

JEM 船内可搬型ビデオカメラシステム実証 2 号機

INT-BALL2

技術実証プラットフォーム ユーザマニュアル



NC: 2024/5/8

宇宙航空研究開発機構 (JAXA)

有人宇宙技術センター

目次

1. 本文書の目的	3
2. 関連文書.....	3
3. Int-Ball2 技術実証プラットフォームの概要	3
4. 環境構築手順	7
4.1 動作環境.....	7
4.2 OS インストール	7
4.3 ROS インストール	7
4.4 Python3 インストール	7
4.5 地上運用支援ツール.....	8
4.5.1 Nasm (アセンブラ).....	8
4.5.2 映像受信環境	8
4.5.3 VLC media player のインストール.....	10
4.5.4 Qt.....	11
4.5.5 フォントファイル	13
4.5.6 ソースコード配備	13
4.5.7 パラメータ設定.....	14
4.6 Int-Ball2 技術実証プラットフォームシミュレータ	14
4.6.1 Docker.....	14
4.6.2 Python	14
4.6.3 コンテナ起動設定	14
4.6.4 ソースコード配備	15
4.6.5 パラメータ設定.....	15
4.6.6 Docker 設定.....	15
4.6.7 コンテナのセットアップ	16
4.7 ユーザプログラムの配備	16
4.8 ユーザプログラム用 roslaunch	17
4.9 ビルド手順.....	18
4.9.1 Int-Ball2 GSE.....	18
4.9.2 Int-Ball2 技術実証プラットフォーム.....	18
5. 操作手順.....	19
6. 問い合わせ先	20

1. 本文書の目的

JEM 船内可搬型ビデオカメラシステム実証 2 号機(以下、Int-Ball2) は地上からの遠隔操作により JEM 船内を飛行し移動することで映像撮影を行いクルーの撮影作業を代替するカメラロボットである。加えて Int-Ball2 はその拡張機能としてユーザが開発した任意のソフトウェアを実行し、宇宙空間でロボット技術の実証のプラットフォームとして使用できる。本書ではきぼうロボットプログラミングチャレンジ(K-RPC)等での使用を想定して技術実証プラットフォームを使う際の環境整備方法をマニュアルとしてまとめる。

2. 関連文書

- (1) JX-ESPC-101815 JEM 船内可搬型ビデオカメラシステム実証 2 号機(Int-Ball2)
総合システム開発仕様書
- (2) JMX-2018280 JEM 船内可搬型ビデオカメラシステム実証 2 号機(Int-Ball2)
利用運用コンセプト
- (3) JDX-2020224 JEM 船内可搬型ビデオカメラシステム実証 2 号機(Int-Ball2)
総合システム ユーザプログラミング機能要求仕様書
- (4) JDX-2020225 JEM 船内可搬型ビデオカメラシステム実証 2 号機(Int-Ball2)
シミュレータユーザプログラミング機能要求仕様書

3. Int-Ball2 技術実証プラットフォームの概要

Int-Ball2 フライトソフトウェアでは技術実証環境対応としてユーザに対して Int-Ball2 が有するセンサデータとアクチュエータへの制御入力インタフェースを提供する。これにより、ユーザは独自の航法機能 (Visual SLAM、センサフュージョンなど) や誘導制御機能 (PID 制御、ビジュアルフィードバック制御)等を実装し、Int-Ball2 のフライトソフトウェアをカスタマイズすることが可能となる。また、ユーザが実装したプログラムを Int-Ball2 機体に適用する際は既存機能の一部と連携させることも可能であり、ユーザが技術実証として実装する機能の範囲を調整することが可能である。

技術実証プラットフォームは以下により構成される

- 技術実証プラットフォーム S/W
- 技術実証プラットフォーム 地上運用支援ツール
- 技術実証プラットフォーム シミュレーション

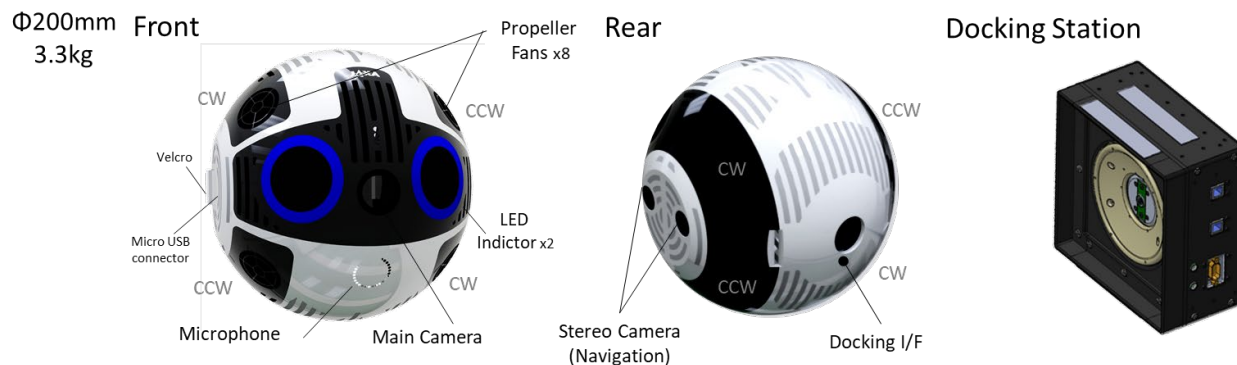


図 1: Int-Ball2 機体概要

Int-Ball2 フライトソフトウェアの各機能は ROS(Robot Operating System)のノードとして実装されており、既存ソースコードの改修は実施せずに設定変更のみで、技術実証範囲に応じた機能(ノード)の起動有無を制御可能である。また、既存機能におけるデータ出力や機能呼び出しの内部インタフェースは、ROS のメッセージ形式(Topic, Service, Action)で統一されているため、新規機能追加の際も、実装済みの内部通信インタフェースを利用することで既存機能の出力データや処理を利活用可能である。

Int-Ball2 技術実証プラットフォームは、既存 Int-Ball2 フライトソフトウェアと同様の通信ラインを介して軌道上と地上間でテレコマ送受信を行い運用される。(図 2、図 3 参照)

