）。
epoch_milliseconds	UInt64	IMU 起動からの時間 [ms]。
Acc_X	Float64	X 軸の加速度 [m/s²]。
Acc_Y	Float64	Y 軸の加速度 [m/s²]。
Acc_Z	Float64	Z 軸の加速度 [m/s²]。
Gyro_X	Float64	X 軸の角速度 [deg/s]。
Gyro_Y	Float64	Y 軸の角速度 [deg/s]。
Gyro_Z	Float64	Z 軸の角速度 [deg/s]。
Angle_roll	Float64	ロール角 [deg]。
Angle_pitch	Float64	ピッチ角 [deg]。
Angle_yaw	Float64	ヨー角 [deg]。
Magnetic_X	Float64	X 軸の磁場。
Magnetic_Y	Float64	Y 軸の磁場。
Magnetic_Z	Float64	Z 軸の磁場。
Data0	UInt16	データポート チャンネル 0 の値。

Data1	UInt16	データポート チャンネル 1 の値。
Data2	UInt16	データポート チャンネル 2 の値。
Data3	UInt16	データポート チャンネル 3 の値。
air_pressure	UInt32	気圧 [Pa]。
temperature	Float64	温度 [°C]。
Quaternion	Float64 vector	姿勢のクォータニオン（4 要素）。

プロパティ	**データ型**	**説明**
-		

Hi_Link_LD2410C



シリアルポートに接続した Hi-Link HLK-LD2410C（24 GHz ミリ波人感レーダー）のドライバです。解読した目標の状態・距離・エネルギーを出力します。

処理内容

- 開始時にポートを 256000 bps（8N1）で開きます。センサを設定モードにし、ファームウェアのバージョン、距離分解能、現在の検出の設定（最大ゲート、感度、無人の時間）をコンソールに出力し、engineering_mode に従ってエンジニアリングモードを ON／OFF にしてから、設定モードを終了します。各コマンドは ACK を待ち、ACK が来なければエラーで停止します。

- apply_sensor_settings が true のとき、プロパティと違う検出の設定をセンサに書き込み、読み直して確かめます。この設定はセンサに保存され、電源を切っても残ります。false のときは、センサの設定を変えません。

- polling_ratio_in_ms ごとにシリアルポートを読みます。受信したバイト列から報告フレーム（ヘッダ F4 F3 F2 F1、テール F8 F7 F6 F5）を取り出し、長さ・AA/55/00 の位置・テールを確かめます。形式が合わないフレームは捨て、その数を停止時にコンソールに出力します。

- 正しいフレームごとに 1 回出力します。1 つのフレームから出す出力は、すべてバイト列を読んだ時刻をタイムスタンプに持ちます。

- polling_ratio_in_ms は 50 ms（20 Hz、報告のレートの 2 倍）以下に制限しているので、1 回の読み取りに入るフレームはふだん 1 つまでです。それでも複数のフレームが入った読み取り（それらのフレームは同じタイムスタンプになります）は回数を数え、停止時にコンソールに出力します。

使い方

- 主な出力: moving_distance_in_m → Oscilloscope.inputA、stationary_distance_in_m → Oscilloscope.inputB、detection_distance_in_m → Oscilloscope.inputC、target_state → Oscilloscope.inputA。

データ形式

- target_state: 0 なし、1 動いている、2 静止、3 動いている＋静止（4〜6 は背景ノイズ検出の実行中だけ）。

- 距離は cm から m に変換します。エネルギーは 0〜100 です。

- エンジニアリングモードのみ: moving_gate_energy と stationary_gate_energy は距離ゲート 0〜N ごとに 1 つの値を持ちます（工場出荷の設定では 9 個、1 ゲート 0.75 m）。light_level は 0〜255、out_pin は OUT ピンの状態です。基本モードではこの 4 つは出力しません。

注意事項

- port_name にシリアルデバイス（例: COM3、/dev/ttyUSB0）を指定してください。センサのボーレートは工場出荷値の 256000 である必要があります。

- 検出の設定は、apply_sensor_settings が true のときだけ変更します。プロパティの初期値は工場出荷値と同じです。初期値のまま apply_sensor_settings = true で 1 回動かすと、工場出荷の設定に戻ります。

- センサの報告周期は約 10 Hz です。

- バージョン 1.02: 生のデータ出力を解読した出力に置き換え、engineering_mode と検出の設定のプロパティを追加しました。

- バージョン 1.03：polling_ratio_in_ms は 1〜50 です。それより大きい値（バージョン 1.01 で保存された 100 など）では、エラーで停止します。

参考文献

- [1] Shenzhen Hi-Link Electronic Co., Ltd., HLK-LD2410C Serial Communication Protocol, V1.07, 2024-08-05. ー 使っている機能：シリアルのフレーム、コマンド、報告データ。

入力ポート	データ型	説明
-		

出力ポート	データ型	説明
target_state	Integer32	目標の状態: 0 なし、1 動いている、2 静止、3 両方（4〜6: 背景ノイズ検出）。
moving_distance_in_m	Float64	動いている目標の距離 [m]。
moving_energy	Integer32	動いている目標のエネルギー（0〜100）。
stationary_distance_in_m	Float64	静止している目標の距離 [m]。
stationary_energy	Integer32	静止している目標のエネルギー（0〜100）。
detection_distance_in_m	Float64	検出距離 [m]。
moving_gate_energy	Integer32 vector	距離ゲート 0〜N ごとの動きのエネルギー（エンジニアリングモードのみ）。
stationary_gate_energy	Integer32 vector	距離ゲート 0〜N ごとの静止のエネルギー（エンジニアリングモードのみ）。

light_level	Integer32	光センサの値、0～255（エンジニアリングモードのみ）。
out_pin	Integer32	OUT ピンの状態。人を検出しているとき 1（エンジニアリングモードのみ）。

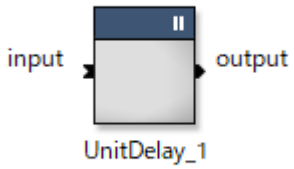
プロパティ	**データ型**	**説明**
port_name	Text	シリアルポート名（例: COM3、/dev/ttyUSB0）。空にはできません。（既定値: COM1）
engineering_mode	Bool	true のとき、センサをエンジニアリングモードにし、ゲートごとのエネルギー・光センサの値・OUT ピンの状態を出力します。（既定値: true）
polling_ratio_in_ms	Integer	シリアルポートを読む周期 [ms]（1～50。それ以外のときはエラーで停止します）。この上限で、ポーリングを報告のレート（約 10 Hz）の 2 倍の 20 Hz 以上に保ちます。（既定値: 10）
apply_sensor_settings	Bool	true のとき、下の 5 つの検出の設定をセンサに書き込みます（違う値だけ）。センサに保存され、電源を切っても残ります。（既定値: false）
max_moving_gate	Integer	動いている目標の最大ゲート（2～8）。apply_sensor_settings が true のときだけ使います。（既定値: 8）
max_stationary_gate	Integer	静止している目標の最大ゲート（2～8）。apply_sensor_settings が true のときだけ使います。（既定値: 8）

no_one_duration_in_s	Integer	最後に検出してから「なし」になるまでの時間 [s]（0〜65535）。apply_sensor_settings が true のときだけ使います。（既定値: 5）
moving_sensitivity	Text	ゲート 0〜8 の動きの感度。0〜100 の整数をカンマ区切りで 9 個。エネルギーがこの値を超えると検出します。100 にするとそのゲートでは検出しません。apply_sensor_settings が true のときだけ使います。（既定値: 50,50,40,30,20,15,15,15,15）
stationary_sensitivity	Text	ゲート 0〜8 の静止の感度（形式は同じ）。ゲート 0、1 は設定できないので無視します。apply_sensor_settings が true のときだけ使います。（既定値: 0,0,40,40,30,30,20,20,20）

