

User's Documents for ADAMR2 Project

一関高専 専攻科 生産工学専攻 2年 藤野航汰

2021年2月25日

目次

第 1 章	プロジェクト概要	2
1.1	背景	2
1.2	自律移動ロボットプラットフォーム ADAMR2	2
1.3	動作環境	2
第 2 章	セットアップガイド -ハードウェア編-	4
2.1	電装の繋ぎ方	4
2.2	USB デバイスの接続	7
第 3 章	セットアップガイド -ソフトウェア編-	9
3.1	Ubuntu 18.04LTS のインストール	9
3.2	ROS Melodic のインストール	10
3.3	ADAMR2 用 ROS パッケージのインストール	12
第 4 章	ADAMR2 の使用方法	14
4.1	ラジコン操縦の手順	14
4.2	SLAM 実験の手順	15
4.3	自律走行実験の手順	17

第1章

プロジェクト概要

1.1 背景

社会実装指向型研究では、研究成果を現場へ持ち込み、ユーザのレビューから改善点を即座にフィードバックするアジャイル型開発のアプローチが要求されます。しかし、ロボット等のメカニカルデバイスを必要とする研究テーマでは、デバイスの設計開発に多くの時間を必要とすることから、アジャイル型開発が適用しにくい状況がありました。

この課題に対し、一関高専では、東京高専・和歌山高専と共同で自律移動ロボットプラットフォームの開発に取り組んでいます。これは、自律移動に関する部分をプラットフォーム化して提供することで、目的の機能の開発に注力できることを目標としたものです。これまでの研究で、図 1.1 に示すような大径インホイールモータを搭載した移動ロボットが開発されました。

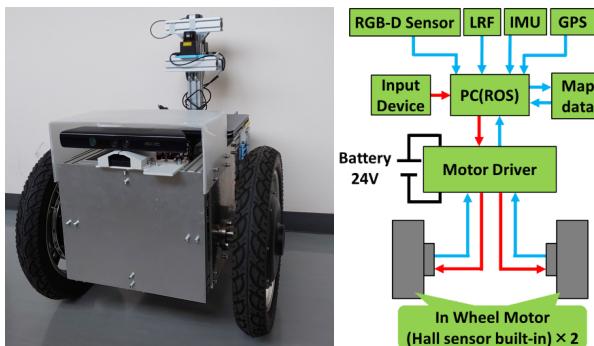


Fig.1.1 Appearance and System Configuration of The Mobile Robot Platform Developed so far

1.2 自律移動ロボットプラットフォーム ADAMR2

現在、図 1.2 に示すような対向 2 輪型移動ロボット「ADAMR2」が開発されています。アジャイル型開発に対応できるという意味（というよりも願い）を込めて、「Agile Developable Autonomous Mobile Robot ver.2」と名付けました。

1.3 動作環境

ADAMR2 は ROS^{*1}を活用して開発されています。ROS パッケージは GitHub 上で公開されており、誰でも利用することができます。

ADAMR2 の ROS パッケージは以下の環境で動作することを確認しています。

- Intel NUC8i7BEH
- CPU: 8th Gen quad-core intel ® Core™ i7 Processor
- GPU: Iris ® Plus Graphics 655
- RAM: Crucial 16GB DDR4-2400 SODIMM CT16G4SFD824A

*1 <http://wiki.ros.org/>

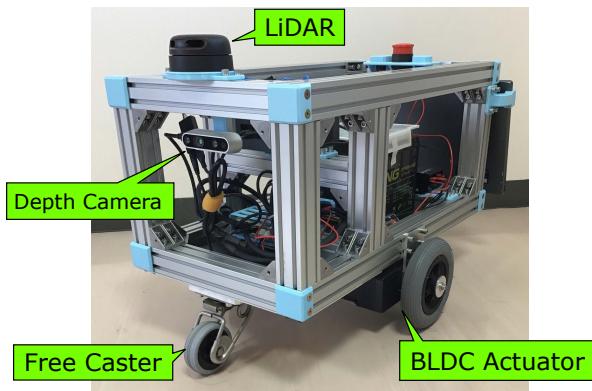


Fig.1.2 Appearance of ADAMR2

- Ubuntu 18.04 LTS
- ROS Melodic Morenia

第2章

セットアップガイド -ハードウェア編-

2.1 電装の繋ぎ方

鉛蓄電池、スイッチ、モータードライバ等のロボットの電装を接続します。電装の接続には主に絶縁被覆付圧着端子を使用します。

2.1.1 部品一覧

電装の配線には以下の部品を使用します。

Name	Model Number	Quantity
鉛蓄電池 12V 20Ah	WP20-12IE	2
端子台	ATK-30-6P	1
メイン電源用スイッチ	273-025	1
DC コンバータ用スイッチ	RB625011FF-10P	1
DC コンバータ 12V-19V	a16052500ux0037	1
DC プラグ	761KS15	1
T-Frog モータードライバ	TF-2MD3-R6	1
T-Frog 電源基板	TF-PW36-5/12M	1
緊急停止スイッチ基板	-	1