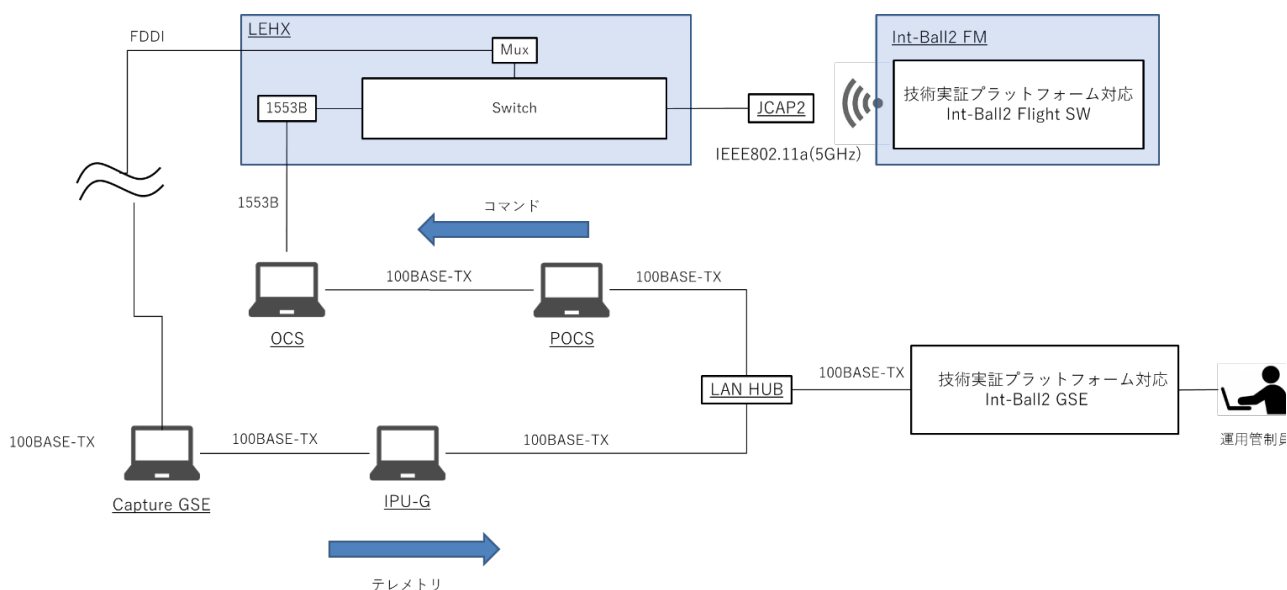


図 2: Int-Ball2 技術実証プラットフォーム 軌道上運用時の通信経路とシステム構成

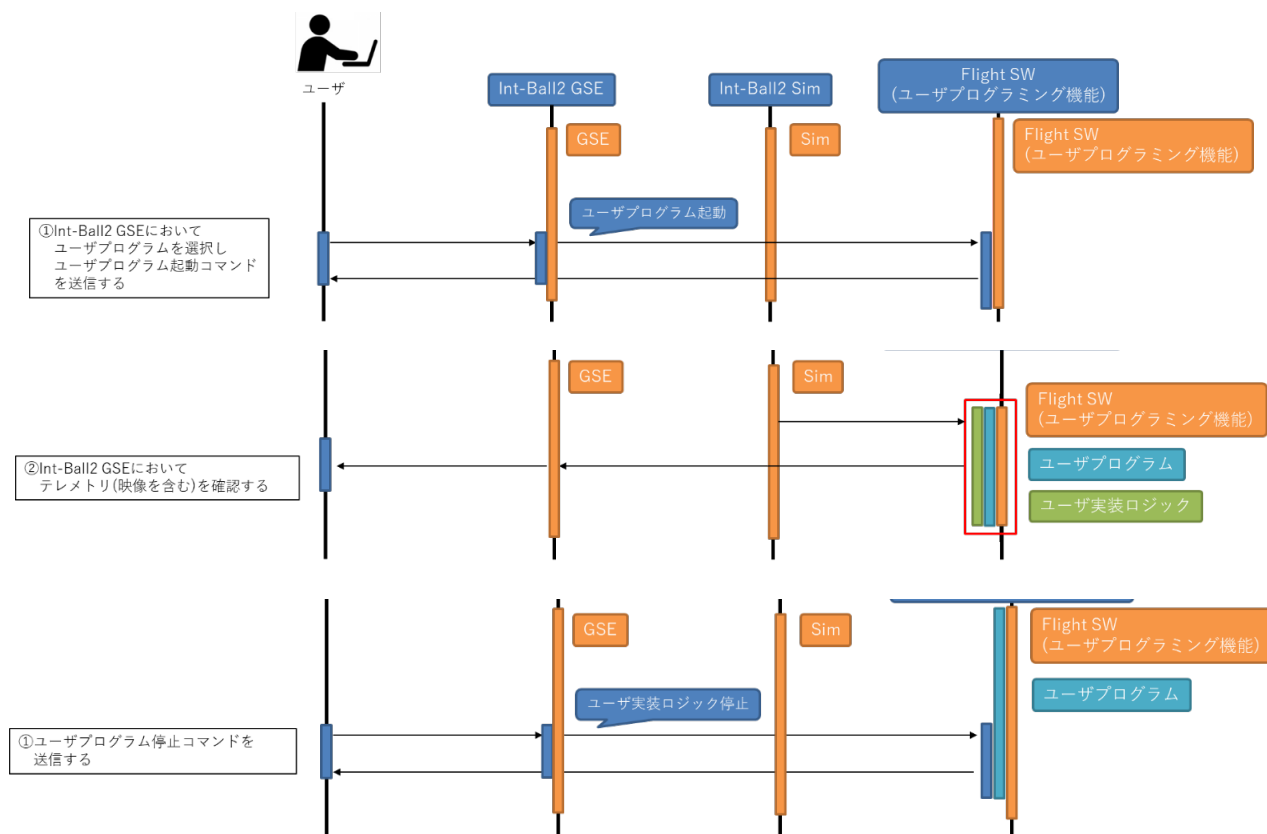


図 3: ユーザプログラム起動の流れ

実証プラットフォーム機能は Docker 上で動作し、ネイティブの環境とは仮想的に分けられている。ユーザプログラムは運用管制員により Int-Ball2 機体にアップリンクされ、地上運用支援ツールを介して起動させることにより実行される。これにより既存のフライトソフトウェアおよび地上運用支援ツールの機能に影響を与えずに、新規なプログラム・ロジックの追加・置換を行い Int-Ball2 上で技術実証を行うことが可能である。