3.5 信号処理コンポーネント

必要に応じてこれらの補助コンポーネントを使用できます。

UnitDelay



信号を delay_length サンプル遅らせます。

処理内容

- サンプルを長さ delay_length のバッファに保持し、新しい入力ごとに delay_length サンプル前に受信したサンプルを出力します。

- 出力のタイムスタンプは、いま受け取ったサンプルのタイムスタンプです（値は delay_length サンプル前に受け取ったもの）。

- 最初の delay_length サンプルのあいだは、入力と同じ長さの 0 を出力します。

使い方

- 主な入力: data_vectorizer.o_data → input。

- 主な出力: output → GeoDistance.target_geoPoint。

データ形式

- data_type で選んだ型の入出力が 1 組作られます。ベクトルにも対応します。

- ベクトルの長さはサンプルごとに変わってもかまいません（最初のサンプルのバッファの大きさまで）。遅延したサンプルは、そのサンプルの長さで出力します。

- delay_length は 1 以上にします。

入力ポート	データ型	説明
input_int8	Int8	入力信号（data_type = Int8 のとき存在）。
input_int16	Int16	入力信号（data_type = Int16 のとき存在）。
input_int32	Integer32	入力信号（data_type = Int32 のとき存在）。
input_int64	Integer64	入力信号（data_type = Int64 のとき存在）。

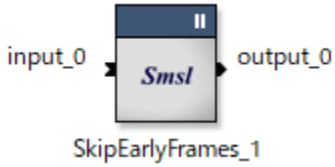
input_uint8	UInt8	入力信号（data_type = Uint8 のとき存在）。
input_uint16	UInt16	入力信号（data_type = Uint16 のとき存在）。
input_uint32	UInt32	入力信号（data_type = Uint32 のとき存在）。
input_uint64	UInt64	入力信号（data_type = Uint64 のとき存在）。
input_float32	Float32	入力信号（data_type = Float32 のとき存在）。
input_float64	Float64	入力信号（data_type = Float64 のとき存在）。
input		遅延させる信号（data_type で選んだ型。ベクトル対応）。

出力ポート	**データ型**	**説明**
output_int8	Int8 vector	遅延した信号（data_type = Int8 のとき存在）。
output_int16	Int16 vector	遅延した信号（data_type = Int16 のとき存在）。
output_int32	Integer32 vector	遅延した信号（data_type = Int32 のとき存在）。
output_int64	Integer64 vector	遅延した信号（data_type = Int64 のとき存在）。
output_uint8	UInt8 vector	遅延した信号（data_type = Uint8 のとき存在）。
output_uint16	UInt16 vector	遅延した信号（data_type = Uint16 のとき存在）。

output_uint32	UInt32 vector	遅延した信号（data_type = Uint32 のとき存在）。
output_uint64	UInt64 vector	遅延した信号（data_type = Uint64 のとき存在）。
output_float32	Float32 vector	遅延した信号（data_type = Float32 のとき存在）。
output_float64	Float64 vector	遅延した信号（data_type = Float64 のとき存在）。
output		delay_length サンプル遅延した入力。タイムスタンプは、いま受け取ったサンプルのものです。

プロパティ	**データ型**	**説明**
delay_length	Integer	遅延量（サンプル数。1 = 1 サンプル遅延）。（既定値: 1）
data_type	ENUM	入出力のデータ型。選択肢: Int8, Int16, Int32, Int64, Uint8, Uint16, Uint32, Uint64, Float32, Float64。（既定値: Int8）

SkipEarlyFrames



入力ごとに最初の n_skip_early_frames フレームを捨て、その後のフレームをそのまま出力します。

処理内容

- センサの直後に置くと、下流のどのコンポーネントも同じフレームを受け取ります（センサの起動直後に壊れたフレームを捨てるときなど）。

- 入力と出力の組を nb_channels 個作ります。組ごとに、ダイアグラムの開始から自分のフレームを数え、途中で数え直しません。

- データ、タイムスタンプ、Quality、Frequency、Misc1～Misc3 を写します。出力の型・型の名前・単位は、最初に通すフレームで入力のものになります。

使い方

- 主な入力: hdl_64e_s2_convert_to_XYZ.XYZ_points → input_0、pandar_xt_convert_to_XYZ.o_XYZ_points → input_0、pandarX_udp_decoder.o_intensities → input_1、ouster_lidar.pointcloud_ret1 → input_0。

- 主な出力: output_0 → Rotation3D.xyz_points_in、output_0 → DataViewer.input_0、output_0 → GL2_3D_Viewer.object_2_data、output_0 → rings_transpose.pc。

データ形式

- 通せる入力：整数と浮動小数点の配列（Integer8/16/32/64、UnsignedInteger8/16/32/64、Float32、Float64）、IplImage、ポインタを持たないカスタムストラクチャー（SMSL_MapSpec など）。

- 配列の長さはフレームごとに変わってもよく、最初に通したフレームのバッファの大きさまでです。IplImage は大きさ・チャネル数・深さを変えないでください。

- ほかの型や、最初に通したフレームから型や大きさが変わったときは、エラーで止まります。

注意事項

- 時間ではなくフレームの数で捨てます。n_skip_early_frames = 0 なら全部のフレームを通します。

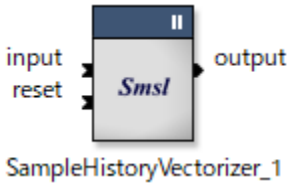
入力ポート	データ型	説明
input	Any	入力。nb_channels 個のポート input_0、input_1、…
input_$1		入力 $1（整数・浮動小数点の配列、IplImage、カスタムストラクチャー）。

出力ポート	データ型	説明
output	AnyType vector	出力。nb_channels 個のポート output_0、output_1、…（output_k は input_k を通す）。
output_$1		最初の n_skip_early_frames フレームを除いた input_$1。型とタイムスタンプは入力と同じ。

プロパティ	データ型	説明

nb_channels	Integer	入力と出力の組の数（1 以上）。(既定値: 1)
n_skip_early_frames	Integer	入力ごとに、最初に捨てるフレームの数（0 以上。0 なら捨てない）。(既定値: 3)

SampleHistoryVectorizer



1 つの数値の入力の直近 max_samples サンプルをため、サンプルが届くたびに、古い順につなげた 1 本のベクトルにして出力します。

処理内容

- 1 サンプルは K 個の値のベクトルです。出力は [sample(t-n+1), …, sample(t-1), sample(t)]（n = 1, 2, … max_samples）で、長さは K × n です。max_samples 個たまったあとは、いちばん古いサンプルを捨てます（リングバッファ）。

- 1 つずつ届く姿勢から軌跡を描くときや、1 つの信号の移動窓を MaxMinMeanforArray に渡すときに使います。

- reset：最後に届いた値が 0 以外のあいだと、0 以外の値が届いたフレーム（短いパルスも）では、ためたサンプルと今のサンプルを捨て、何も出力しません。リセットのあとは、また 1 サンプルずつ増えます。

データ形式

- 受ける入力：整数と浮動小数点のベクトル（Integer8/16/32/64、UnsignedInteger8/16/32/64、Float32、Float64）。出力は最初のサンプルで入力の型・型の名前・単位になります。