Table2.1 List of Electrical Components

Name	Model Number
ニチフ 銅線用絶縁被覆付圧着端子丸型	TMEV1.25-3
差込み型接続端子 187 シリーズ (バリュー品) メス (嵌合部絶縁型)	MTR-480809-FA
絶縁付圧着端子 Y 型	F1.25-5
通信機器用ビニル電線 KV シリーズ	KV 1.25SQ Φ-200
通信機器用ビニル電線 KV シリーズ	KV 1.25SQ Φ-200

Table2.2 List of Wiring Components

2.1.2 必要となる工具

電装の取り付け作業には以下の工具が必要となります。

- 二面幅 8 mm のレンチ
- プラスドライバ (No.2 以上)
- 絶縁被覆付圧着端子用圧着工具 ホーザン P-743

● ワイヤーストリッパ

鉛蓄電池の端子は二面幅 8 mm の六角ナット / ボルトが使用されているので、二面幅 8 mm のレンチが必要になります。電池を取り扱うので、絶縁されている工具が望ましいです。また、レンチにラチェット機能が付いていると作業性が非常に向上します。ラチェット機能付きのソケットレンチを使用することをお勧めします。

ADAMR2 の電装で使用している圧着端子はほぼ全て絶縁被覆なので、圧着工具も絶縁被覆付圧着端子用のものを使用する必要があります。裸圧着端子用の工具を使用すると被覆が破れて不具合が発生する恐れがあるので、使用しないでください。

2.1.3 接続図

電装部品の接続図を図 2.1 に示します。

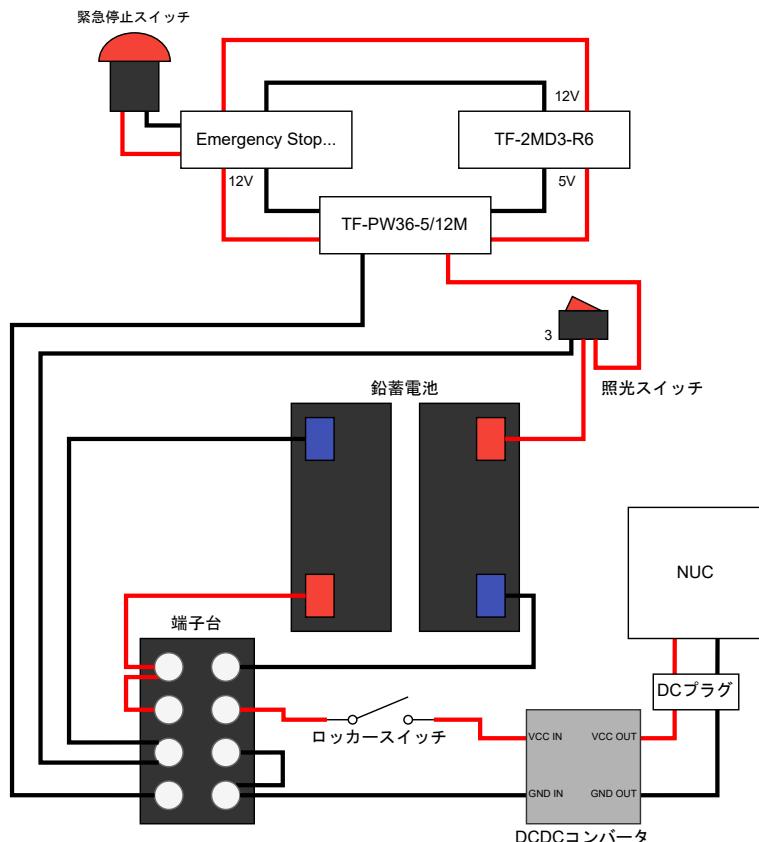


Fig.2.1 Electrical Connection Diagram

2.1.4 端子の接続方法の詳細

鉛蓄電池

鉛蓄電池 WP20-12IE は M5 ねじの接続端子を持っているので、Y 型圧着端子をねじ止めすることで接続することができます。ねじの締結には 8mm 幅のレンチを使用できます。感電やショートに十分注意して作業してください。

端子台

端子台 ATK-30-6P は M5 ねじの接続端子を持っているので、Y 型圧着端子をねじ止めすることで接続できます。

1 つの端子に 2 つの圧着端子を接続することができる、圧着端子を両端につけたケーブルを使って 2 つの端子列をショートさせることができます。上の接続例でも、GND や 12V の箇所でこのような接続方法をとっています。