加えてこれらのプログラムは Int-Ball2 のシミュレーション環境でも動作可能である。(図 4)

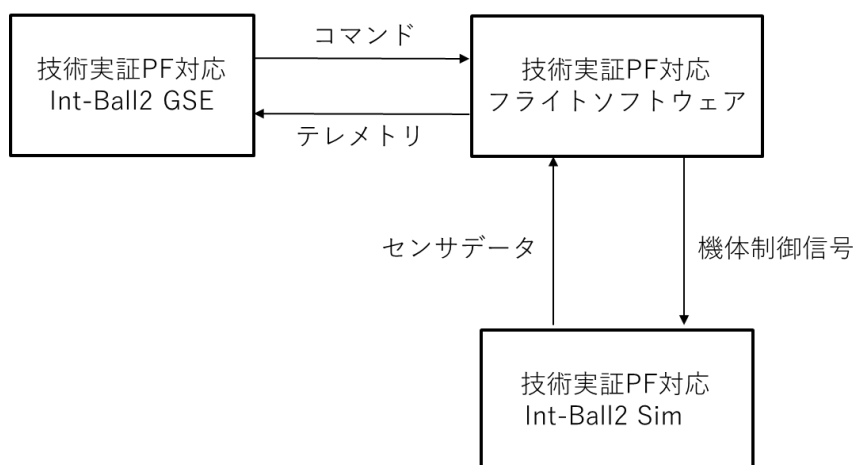


図 4: シミュレーション環境での技術実証プラットフォーム動作

Int-Ball2 における既存フライトソフトウェアのシステム構成概要を図 5 に示す。

【本実証プラットフォームを使用するにあたり考慮が必要な制約】

- 実証プラットフォーム上で動作するソフトウェアは Docker にて起動する。起動コマンドは地上から送信される。基本実証中に地上のユーザーソフトウェアとのリアルタイムでの通信はしない(通信が必要な実証ソフトウェアを実装する場合は Int-Ball2 原局と調整すること)。
- 実証プログラムのサイズについては Int-Ball2 原局と調整すること(Int-Ball2 機体及び SD カード容量の空き状況による)。

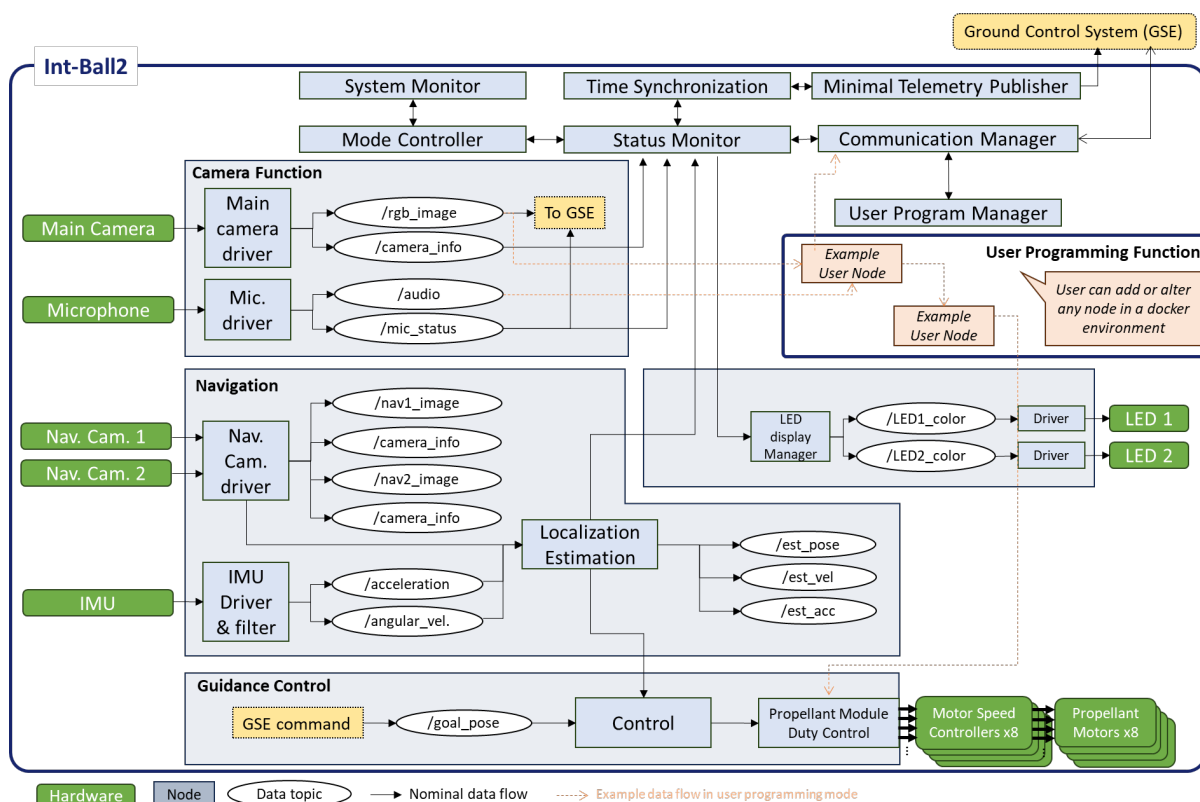


図 5: Int-Ball2 フライトソフトウェアユーザプログラミング機能システム構成図

※現状では航法カメラの映像を別途使用する場合、Int-Ball2 に実装されている既存の Visual-SLAM(自己位置姿勢推定機能)を使用することができない。(航法カメラの映像を使用する場合は自身で自己位置姿勢推定の実装が必要である。)