- K は最初のサンプルで決まり、変わってはいけません。長さや型が変わると、エラーで止まります。

注意事項

- 1 つの入力を時間の向きにつなげます。同じ時刻の複数の入力をつなげるときは、RTMaps の Vectorizer を使います。

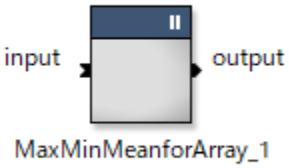
入力ポート	データ型	説明
input	Numbers	ためる信号（長さ K の決まった、整数か浮動小数点のベクトル）。
reset	UInt8	リセット信号（つながなくてよい、UnsignedInteger8）。0 以外で、ためたサンプルを捨てます。0 以外のあいだは何も出力しません。

出力ポート	データ型	説明
output	AnyType vector	直近 n 個（$n \leq max_samples$）のサンプルを古い順につなげたもの。長さ $K \times n$。型は入力と同じ。

プロパティ	データ型	説明
max_samples	Integer	ためるサンプルの数（1 以上）。（既定値: 10）
output_timestamp	ENUM	出力のタイムスタンプ。newest：今届いたサンプル、oldest：出力の中でいちば

		ん古いサンプル（先頭の組）。選択肢: newest, oldest。（既定値: newest）

MaxMinMeanforArray



ベクトルの要素の最大値・最小値・平均値を計算します。

使い方

- 主な入力: number_generator.o_float64 → input。

- 主な出力: output → DataViewer.input_0、output → DataViewer.input_1、output → DataViewer.input_2。

データ形式

- 入力: Float64 ベクトル。出力: Float64 スカラー。

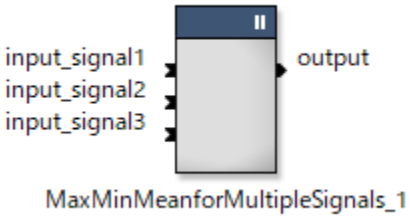
入力ポート	データ型	説明
input	Float64	入力ベクトル。

出力ポート	データ型	説明
output	Float64	入力要素の最大値・最小値・平均値。

プロパティ	データ型	説明

mode	ENUM	計算する統計量。選択肢: max, min, mean。（既定値: max）

MaxMinMeanforMultipleSignals



複数のスカラー信号にわたる最大値・最小値・平均値を計算します。

処理内容

- n_input_signal 個の入力を作り、それらの現在値に対して統計量を計算します。

使い方

- 主な入力: number_generator.o_float64 → input_signal1、number_generator.o_float64 → input_signal2、number_generator.o_float64 → input_signal3。

- 主な出力: output → DataViewer.input_0、output → DataViewer.input_1、output → DataViewer.input_2。

データ形式

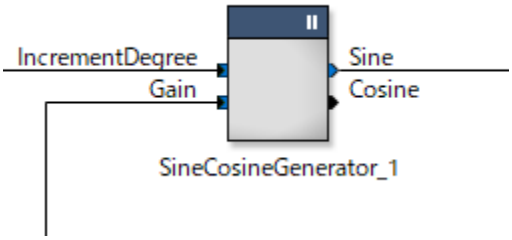
- 入力: Float64 スカラー（ベクトルサイズ 1）。出力: Float64 スカラー。

入力ポート	データ型	説明
input	Float64	スカラー入力信号。n_input_signal 個のポート（input_signal1、input_signal2、…）。
input_signal$1		$1 番目のスカラー入力信号（ベクトルサイズ 1）。

出力ポート	データ型	説明
output	Float64	入力の最大値・最小値・平均値。

プロパティ	データ型	説明
n_input_signal	Integer	作成する入力ポートの数。（既定値: 1）
mode	ENUM	計算する統計量。選択肢: max, min, mean。（既定値: max）

NumericalTextWriter



数値信号を 16 進または 2 進のテキストとして CSV ファイルに書き出します。

処理内容

- output_path に <output_filename_header>_<開始時刻 ns>.csv というファイルを作成します。

- 受信したサンプルごとに、タイムスタンプ、信号名、値（16 進 0x.. または 2 進 0b..）を 1 行として書き込みます。

データ形式

- 入力: UInt8 ベクトル。n_input_signal 個のポートを作成します。

注意事項

- Byte2bits / bits2Byte と組み合わせて、ビット・バイト単位のデータを記録するのに便利です。

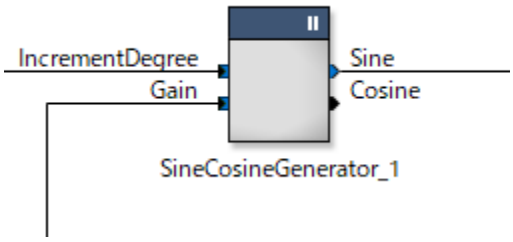
入力ポート	データ型	説明
input	UInt8	書き出す信号。n_input_signal 個のポート（input1、input2、…）。
input$1		CSV ファイルに書き出す $1 番目の信号。

出力ポート	データ型	説明

-		

プロパティ	データ型	説明
n_input_signal	Integer	作成する入力ポートの数。（既定値: 1）
output_format	ENUM	値の数値形式。選択肢: hex, bin。（既定値: hex）
add_format_identifier	Bool	true の場合、値に 0x または 0b を付けます。（既定値: true）
output_path	Text	出力フォルダ（必要に応じて作成）。
output_filename_header	Text	出力ファイル名の接頭辞。（既定値: rtmaps）
signal_name	Text	各行に書き込む信号名。（既定値: signal_name）
input$1_name	Text	input$1 の CSV 行に書き込む信号名。

bits2Byte



8 つのビット信号から 1 バイトを合成します。

処理内容

- Byte = Σ (bit_k & 1) << k、k = 0..7（bit_0 が LSB）。

使い方

- 主な入力: Devectorizer.element0 → bit_0、Devectorizer.element1 → bit_1、Devectorizer.element2 → bit_2、Devectorizer.element3 → bit_3。

- 主な出力: Byte → DataViewer.input_0。

データ形式

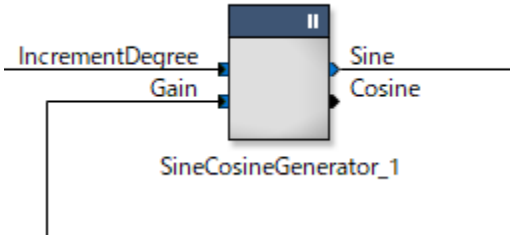
- 入力: UInt8 スカラー（0 または 1）。出力: UInt8 スカラー。

入力ポート	データ型	説明
bit_0	UInt8	ビット 0（LSB）。
bit_1	UInt8	ビット 1。
bit_2	UInt8	ビット 2。
bit_3	UInt8	ビット 3。
bit_4	UInt8	ビット 4。
bit_5	UInt8	ビット 5。
bit_6	UInt8	ビット 6。
bit_7	UInt8	ビット 7（MSB）。