Fig.2.2 Detail of How to Connect Pb-Battery

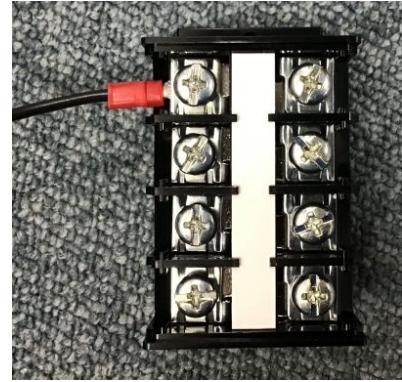


Fig.2.3 Detail of How to Connect Screw Terminal Block

スイッチ

ADAMR2 で使用しているスイッチは、いずれも 187 シリーズの差込型接続端子（ファストン端子）を持っています。接続には 187 シリーズの差込型端子（メス）を付けたケーブルを使用する必要があります。

差込型接続端子ははんだ付け無しで何度も取り外しができますが、付け直す度に端子が摩擦で削れて接続が緩くなる可能性があります。取り外しはできるだけ最小限に抑えるよう注意してください。

T-Frog 電源基板

T-Frog 電源基板は、M3 ねじ止めターミナルを持っていて、接続には内径 ϕ 3.2 の圧着端子を使用します。ADAMR2 では丸形圧着端子を使用しています。

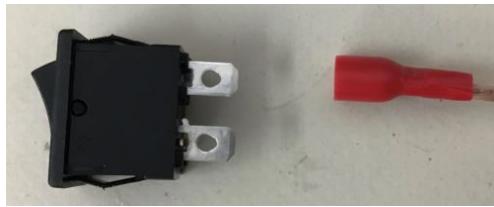


Fig.2.4 Detail of How to Connect Switch

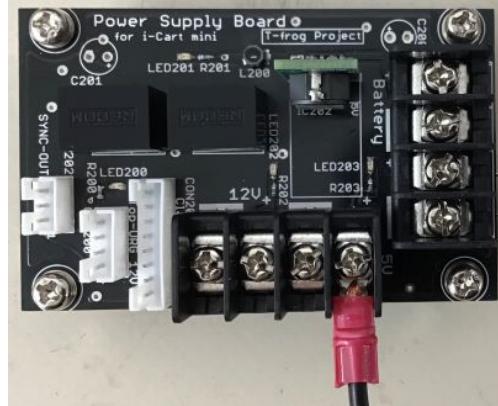


Fig.2.5 Detail of How to Connect T-Frog Power Supply Board

T-Frog モータードライバ

T-Frog モータードライバは、専用のコネクタを介してロジック電源・モーター電源の 2 種類の電源を接続します。購入時に付属してくるコネクタをケーブルに取り付けて接続します。詳細は i-cart mini 組立説明書 (http://t-frog.com/products/icart_mini/assembly/) を参照してください。

緊急停止スイッチ基板

緊急停止スイッチ基板は ADAMR2 プロジェクトオリジナルの基板です。外観を図 2.6 に示します。パワー リレーと緊急停止スイッチを用いて、モーター電源を遮断できるようになっています。T-Frog 電源基板に使用されているものと同一のねじ止めターミナルを持っており、モーター電源はここに接続します。緊急停止スイッチは XH コネクタによって接続します。

この緊急停止スイッチ基板の設計データは https://github.com/fjnkt98/adamr2_emg_board/releases/tag/v0.1.0 にて公開されているため、誰でも同じものを作ることができます。



Fig.2.6 Emergency Stop Switch Circuit Board for ADAMR2

2.2 USB デバイスの接続

ADAMR2 では以下の USB デバイスを使用します：

- T-Frog モータードライバ (TF-2MD3-R6)
- RPLiDAR A2
- Intel Realsense D435
- Logicoool F710 ゲームパッド (オプション)

他に、ディスプレイとして HDMI 接続のモバイルディスプレイを接続し、モバイルディスプレイと RPLiDAR に 5V 電源を供給するためのモバイルバッテリーも搭載します。