4. 環境構築手順

4.1 動作環境

- OS: Ubuntu18.04
- ミドルウェア: Robot Operating System(ROS1/Melodic)
- Int-Ball2 機体処理装置: Nvidia Jetson TX2, Linux for Tegra (Ubuntu Based File System)
 - シミュレータ使用の場合は PC 環境のみで可

4.2 OS インストール

Int-Ball2 技術実証プラットフォームは Ubuntu18.04 上で動作する。

以下サイトを参考に Ubuntu18.04 を自身の環境にインストールする。

[Ubuntu 18.04.6 LTS \(Bionic Beaver\)](#) (2024/3/28 時点)

4.3 ROS インストール

Int-Ball2 技術実証プラットフォームおよびそのシミュレータは ROS/Gazebo を用いて実行する。このためこれらをインストールする。なお、インストールを行う際、Gazebo のバージョンが 9.0.0 より高いものを利用する必要があるため、以下のような手順でインストールを実施する。

1. Gazebo9 最新版取得準備 Gazebo9 の最新版を取得可能とするため以下サイトの「Alternative installation: step-by-step」における 1 および 2 を実行する。

[Gazebo : Tutorial : Ubuntu \(gazebosim.org\)](#) (2024/3/28 時点)

2. ROS インストール 以下サイトにおける 1.1 から 1.6 までの手順を実施し、ROS をインストールする
[melodic/Installation/Ubuntu – ROS Wiki](#) (2024/3/28 時点)

1.4.Installation では「sudo apt install ros-melodic-desktop」を選択する。

3. ROS 関連パッケージのインストール

以下のコマンドで ROS-Gazebo 連携のためのパッケージをインストールする

```
$ sudo apt install ros-melodic-gazebo-*
```

4.4 Python3 インストール

本システムでは、Python3 および関連パッケージを利用するため関連パッケージのインストールおよび設定ファイルを変更する。

1. Python3 をインストールする。

```
$ sudo apt install python3 python3-pip
```

2. Python3 の ROS 関連パッケージをインストールする。

```
$ pip3 install rospkg  
$ pip3 install empy==3.3.4
```

※pip3 install rospkg empy で empy をインストールすると、最新版(執筆時点では 4.1)がインストールされる。しかしこのバージョンは参考サイト(<http://soup01.com/ja/2023/12/25/post-9789/>)に記載があるように、`AttributeError: 'module' object has no attribute 'RAW_OPT'`というエラーを引き起こす不具合が確認されている。そのためバージョンを 3.3.4 に指定してインストールする必要がある。

3. ROS パッケージのビルド(catkin)で利用する Python2 を Python3 に変更する。もし「1.ros_python_version.sh」が存在しない場合は設定ファイルを新規作成し、変数を追記する。

```
$ sudo nano /opt/ros/melodic/etc/catkin/profile.d/1.ros_python_version.sh
...
export ROS_PYTHON_VERSION=3                # ← 1 行追記(新規作成)
...
```

4.5 地上運用支援ツール

技術実証プラットフォームのソフトウェアの実行は Int-Ball2 の地上運用支援ツール(GSE)から行われる。以下の手順で Int-Ball2 GSE 動作環境を整備する。

4.5.1 Nasm (アセンブラ)

nasm アセンブラを以下手順でソースファイルからビルド&インストールする。

1. nasm のソースファイルを/usr/local/src は以下に解凍して配置する。

```
$ cd /usr/local/src
$ sudo wget https://www.nasm.us/pub/nasm/releasebuilds/2.15.05/nasm-2.15.05.tar.gz
$ sudo tar zxvf nasm-2.15.05.tar.gz
```

2. 配置したソースファイルのディレクトリに移動し、ビルド設定を行う。

```
$ cd nasm-2.15.05
$ sudo ./configure
```

3. ソースファイルをビルドしてインストールする。

```
$ sudo make install
```

※参考記事(<https://trans-it.net/centos7-ffmpeg43-h264-fdkaac/>)にあるように、FFmpeg をインストールするには先に nasm をインストールする必要がある。

4.5.2 映像受信環境

x264 に関して以下手順でソースファイルからビルドを行う。

1. (オプション 1)ソースファイルを/usr/local/src 配下に解凍して配置する。


```
# x264-master.tar.bz2 をダウンロードしてくる。
$ sudo tar jxvf x264-master.tar.bz2 -C /usr/local/src/
```

(オプション 2) もしくはリポジトリより下記の通り入手する

```
$ sudo git clone https://code.videolan.org/videolan/x264.git \
/usr/local/src/x264-master
```

2. 配置したソースファイルのディレクトリに移動し、ビルド設定を行う。

```
$ cd /usr/local/src/x264-master
$ sudo ./configure --disable-asm --enable-shared --enable-static --enable-pic
```

3. ソースファイルをビルドしてインストールする。

```
$ sudo make install
```

4. ffmpeg のソースファイルを /usr/local/src 配下に配置する。

(オプション1) ソースファイルを入手し解凍

```
# ffmpeg-4.1.3.tar.gz をダウンロードしてくる
$ sudo tar xvzf ffmpeg-4.1.3.tar.gz -C /usr/local/src/
```

(オプション 2) Github より入手

```
$ sudo git clone https://github.com/FFmpeg/FFmpeg.git -b n4.1.3 \
/usr/local/src/ffmpeg-4.1.3
```

5. 配置したソースファイルのディレクトリに移動し、ビルド設定を行う。

```
$ cd /usr/local/src/ffmpeg-4.1.3
$ sudo ./configure \
--extra-cflags="-I/usr/local/include" \
--extra-ldflags="-L/usr/local/lib" \
--extra-libs="-lpthread -lm -ldl -lpng" \
--enable-pic \
--disable-programs \
--enable-shared \
--enable-gpl \
--enable-libx264 \
--enable-encoder=png \
--enable-version3
```