出力ポート	データ型	説明
Byte	UInt8	合成したバイト。

プロパティ	データ型	説明
-		

Byte2bits



1 バイトを 8 つのビットに分解します。

処理内容

- bits[k] = (Byte >> k) & 1、k = 0..7（要素 0 が LSB）。

使い方

- 主な入力: number_generator.o_uint8 → Byte。

- 主な出力: bits → Devectorizer.iVector。

データ形式

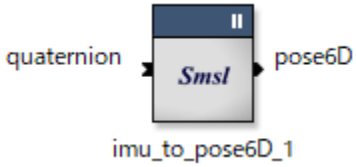
- 入力: UInt8 スカラー。出力: 8 要素の UInt8 ベクトル。

入力ポート	データ型	説明
Byte	UInt8	入力バイト（先頭要素を使用）。

出力ポート	データ型	説明
bits	UInt8 vector	バイトの各ビット（LSB から順）。

プロパティ	データ型	説明
-		

imu_to_pose6D



IMU・AHRS の姿勢のクォータニオンを、機種によらず pose6D にします。

処理内容

- クォータニオンを正規化して回転行列にし、ロール・ピッチ・ヨーにします（R = Rz(yaw) Ry(pitch) Rx(roll)）。

- reference_frame = NED のときは、ワールド ENU・機体 FLU に直します。ロールは同じ符号、ピッチは逆の符号で、yaw = 90° − 方位です。

- 出力のタイムスタンプは入力と同じです。

- ここではセンサーフュージョンをしません。姿勢を出すセンサか、`imu_orientation_estimator` を使ってください。

データ形式

- quaternion は 4 要素です。ほかの要素数ではエラーで止まります。

- pose6D は [0, 0, 0, roll, pitch, yaw]（m、rad）です。IMU では位置を測れないので、並進は常に 0 です。

注意事項

- 有限でないサンプルと、ノルムが 1e-6 より小さいサンプルは飛ばします。警告は 1 回だけ出します。

- 出力は IMU 自身の姿勢です。IMU の取り付けは、姿勢を使う側で Sensor Calibration File Parser の行列を当ててください。

- ヨーの基準は機種によります。6 軸のセンサはヨーがドリフトします。

参考文献

- [1] T. Foote and M. Purvis, REP 103: Standard Units of Measure and Coordinate Conventions, ROS Enhancement Proposals. https://www.ros.org/reps/rep-0103.html ー 使っている機能：ワールド ENU と機体 FLU の軸、単位。

- [2] REP 145: Conventions for IMU Sensor Drivers, ROS Enhancement Proposals. https://www.ros.org/reps/rep-0145.html ー 使っている機能：IMU のドライバが出す姿勢の約束。

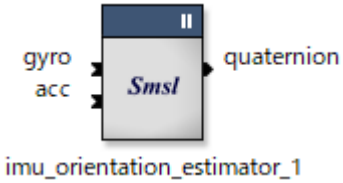
- [3] sensor_msgs/Imu message definition, ROS common_msgs. https://docs.ros.org/en/api/sensor_msgs/html/msg/Imu.html ― 使っている機能：クォータニオンの並び xyzw。

入力ポート	データ型	説明
quaternion	Float64	姿勢のクォータニオン（4 要素）。並びは quaternion_order で決まります。

出力ポート	データ型	説明
pose6D	Float64 vector	IMU の姿勢：[0, 0, 0, roll, pitch, yaw]（m、rad）。

プロパティ	データ型	説明
quaternion_order	ENUM	要素の並び：wxyz（スカラーが先頭）、または xyzw（スカラーが末尾。ROS の sensor_msgs/Imu と同じ）。選択肢: wxyz, xyzw。（既定値: wxyz）
reference_frame	ENUM	入力の座標：ENU（ワールド ENU、機体 FLU。ROS と同じ）、または NED（ワールド NED、機体 FRD）。選択肢: ENU, NED。（既定値: ENU）

imu_orientation_estimator



ジャイロと加速度から、選べば磁気も使って、GNSS なしで姿勢のクォータニオンを推定します。

処理内容

- `gyro` のサンプルごとに、前と今のサンプルの平均で姿勢を動かし、最新の加速度でロールとピッチを補正します。

- `filter_mode` で補正の方式を選びます。Mahony（相補フィルタ。ジャイロのバイアスも推定）または Madgwick（勾配降下法）です。

- `use_magnetometer` を使うと、最新の磁場で yaw も補正し、yaw がドリフトしなくなります。

- 最初の `gyro` のサンプルで、加速度（と磁場）から最初の姿勢を作ります。磁気がなければ yaw は 0 から始まります。

- `gyro`、`acc`、`mag` のタイムスタンプはそろっていなくてかまいません。出力のタイムスタンプは `gyro` と同じです。

データ形式

- `gyro`、`acc`、`mag` は 3 要素で、ほかの要素数はエラーで止まります。軸ごとに出力があるセンサは、RTMaps の Vectorizer（Sampling mode = Synchronized）でまとめます。

- `acc` は比力で、静止して水平なとき +Z が正です。`acc` と `mag` は向きだけを使うので、単位は問いません。

- `quaternion` は `[w, x, y, z]`（機体 → ワールド）です。機体の軸が X 前、Y 左、Z 上なら、`imu_to_pose6D` に `wxyz`、`ENU` でつなぎます。

注意事項

- 磁気がなければ yaw は相対で、ドリフトします。磁気があれば yaw は磁北が基準で、`magnetic_declination_in_deg`（東偏が正）を入れると真北が基準になります。Topic「磁気偏角」を参照してください。

- 動きの加速度も重力として扱うので、大きな加減速の間はロールとピッチに誤差が出ます。

- 磁気センサはあらかじめ較正してください。モータ、鉄骨、バッテリーの近くでは乱れます。

- `acc`、`mag` が NaN・無限大、またはノルムが 1e-6 より小さいサンプルは無視し、最後の有効なサンプルを使います。`gyro` が NaN・無限大のサンプルは飛ばします。警告は 1 回出します。

- GNSS があるときは EKF を使ってください。このコンポーネントは位置を使いません。

参考文献

- [1] R. Mahony, T. Hamel and J.-M. Pflimlin, "Nonlinear Complementary Filters on the Special Orthogonal Group," IEEE Transactions on Automatic Control, vol. 53, no. 5, pp. 1203-1218, 2008. https://doi.org/10.1109/TAC.2008.923738 — 使っている機能：Mahony の更新。

- [2] S. O. H. Madgwick, A. J. L. Harrison and R. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," 2011 IEEE International Conference on Rehabilitation Robotics, pp. 1-7, 2011. https://doi.org/10.1109/ICORR.2011.5975346 — 使っている機能：Madgwick の更新。

- [3] T. Foote and M. Purvis, REP 103: Standard Units of Measure and Coordinate Conventions, ROS Enhancement Proposals. https://www.ros.org/reps/rep-0103.html — 使っている機能：軸と単位。

入力ポート	データ型	説明

gyro	Float64	角速度 `[x, y, z]`（単位は `gyro_unit`）。サンプルごとに姿勢を更新して 1 回出力します。
acc	Float64	加速度 `[x, y, z]`（単位は問わない）。最新の有効なサンプルを使います。
mag	Float64	磁場 `[x, y, z]`（単位は問わない、較正済み）。`use_magnetometer` のときだけあります。最新の有効なサンプルを使います。