6. ソースファイルをビルドしてインストールする。

```
$ sudo make install
```

4.5.3 VLC media player のインストール

VLC media player に関して以下手順でソースファイルからビルドを行う。

1. 依存パッケージをインストールする。

```
$ sudo apt install libasound2-dev libxcb-shm0-dev libxcb-xv0-dev \
libxcb-keysyms1-dev libxcb-randr0-dev libxcb-composite0-dev \
lua5.2 lua5.2-dev protobuf-compiler bison libdvbpsi-dev libpulse-dev
```

2. ダウンロードしたソースファイルを/usr/local/src 配下に解凍して配置する。
(オプション 1) ソースファイルを手し解凍

```
$ sudo wget https://repo.jing.rocks/videolan/vlc/3.0.7.1/vlc-3.0.7.1.tar.xz
$ sudo tar Jxvf vlc-3.0.7.1.tar.xz -C /usr/local/src/
```

(オプション 2) Github から入手

```
$ sudo git clone https://github.com/videolan/vlc.git -b 3.0.7.1 \
/usr/local/src/vlc-3.0.7.1
```

3. 配置したソースファイルのディレクトリに移動し、ビルド設定を行う。

```
$ cd /usr/local/src/vlc-3.0.7.1
# ビルド用変数設定および configure を 1 行にまとめて実行する
$ CFLAGS="-I/usr/local/include" \
LDLFLAGS="-L/usr/local/lib" \
X264_CFLAGS="-L/usr/local/lib -I/usr/local/include" \
X264_LIBS="-lx264" \
X26410b_CFLAGS="-L/usr/local/lib -I/usr/local/include" \
X26410b_LIBS="-lx264" \
AVCODEC_CFLAGS="-L/usr/local/lib -I/usr/local/include" \
AVCODEC_LIBS="-lavformat -lavcodec -lavutil" \
AVFORMAT_CFLAGS="-L/usr/local/lib -I/usr/local/include" \
AVFORMAT_LIBS="-lavformat -lavcodec -lavutil" \
sudo ./configure \
--disable-a52 \
--enable-merge-ffmpeg \
--enable-x264 \
--enable-x26410b \
```

```
--enable-dvbpsi
```

4. ソースファイルをビルドしてインストールする。

```
$ sudo make install
```

5. ダウンロードしたソースファイルに対するシンボリックリンクを作成する。

```
$ sudo ln -s /usr/local/src/vlc-3.0.7.1 /usr/local/src/vlc
```

4.5.4 Qt

Qt に関して以下手順でインストールを実施する。

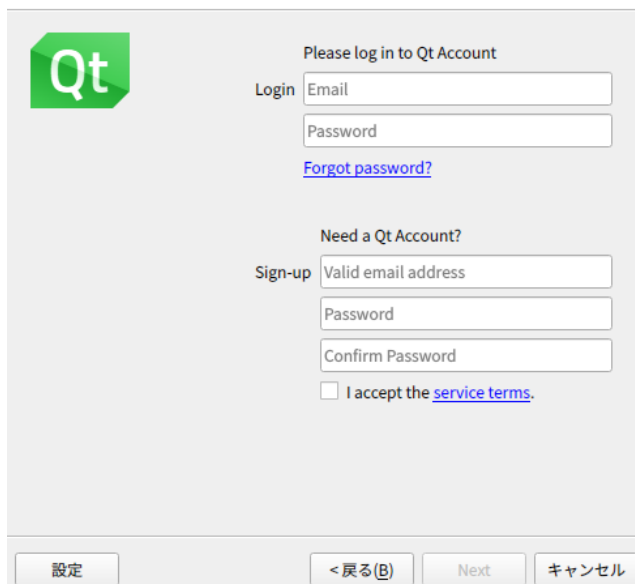
1. インストールディレクトリを作成する

```
$ sudo mkdir /opt/Qt
```

2. ダウンロードした Linux 用インストーラを起動する(インストール中にアカウント登録が必要となる)

```
$ wget https://download.qt.io/archive/qt/5.12/5.12.3/qt-opensource-linux-x64-5.12.3.run
$ chmod +x qt-opensource-linux-x64-5.12.3.run
$ ./qt-opensource-linux-x64-5.12.3.run
```

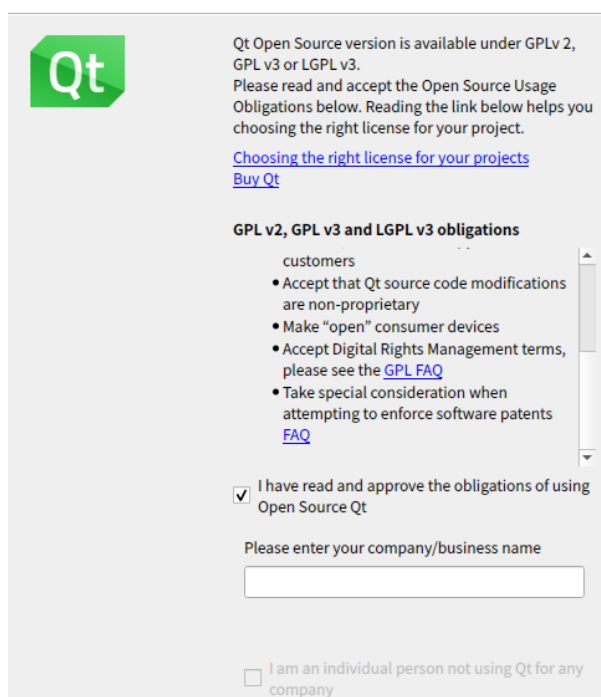
Qt Account - Your unified login to everything Qt



The image shows a Qt Account login and sign-up window. On the left is the Qt logo. The main area is divided into two sections: 'Please log in to Qt Account' and 'Need a Qt Account?'. The login section has fields for 'Email' and 'Password', and a link for 'Forgot password?'. The sign-up section has fields for 'Valid email address', 'Password', and 'Confirm Password', and a checkbox for 'I accept the service terms.' with a link to 'service terms.'. At the bottom are four buttons: '設定' (Settings), '< 戻る(B)' (Back), 'Next', and 'キャンセル' (Cancel).

3. Open Source Qt の利用にチェックを付ける。

Qt Open Source Usage Obligations



Qt Open Source version is available under GPLv 2, GPLv 3 or LGPL v3.
Please read and accept the Open Source Usage Obligations below. Reading the link below helps you choosing the right license for your project.