出力ポート	**データ型**	**説明**
quaternion	Float64 vector	姿勢 `[w, x, y, z]`（機体 → ワールド）。

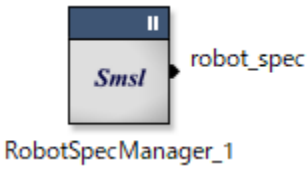
プロパティ	**データ型**	**説明**
filter_mode	ENUM	`Mahony` または `Madgwick`。選択肢: Mahony, Madgwick。（既定値: Mahony）
gyro_unit	ENUM	`gyro` の単位。`rad_per_s` または `deg_per_s`。選択肢: rad_per_s, deg_per_s。（既定値: rad_per_s）
use_magnetometer	Bool	入力 `mag` で yaw を補正します。（既定値: false）
max_gap_in_ms	Integer	2 つの `gyro` のサンプルの間隔がこれより長い、または 0 以下のとき、そのサンプルでは姿勢を更新しません。（既定値: 500）
mahony_kp	Float64	補正の強さ（1/s）。時定数は 1/`mahony_kp` です。0 より大きい値。（既定値: 1.0）

mahony_ki	Float64	ジャイロのバイアスの推定の強さ（1/s^2）。0 で推定しません。0 以上の値。（既定値: 0.01）
madgwick_beta	Float64	補正の速さ（rad/s）。0 より大きい値。（既定値: 0.1）
magnetic_declination_in_deg	Float64	磁気偏角（東偏が正、−180〜180）。0 なら yaw は磁北が基準です。（既定値: 0.0）

3.6 ハードウェア仕様コンポーネント

ロボットの体など、ハードウェアの仕様をほかのコンポーネントへ渡すコンポーネントです。

RobotSpecManager



ロボットの仕様の XML ファイルを読み、ロボットの寸法と質量（SMSL_RobotSpec）と、ファイルに書いてあればその性能値の限界（SMSL_RobotMotionLimits）を、始めに 1 回だけ出力します。

処理内容

- 始めに spec_file のファイルを読んで確かめ、ロボットの仕様を 1 回だけ出力し、あとはダイアグラムが止まるまで待ちます。ロボットを使う部品は値を線で受けるので、ロボットの大きさのプロパティを持つ必要がありません。

- ロボットの体の輪郭（フットプリント）は、長方形（ロボットの原点から前端・後端・左端・右端までの距離）、円、多角形（3〜32 頂点、ROS の footprint の書き方）の

どれかで与えます。出力する輪郭は必ず反時計回りです。内接円の半径は原点から辺までの最短距離、外接円の半径は原点から頂点までの最長距離です。円のときは、どちらも半径です（輪郭は円を内に含む正 16 角形）。

- ロボットの種類（differential、omni、ackermann、drone）によって、body 要素と motion 要素のどの属性が必須か、任意か、書けないかが決まります。body 要素には、高さ、質量と重心、車輪の配置を書きます。motion 要素には、機体の限界（最大速度、最大加速度、最大減速度（非常停止の減速度を含む）、最大角速度、ドローンでは傾きの最大角、ackermann では舵角を切る速さの最大）を書きます。ロボットが動けない方向は、出力では 0 です。

- 構造体は DataViewer で見えないので、値は始めにログへ出します。

- ファイルが読めないとき、属性が足りないか種類に合わないとき、値が有限の数でないか範囲外のとき、多角形が自分と交わるか面積が 0 のとき、ロボットの原点が輪郭の内側にないときは、エラーで止まります。

使い方

- 主な出力: robot_spec → rings_pose_estimation_icp.robot_spec、robot_motion_limits → rings_pose_estimation_icp.robot_motion_limits、robot_spec → rings_pose_estimation_p2plane.robot_spec、robot_motion_limits → rings_pose_estimation_p2plane.robot_motion_limits。

データ形式

- ロボットの座標は右手系で、X が前、Y が左、Z が上、単位は m です。原点はロボットの姿勢（pose2D）が指す点です。地上のロボットは z = 0 が接地面です。

- ファイル：UTF-8 の XML。ルート要素は robot_spec で、属性 type を持ちます。footprint 要素（属性 shape：rectangle、circle、polygon）を 1 つ、body 要素と motion 要素

を任意で 1 つずつ書きます。body か motion を書くときは type が必須です。知らない要素・属性はエラーです。

- 出力 robot_spec：カスタムストラクチャー SMSL_RobotSpec（632 バイト）。頂点の数、種類、内接円と外接円の半径、最大 32 個の頂点（x, y、反時計回り）、続いて body 要素の値（書いていないか、種類で使わない値は 0）。出力 robot_motion_limits：カスタムストラクチャー SMSL_RobotMotionLimits（120 バイト）。motion 要素を書いたときだけ作られます。速度と加速度は 0 以上です。

- 4 つの種類のサンプルが example フォルダにあります（robot_spec_differential.xml、robot_spec_omni.xml、robot_spec_ackermann.xml、robot_spec_drone.xml）。

注意事項

- センサの取り付け（各センサの位置と向き）は、このコンポーネントでは与えません。RTMaps 標準のコンポーネント Sensor Calibration File Parser を使い、その行列の出力を Rotation3D（Matrix Input）につないでください。トピック「ロボットの仕様とセンサの取り付け」を参照してください。

- motion 要素に書くのは機体の限界だけです。今日使う速さは、使う部品のプロパティに、限界以下の値で持ちます。

- 摩擦係数はロボットの仕様に入りません。床の値だからです。

- ファイルの motion 要素を足したり外したりしたときは、出力 robot_motion_limits が作られる・消えるように、spec_file を選び直してください。

入力ポート	データ型	説明
-		

出力ポート	データ型	説明

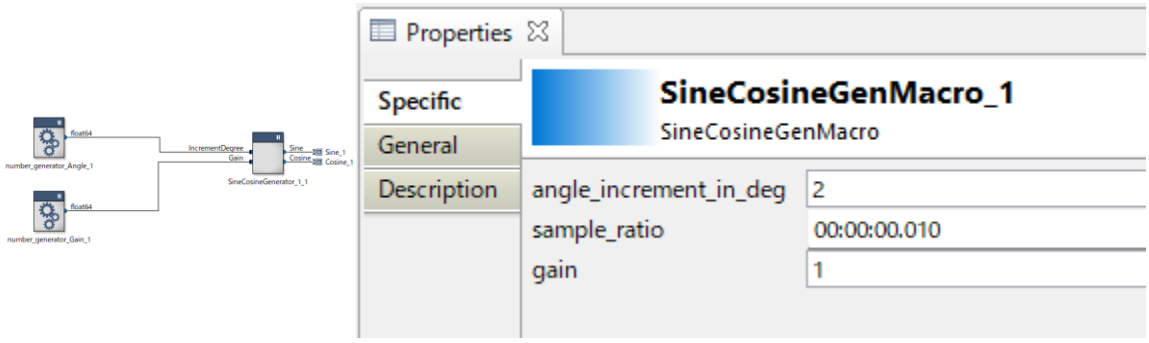
robot_spec	SMSL_RobotSpec vector	ロボットの寸法と質量をカスタムストラクチャー SMSL_RobotSpec で出力します：輪郭、内接円と外接円の半径、種類、body 要素の値。始めに 1 回だけ出力します。
robot_motion_limits	SMSL_RobotMotionLimits vector	ロボットの性能値の限界をカスタムストラクチャー SMSL_RobotMotionLimits（最大速度、最大加速度、最大減速度、最大角速度、傾きの最大角）で出力します。ファイルに motion 要素を書いたときだけ作られ、始めに 1 回だけ出力します。

プロパティ	**データ型**	**説明**
spec_file	Text	ロボットの仕様の XML ファイルのパス（UTF-8）。書き方はコンポーネントの説明を参照してください。

4. マクロコンポーネント

本パッケージは、補助マクロコンポーネントを提供します。これらから各コンポーネントを使用するヒントが得られます。よく編集されるプロパティは、マクロの property タブの Specific タブにリストされています。

SineCosineGenMacro



指定した角度の増分、周期、ゲインで正弦波と余弦波を出力するマクロコンポーネントです。

処理内容

- number_generator が角度の増分を周期的に出力し、サンプルごとに SineCosineGenerator の角度が angle_increment_in_deg だけ進みます。