[Choosing the right license for your projects](#)
[Buy Qt](#)

GPL v2, GPL v3 and LGPL v3 obligations

customers

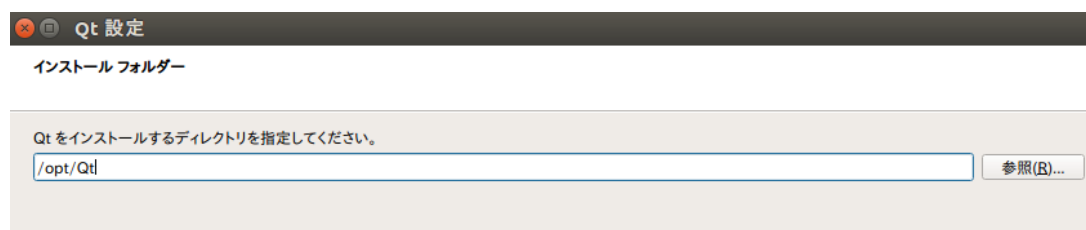
- Accept that Qt source code modifications are non-proprietary
- Make "open" consumer devices
- Accept Digital Rights Management terms, please see the [GPL FAQ](#)
- Take special consideration when attempting to enforce software patents [FAQ](#)

☒ I have read and approve the obligations of using Open Source Qt

Please enter your company/business name

☐ I am an individual person not using Qt for any company

4. インストールディレクトリの指定は/opt/Qt とする。



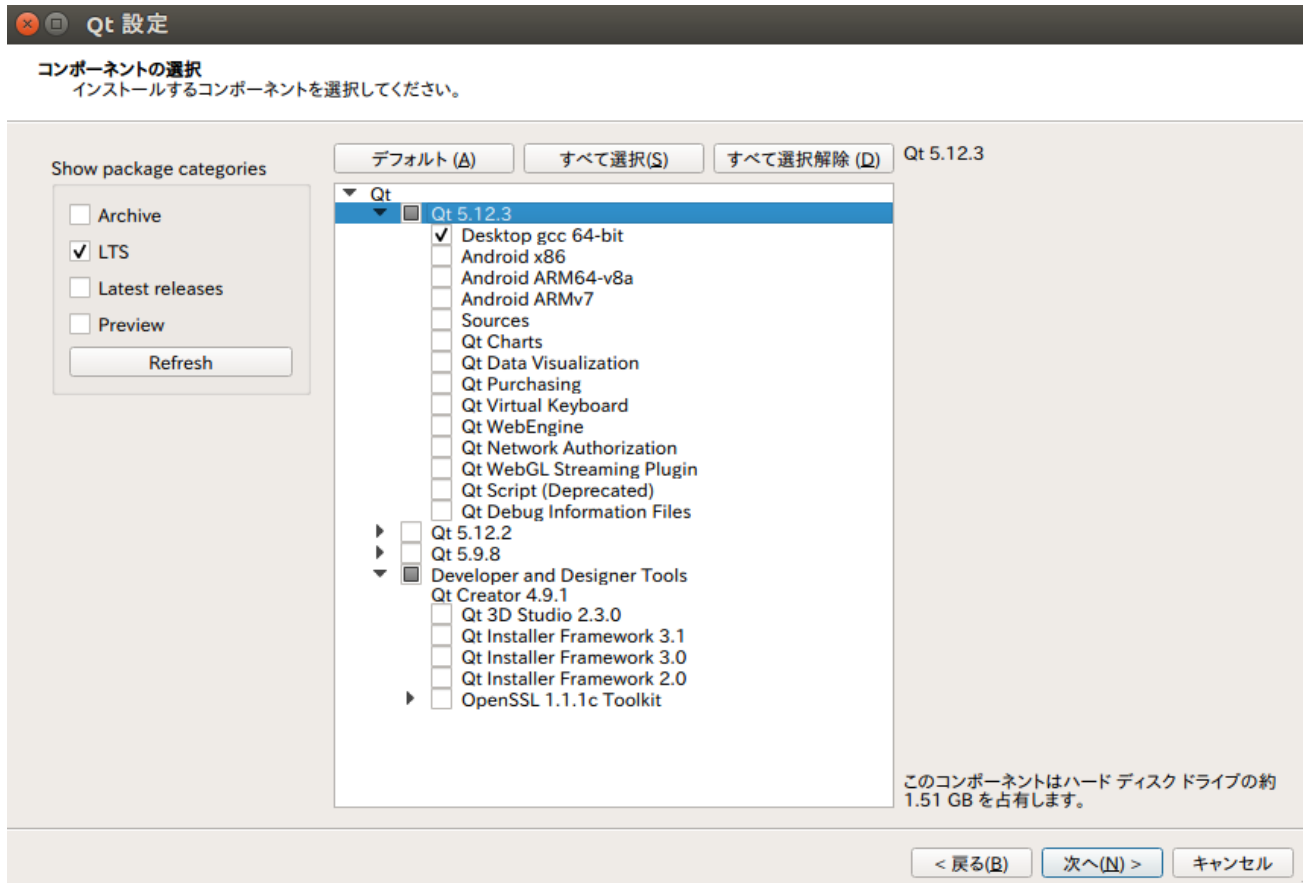
Qt 設定

インストール フォルダー

Qt をインストールするディレクトリを指定してください。

[参照\(R\)...](#)

5. インストール対象バージョンは、5.12.3 の Desktop gcc 64-bit とし以降はインストーラの指示に従う。



6. インストール完了後、シンボリックリンクを作成する。

```
$ sudo ln -s /opt/Qt/5.12.3 /opt/Qt/5
```

4.5.5 フォントファイル

Int-Ball2 GSE では Roboto フォントを用いているため、インストールを実施する。

(オプション1) ソースファイルを入手しコピーする

```
$ sudo cp Roboto*.ttf /usr/local/share/fonts/.
$ fc-cache -f -v
```