- もう1つの number_generator が開始時にゲインを1回出力します。

使い方

- テスト信号として使います。例: Sine_1 → Add_Diff.input1（ノイズを加える）、Oscilloscope.inputA。

- SMSL_BasicToolkit_Macros.rtmc で定義されています。RobotPositioningToolkit のカルマンフィルタのサンプル（例: KalmanFilter_1D_demo.rtd）で使われています。

データ形式

- 出力: Float64 スカラー、1周期に1サンプル。

注意事項

- 名前に反して、sample_ratio は出力周期 [μs] です（既定値 10000 = 100 Hz）。

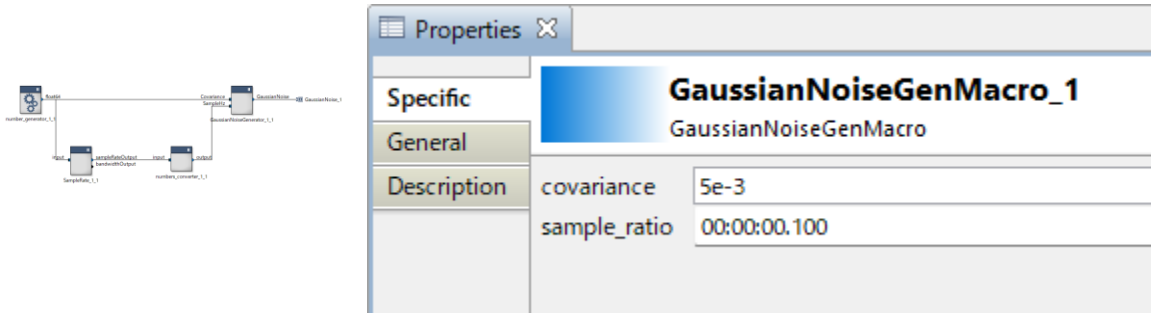
- 波の周波数は angle_increment_in_deg / 360 × (1e6 / sample_ratio) Hz です。

入力ポート	データ型	説明
-		

出力ポート	データ型	説明
Sine_1	Float64 vector	Gain･sin(θ)。
Cosine_1	Float64 vector	Gain･cos(θ)。

Macro プロパティ	データ型	説明
angle_increment_in_deg	Float64	出力1サンプルあたりの角度の増分 [deg]。（既定値: 2）
sample_ratio	Integer	出力周期 [μs]（内部の角度用 number_generator の周期）。（既定値: 10000）
gain	Float64	出力波形の振幅。（既定値: 1）

GaussianNoiseGenMacro



指定した共分散のガウスノイズを一定周期で出力するマクロコンポーネントです。

処理内容

- number_generator が共分散の値を周期的に出力し、そのサンプルごとに GaussianNoiseGenerator が動作します。

- ノイズの 1 サンプルごとの分散は、周期によらず共分散の値になります。

使い方

- 出力をきれいな信号に加えて観測ノイズを模擬します。例: GaussianNoise_1 → Add_Diff.input2。

- SMSL_BasicToolkit_Macros.rtmc で定義されています。RobotPositioningToolkit のカルマンフィルタのサンプル（例: KalmanFilter_1D_demo.rtd）で使われています。

データ形式

- 出力: Float64 スカラー、1周期に1サンプル。

注意事項

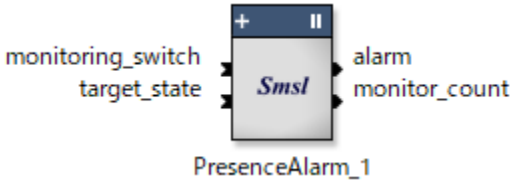
- 名前に反して、sample_ratio は出力周期 [μs] です（既定値 100000 = 10 Hz）。

入力ポート	データ型	説明
-		

出力ポート	データ型	説明
GaussianNoise_1	Float64 vector	ガウスノイズのサンプル。

Macro プロパティ	データ型	説明
covariance	Float64	ガウスノイズの共分散（分散）。（既定値: 5e-3）
sample_ratio	Integer	出力周期 [μs]（内部の number_generator の周期）。（既定値: 100000）

PresenceAlarm



見張り中に人が残っているときに警報を出すマクロコンポーネントです（置き去りの警報）。Hi_Link_LD2410C が解読したターゲットの状態を使います。

処理内容

- Condition_count（RTMaps の Condition）が、条件 (a != 0) * ((s >= 1 && s <= 3) * result + 1) でサンプル数を数えます。monitoring_switch が 0 の間は 0、見張り中で人がいないときは 1、人がいる状態（target_state が 1〜3）が n サンプル続くと n + 1 になります。

- Condition_alarm が、条件 c != 0 && (result != 0 || c > 200) で警報を保持します。数が 200 サンプル（センサの報告の周期 約 10 Hz で約 20 秒）を超えると警報が出て、monitoring_switch が 0 に戻るまで続きます。

- どちらの Condition も、入力が届くたびに評価され（Synchro tolerance = -1）、毎回条件の値を出力します。

使い方

- Hi_Link_LD2410C.target_state → target_state と、「このあとは誰もいないはず」を表す 0／1 のスイッチの信号（施錠、扉の閉のスイッチ、イグニッションのオフ、作業の終わり。試すときは Bi-state button）→ monitoring_switch をつなぎます。alarm は LED、ブザー、記録などにつなぎます。

- 見張りを始めたあとに検出の範囲から出た人は、センサの無人の時間（既定 5 秒）のあとに「いない」になり、数が 1 に戻ります。そのため、出ていく時間と無人の時間の

和が約 20 秒より短ければ警報は出ません。残っている人は（静止していても、呼吸で検出されて）警報になります。

- SMSL_BasicToolkit_Macros.rtmc で定義しています。

データ形式

- 入力・出力：Integer32 のスカラー。評価ごとに1サンプルを出力します。

注意事項

- 時間はサンプル数で数えるので、センサの報告の周期が変わると警報までの時間も変わります。閾値（200 サンプル）は Condition_alarm の条件の中にあります。マクロにプロパティはないので、変えるときはマクロ（.rtmc）を編集してください。

- target_state の 4〜6（背景ノイズの検出）は「いない」として扱います。

- センサの報告が止まると出力も止まり、最後の値のままになります。

- 安全のための装置ではありません（機能安全の認証はありません）。人を守る用途では、認証されたセンサの補助としてだけ使ってください。

入力ポート	データ型	説明
monitoring_switch	Integer32	見張りのスイッチ：0 以外（1）= オン、0 = オフ。オフにすると数と警報が戻ります。
target_state	Integer32	Hi_Link_LD2410C のターゲットの状態：0 なし、1 移動、2 静止、3 両方（4〜6：背景ノイズの検出。「いない」として扱います）。