(オプション2) 標準リポジトリから入手する

```
$ sudo apt install fonts-roboto
```

4.5.6 ソースコード配備

「Int-Ball2_platform_gse」を任意ディレクトリ(例: /home/nvidia/IB2)に展開する。

4.5.7 パラメータ設定

Int-Ball2 技術実証プラットフォームシミュレータと GSE 間でテレメトリおよびコマンドのやり取りを行うため、以下のように通信設定のパラメータを設定する。

- コマンド送信設定

[Int-Ball2_platform_gse]/src/ground_system/communication_software/config/params.ymlにおける「intball2_telecommand_target_ip」に本システムを動作させるPCのIPアドレス(例:127.0.0.1)を設定する。また、「intball2_telecommand_target_port」に任意のコマンド送信ポート(例:23456)を設定する。

- テレメトリ受信設定

[Int-Ball2_platform_gse]/src/ground_system/communication_software/config/params.yml における「intball2_telemetry_receive_port」に任意のテレメトリ受信ポート(例:34567)を設定する。

4.6 Int-Ball2 技術実証プラットフォームシミュレータ

Int-Ball2 技術実証プラットフォームシミュレータを利用するためその動作環境を整備する。

4.6.1 Docker

以下サイトの手順を参考に Docker をインストールする。

[Install Docker Engine on Ubuntu | Docker Docs](#) (2024/3/28 時点)

「Install using the apt repository」の 1～2 を実行する。

4.6.2 Python

Python3 において Docker 関連パッケージをインストールする。

```
$ pip3 install docker defusedxml netifaces
```

4.6.3 コンテナ起動設定

コンテナ用サービスを常時有効とする。

```
$ sudo systemctl enable docker docker.socket
$ sudo systemctl start docker.socket
```

本システムを実行するユーザに対して、コンテナ起動用の権限を付与する。以下、対象ユーザが「nvidia」として手順を示す。

```
# 権限設定
$ sudo gpasswd -a nvidia docker
$ sudo chgrp docker /var/run/docker.sock
# コンテナ用サービスの再起動
$ sudo service docker restart
```

```
# この後、再ログインもしくは再起動をすること
```

本システムを実行するユーザで以下コマンドを実行し、「permission denied」という文言を含むエラーが発生しないことを確認する。SSH 接続で操作している場合には SSH を一度切断し、再接続した後に操作すること。

```
docker ps
```

4.6.4 ソースコード配備

「Int-Ball2_platform_simulator」を任意ディレクトリ(例: /home/nvidia/IB2)に展開する。

また、同じく「platform_works」を、本システム実行ユーザのホーム配下(~/)に展開する。

4.6.5 パラメータ設定

Int-Ball2 技術実証プラットフォームシミュレータと GSE 間でテレメトリおよびコマンドのやり取りを行うため、以下のように通信設定のパラメータを設定する。

- コマンド送信設定

[Int-Ball2_platform_simulator]/src/flight_software/trans_communication/launch/bringup.launchにおける「receive_port」に「4.5.7 パラメータ設定」で設定した「intball2_telecommand_target_port」と同じ値(例:23456)を設定する。

- テレメトリ受信設定

[Int-Ball2_platform_simulator]/src/flight_software/trans_communication/launch/bringup.launchにおける「ocs_host」に、本システムを動作させるPCのIPアドレス(例: 127.0.0.1、localhost)を設定する。また、「ocs_port」に5.4.7.1で設定した「intball2_telemetry_receive_port」と同じ値(例:34567)を設定する。

4.6.6 Docker 設定

Int-Ball2 技術実証プラットフォームは docker を用いてユーザプログラムを実行する。この際のホストとコンテナのやり取りに関する設定に関して以下に示す。

1. ホスト PC の IP

[Int-Ball2_platform_simulator]/src/platform_sim/platform_sim_tools/launch/platform_manager_bringup.launch における「container_ros_master_uri」に、本システムを動作させる PC の IP アドレス(例:http://127.0.0.1:11311)を設定する。

2. ホスト PC におけるコンテナワークスペース

[Int-Ball2_platform_simulator]/src/platform_sim/platform_sim_tools/launch/platform_manager_bringup.launch における「host_ib2_workspace」に Int-Ball2_platform_simulator の置かれているディレクトリパスを設定する。(例:/home/nvidia/IB2)

4.6.7 コンテナのセットアップ

技術実証プラットフォームではコンテナ上でユーザプログラムを実行するため、ユーザ実証プラットフォームを実行するためのコンテナビルドを実行する。

なお、タグ名 (**ib2_user**) 部分は任意に設定可能だが、フライトソフトウェアで取り扱うコンテナについては、必ずプレフィックス「ib2」を持つタグ名とする必要がある。

```
$ cd ~/platform_works/platform_docker/template
$ docker build . -t ib2_user:0.1
```

※Dockerイメージはクロスプラットフォーム非対応であり、別のCPUアーキテクチャで作成したものを使用することができない(通常のPCはIntel/AMD、Int-Ball2はARM)。したがってInt-Ball2実機で動作させるためのイメージを作成する場合は上記コマンドではなく、以下のコマンドのようにDocker buildxという拡張プラグインを用いてARM用のイメージを生成する必要がある。

参考サイト(<https://qiita.com/englishoma/items/0c7c6c3ae3dc173b01b2>)

```
$ docker buildx build --platform linux/arm64/v8 -t ib2_user:(version) --load .
```

ビルド完了後、以下コマンドを実行し、REPOSITORY に **ib2_user** が表示されることを確認する。

```
$ docker images ib2_user
```

4.7 ユーザプログラムの配備

技術実証プラットフォームにおいて利用するユーザプログラムに関してその配備方法を以下に示す。

ユーザプログラムは、以下ディレクトリ内に配置する。

```
[Int-Ball2_platform_simulator 展開フォルダ]/Int-Ball2_platform_simulator/src/user/
```

配備するファイル群については、後述するフォーマットに則る必要がある。

ユーザプログラムは、ROS のパッケージとして定義する。パッケージの新規作成方法は以下公式手順を参照のこと: <https://wiki.ros.org/ROS/Tutorials/CreatingPackage>