出力ポート	データ型	説明

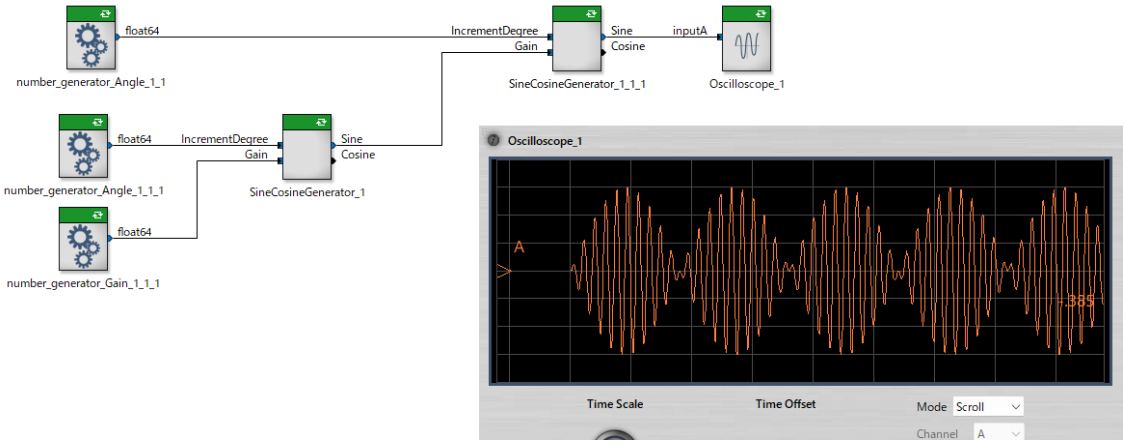
alarm	Integer32 vector	警報：見張り中に人がいる状態が 200 サンプルを超えて続くと 1 になり、monitoring_switch が 0 に戻るまで保持します。それ以外は 0。
monitor_count	Integer32 vector	数：0 = 見張りでない、1 = 見張り中で人がいない、n + 1 = 人がいる状態が n サンプル続いた。

プロパティ	データ型	説明
-		

5. デモダイヤグラム

本パッケージは、各コンポーネントに対する初歩的な RTMaps デモダイヤグラムを提供します。 それらの中で使い方の例が参照できます。

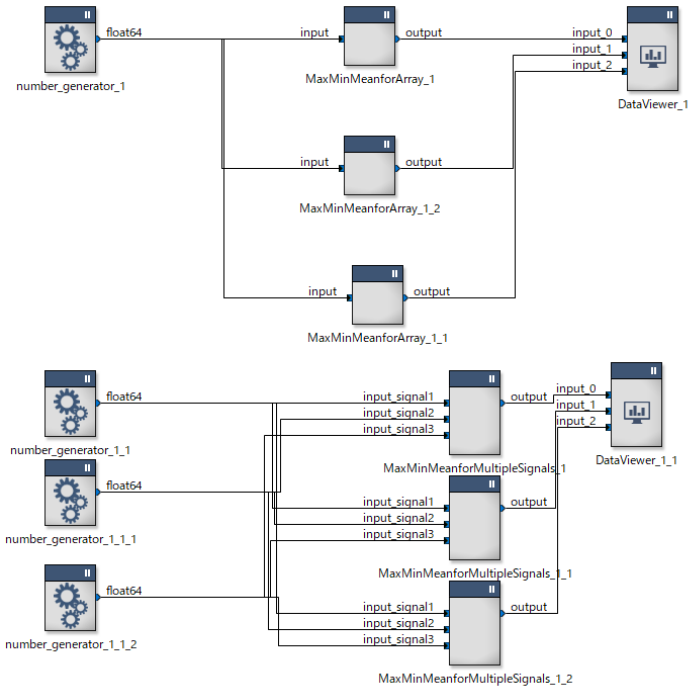
DSB_AM_demo



DSB_AM_demo.rtd は、二つの SineCosineGenenerator コンポーネントを使用して、ダブルサイドバンド AM を生成する例です。

初段のジェネレータは二段目のジェネレータの変調振幅を規定しています。

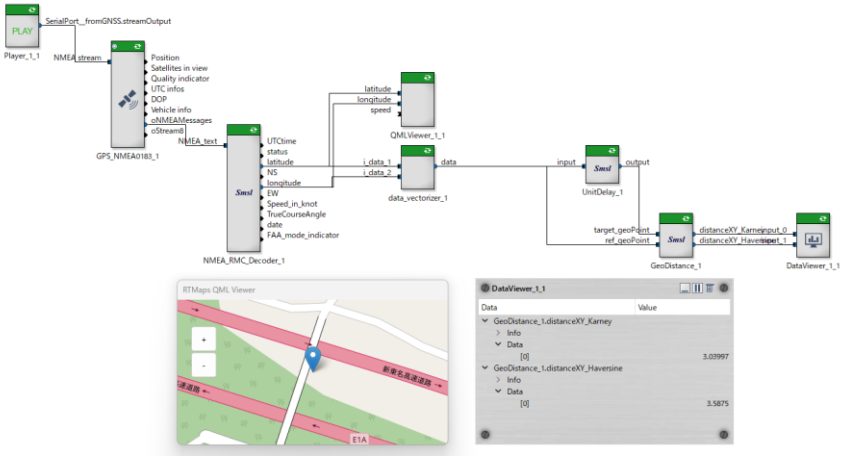
MaxMinMean_demo



MaxMinMean_demo.rtd は、数値データの基本的な計算の例です。

入力データのベクトル信号と複数信号の配置の違いによって、２つのバリエーションが用意されています。

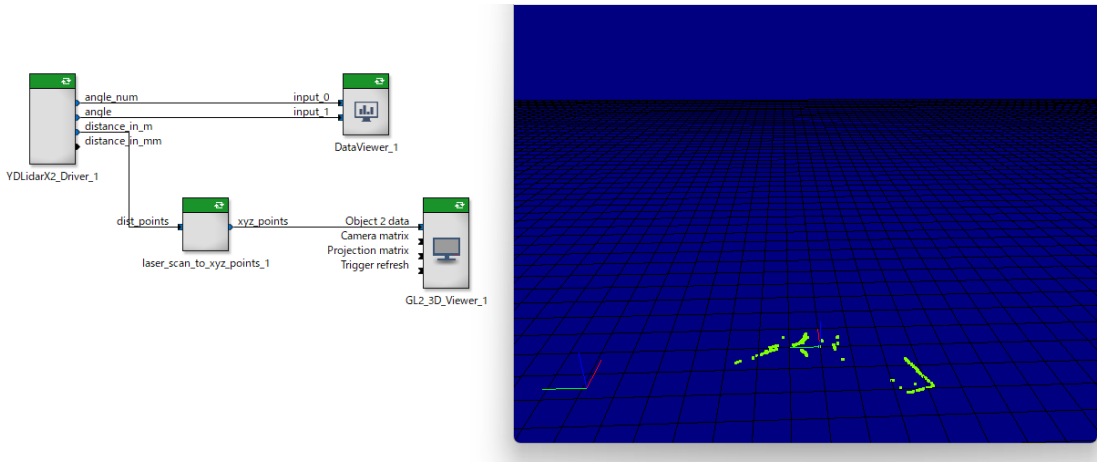
GeoDistance_demo



GeoDistance_demo.rtd is 地理的な緯度経度の二点間の距離を計算する例です。入力信号は動いている地理的な点とすると、ダイヤグラムは UnitDelay を使用してステップごとの移動距離を計算します。

残念ながら入力信号は現在のところ SMSL によって提供されませんので、GNSS からデータを得る必要があります。

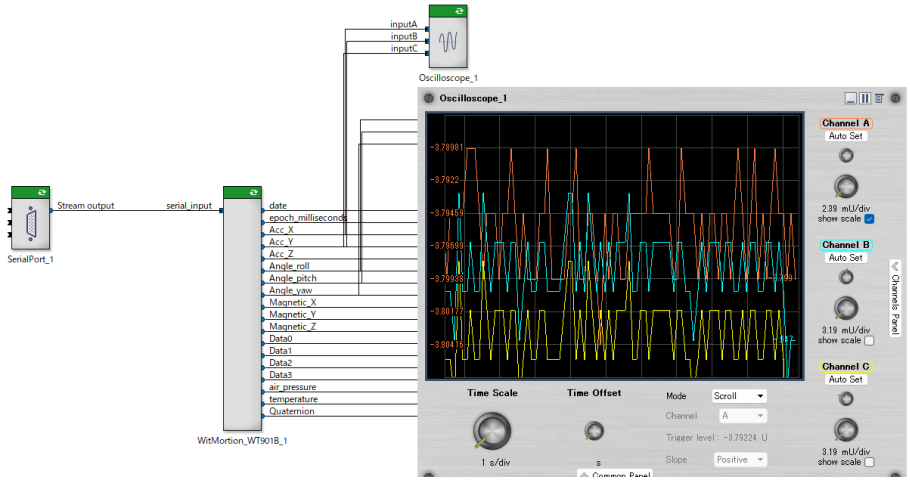
YDLidarDataAcquisition_demo



YDLidarDataAcquisition_demo.rtd は YDLidar X2からデータを取得する例です。

ドライバコンポーネントから、laserscan としての一次元配列信号が得られます。

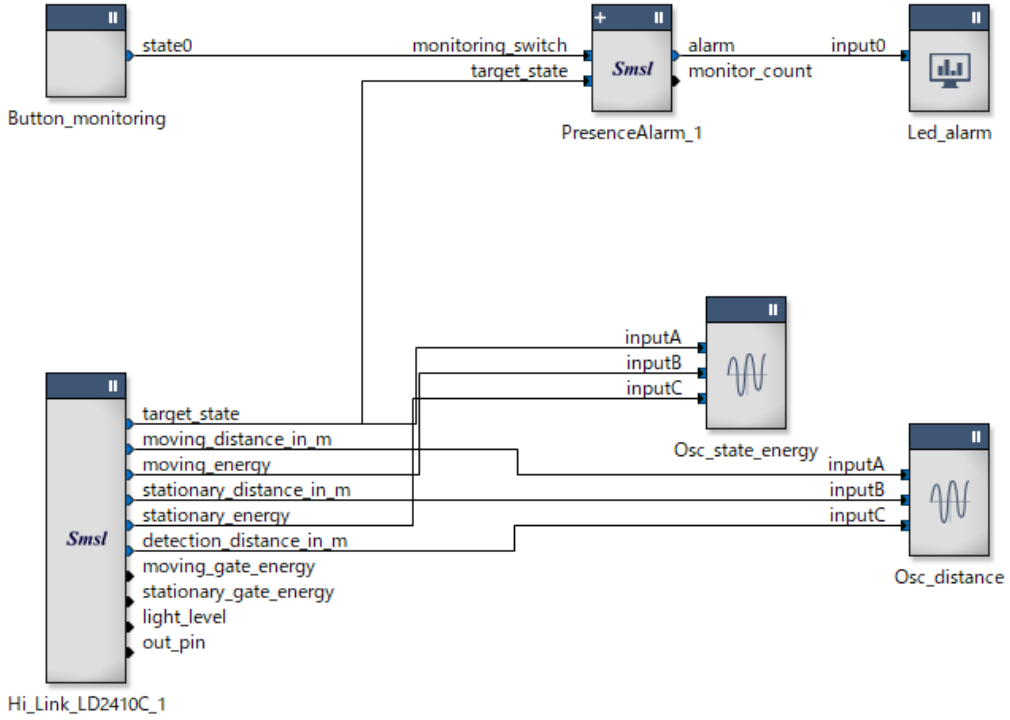
WitMotion_WT901B_imu_demo



WitMotion_WT901B_imu_demo.rtd は、WT901B IMU からデータを取得する例です。

デコーダコンポーネントから、IMU の加速度/角度値が得られます。

Hi_Link_LD2410C_demo



Hi_Link_LD2410C_demo.rtd は、Hi-Link LD2410C（24GHz ミリ波人感レーダー）の検出結果を表示する例です。Hi_Link_LD2410C コンポーネントがデコードした目標の状態・距離・エネルギーを、Oscilloscope（距離、目標の状態とエネルギー）と DataViewer（ゲートごとのエネルギー、光センサ、OUT ピンを含むすべての出力）で表示します。

LD2410C は人の有無（動いている／静止している）とおおよその距離（ゲート幅 0.75 m、約 6 m まで）を出す人感センサで、壁や物までの距離は測れません。移動体では、次のように停止中に使います。

- 発進前の周囲の確認（AGV・AMR・作業車両）：停止中に、近くにしゃがんでいる人や立ち止まっている人がいないかを確かめてから発進する。電波は樹脂のカバーを通るので、車体の内側に取り付けられる。

- 人が近づいたときの合図（受付・配送ロボット）：停止して待っている間に、人が近づいたら案内を始める、扉を開けるなどの動作のきっかけにする。

- 車内モニター：降車後の車内に人が残っていないかを、呼吸のわずかな動き（静止の検出）で確かめる。車内の距離は 2 m 程度なので、最大ゲートを 2〜3 にすると車外を通る人に反応しにくい。

センサが動くと、壁や床も動いている目標に見えて誤検出が増えるため、走行中には使えません。安全機器（機能安全の認証品）ではないため、人の保護に使うときは認証されたセンサの補助にとどめてください。

6. 結論

ここまでで、SMSL_BasicToolkit パッケージの全体像を理解いただきました。著者は使用者が SMSL_BasicToolkit で RTMaps アプリケーションの素晴らしい体験を得ることを期待しています。

7. 更新履歴

Version	Date	Author	説明
1.0	2025 12/25	伊田 祥吾	初版発行
1.1	2026 10/16	伊田 祥吾	バージョンを1.1に更新しました。Hi_Link_LD2410C、bits2Byte、Byte2bits、NumericalTextWriter を追加しました。SineCosineGenerator、WitMotion_WT901B を修正しました。
1.2	2026 10/18	伊田 祥吾	バージョンを1.2に更新しました。Hi_Link_LD2410C に目標の状態・距離・エネルギーの出力、engineering_mode、検出設定のプロパティを追加しました。YDLidarX2_Driver にプロパティ Inverted を追加しました。Hi_Link_LD2410C、YDLidarX2_Driver を修正しました。

1.3	2026 10/25	伊田 祥吾	バージョンを1.3に更新しました。 Byte2bits、GaussianNoiseGenerator、 GeoDistance、Hi_Link_LD2410C、 MaxMinMeanforMultipleSignals、 NumericalTextWriter、 SineCosineGenerator、 WitMotion_WT901B、 YDLidarX2_Driver、bits2Byte を修正しました。
1.31	2026 11/1	伊田 祥吾	バージョンを1.31に更新しました。 RobotSpecManager、 SampleHistoryVectorizer、 SkipEarlyFrames、マクロ PresenceAlarm を追加しました。 UnitDelay の、8 バイトでない型でバッファの外に書き込む不具合、入力の長さが変わると値が混ざる不具合、出力のタイムスタンプ（入力のものにする）を修正しました。
1.32	2026 11/2	伊田 祥吾	バージョンを1.32に更新しました。 RobotSpecManager の出力 robot_motion の名前を robot_motion_limits に、設定ファイルの要素 <motion> の名前を <motion_limits> に変更しました。
1.33	2026 11/7	伊田 祥吾	バージョンを1.33に更新しました。 imu_orientation_estimator、 imu_to_pose6D を追加しました。 StaticImageGenerator に output_mode（Once / Repeat）を追加しました。

			WitMotion_WT901B に出力 Gyro_X、Gyro_Y、Gyro_Z を追加しました。