技術実証プラットフォーム向け ROS パッケージ構成例は以下の通り。以下はパッケージ名が「user001」であり、パッケージ内に含まれるユーザプログラムが「program001.launch」「program002.launch」である場合の構成である。なお、launch 配下に置くファイルについては、後述するユーザプログラム用 roslaunch のテンプレートに則る必要がある。

表 1: 技術実証プラットフォーム向け ROS パッケージ構成例

ディレクトリ	内容
user001/ ※任意に設定可能	<p>特定ユーザのプログラム群を格納するパッケージ。</p> <p>内部に複数のユーザプログラムを含むことができる。</p> <p>ディレクトリ名は後述する ROS パッケージの定義ファイルに記載するパッケージ名と同一の必要がある。</p>
launch/	roslaunch ファイルの配備ディレクトリ。

		※ディレクトリ名は「launch」で固定
	program001.launch	ユーザプログラム(ノード)の起動設定。
	program002.launch	ユーザプログラム(ノード)の起動設定。
	package.xml	ROS パッケージの定義ファイル。定義する「パッケージ名」が、ディレクトリ名「user001」と一致する必要がある。 定義方法は公式手順を参照のこと。 http://wiki.ros.org/catkin/package.xml
	CMakeLists.txt	ROS パッケージのビルド設定ファイル。 定義方法は公式手順を参照のこと。 http://wiki.ros.org/catkin/CMakeLists.txt
	(その他)	任意のソースコードや各種設定ファイルを配備可能。

4.8 ユーザプログラム用 roslaunch

技術実証プラットフォーム用の roslaunch ファイルのテンプレート(定義例)を以下に示す。

<group ns="platform_launch">は必須項目であり、その他項目は任意に設定可能である。

```
<?xml version="1.0"?>
<launch>

  <group ns="platform_launch">
    <!--
    If value set to false, the target nodes will be terminated when user logic is started.
    -->
    <param name="sensor_fusion" value="false" />
    <param name="slam_wrapper" value="false" />
    <param name="ctl_only" value="true" />
    <param name="fsm" value="true" />
    <!--
    If you want to start up camera_left and camera_right,
    you need to stop (set false) slam_wrapper.
    -->
    <param name="camera_left" value="true" />
    <param name="camera_right" value="true" />
  </group>

  <node name="user_template" pkg="user_template" type="user_template.py" output="screen">
    <!-- The parameters to be used in the user's program can be set -->
    <param name="custom_parameter_integer" value="1" />
    <param name="custom_parameter_float" value="2.0" />
  </node>
</launch>
```

```
<param name="custom_parameter_string" value="custom" />
<param name="custom_parameter_boolean" value="true" />
</node>

</launch>
```

<group ns="platform_launch">にて、フライトソフトウェア側の既存機能の利用有無を指定する。ユーザ実装ロジック開始直前に、利用無し(value="false")と定義した機能に対応する ROS ノードが停止する。

<group ns="platform_launch">項目名とフライトソフトウェア機能の対応を以下に示す。

- sensor_fusion : センサフュージョン
- slam_wrapper : Visual SLAM
- ctl_only : 機体推力計算
- fsm : 推力配分

新規作成したプログラムを GSE で呼び出すために、GSE のコンフィグファイルが保存されたフォルダ

[Int-Ball2_platform_gse]\src\ground_system\platform_gui\config

の中にある user_package_list.json に必要事項を追記する(ユーザプログラムを自動で探索するわけではなく、あらかじめリストに書いておく仕様になっている)。

4.9 ビルド手順

4.9.1 Int-Ball2 GSE

Int-Ball2_platform_gse を展開したディレクトリにおいて「catkin_make」を実行する。

```
# cd [Int-Ball2_platform_gse]
$ source /opt/ros/melodic/setup.bash
$ catkin_make
$ sudo mkdir /var/log/ground_system && sudo chown $USER:$USER /var/log/ground_system
```

4.9.2 Int-Ball2 技術実証プラットフォーム

Int-Ball2_platform_simulator を展開したディレクトリにおいて、

```
$ sudo apt install libpcl-dev ros-melodic-pcl-ros
$ catkin_make -DWITH_PCA9685=OFF
```

※パラメータファイルの修正がされていないソースを使用する場合は「catkin_make -DWITH_PCA9685=OFF」を実行する。なお、-DWITH_PCA9685=OFF は、PWM 制御ボードが接続されていない環境で誘導制御機能を動作させるために、推進機能を模擬ノードとしてビルドするためのオプションである。

5. 操作手順

本システムの実行手順を以下に示す。

1. ターミナルより以下コマンドを実行しInt-Ball2 GSEを起動する。

```
$ cd [Int-Ball2 GSE 展開パス]
$ source devel/setup.bash
$ roslaunch platform_gui bringup.launch
```

2. 1と異なるターミナルにおいて以下コマンドを実行しInt-Ball2 技術実証プラットフォームシミュレータを起動する

```
$ cd [Int-Ball2 技術実証プラットフォームシミュレータ展開パス]
$ source devel/setup.bash
$ rosrn platform_sim_tools simulator_bringup.sh
```

3. Gazeboにおいてシミュレーション開始の再生ボタンを押下する。
4. Int-Ball2 GSEの「Platform command」パネルにおける「Operation Type」におけるNavigation ONボタンを押下し、航法機能を起動する。
5. Int-Ball2 GSEの「Platform command」パネルにおける「User programming platform」におけるUser Node、User Launch File、User Containerを実行対象のユーザプログラミング機能に設定し、startボタンを押下する。

例:

User Node	sample_tests
User Launch File	simple_test.launch
User Container	ib2_user:0.1

6. Int-Ball2 GSEの「Platform command」パネルにおける「User programming platform」におけるUser Logicを実行対象のユーザロジックに設定し、startボタンを押下する。
7. 6の実行が完了し、本システムを終了する際は1および2のターミナルにおいてCtrl+Cを押下する。

(注記)

ISSのURDFを読み込む際に、node_1とnode_2のモデルを読み込もうとするとSegmentation Faultが発生する場合がある。このときはiss.urdf内の当該箇所をコメントアウトすること。

6. 問い合わせ先

宇宙航空研究開発機構(JAXA)

space_IVR@jaxa.jp

以上