各種デバイスの接続図を図 2.7 に示します。

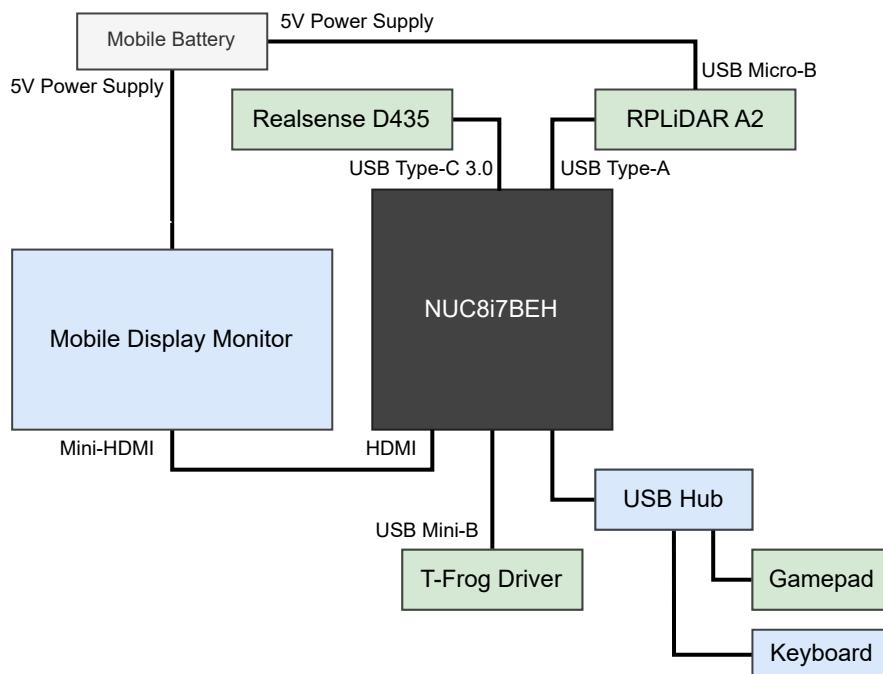


Fig.2.7 USB Device Connection Diagram

注意点として、Intel Realsense D435 と RPLiDAR A2 は、USB ハブを介した接続を行うと電流不足で動

作しなくなる場合があります。PC の USB ポートに直接接続し、電流不足を回避してください。^{*1}

^{*1} RPLiDAR は外部給電を行っているため電流不足にはならないはずなのですが、実際に電流不足に陥ることが多かったため、可能な限り PC の USB ポートに直接接続するようにしてください。

第3章

セットアップガイド -ソフトウェア編-

3.1 Ubuntu 18.04LTS のインストール

この節では Intel NUC へ Ubuntu 18.04LTS をインストールする手順を説明します。尚、NUC の組み立ては完了しているものとします。

3.1.1 必要なもののリスト

Ubuntu のインストール作業には以下のものが必要となります。

- USB メモリ (8GB 以上が望ましい)
- HDMI 入力端子付きディスプレイ
- HDMI ケーブル
- キーボード・マウス (有線)

尚、Wi-Fi 環境がある場合は LAN ケーブル及び有線 LAN 環境は必ずしも必要ではありません。

3.1.2 ライブ USB の作成

NUC へ Ubuntu をインストールするには、Ubuntu OS を記録したメディアを作成する必要があります。今回は USB メモリへ Ubuntu OS の ISO ファイルを書き込んでライブ USB を作成します。

ライブ USB の作成は Windows10 でも行えます。まずは Ubuntu 18.04LTS の ISO イメージファイルと、ライブ USB 作成ソフトの Rufus^{*1}をダウンロードします。

Ubuntu 18.04LTS の ISO ファイルは、以下のミラーサーバからダウンロードすることができます。どのサーバからダウンロードしても構いません。

- 富士大学: <http://cdimage-u-toyama.ubuntulinux.jp/releases/18.04.3/>
- JAIST: <http://ftp.jaist.ac.jp/pub/Linux/ubuntu-jp-cdimage/releases/18.04.3/>
- KDDI 研究所: <http://ftp.kddilabs.jp/Linux/packages/ubuntu-jp/release-cd/releases/18.04.3/>
- 株式会社アプセル: <http://cdimage-appcel.ubuntulinux.jp/releases/18.04.3/>

ダウンロードするファイルは、ubuntu-ja-18.04.3-desktop-amd64.iso です。ファイルサイズが 1.9 GB のものをダウンロードします。

次に Rufus をダウンロードします。公式サイト <https://rufus.ie/> から最新版をダウンロードします。インストールは不要で、.exe ファイルがダウンロードされるのでそれを起動します。

Rufus を起動すると図 3.1.2 のようなダイアログが表示されます。

「デバイス」の欄にはライブ USB にする USB メモリを選択します。「ブートの種類」の欄には先ほどダウンロードした Ubuntu 18.04LTS の ISO イメージファイルを指定します。

「スタート」ボタンをクリックすれば書き込みが開始されます。イメージを書き込んだ USB メモリの中身は全て消去されます。重要なファイルが入っている場合は事前にバックアップを取っておきましょう。

無事にライブ USB が作成できたら、いよいよ Ubuntu のインストール作業に入ります。

^{*1} Fedora Media Writer でもライブ USB の作成を行えます。好きな方を選んでください。

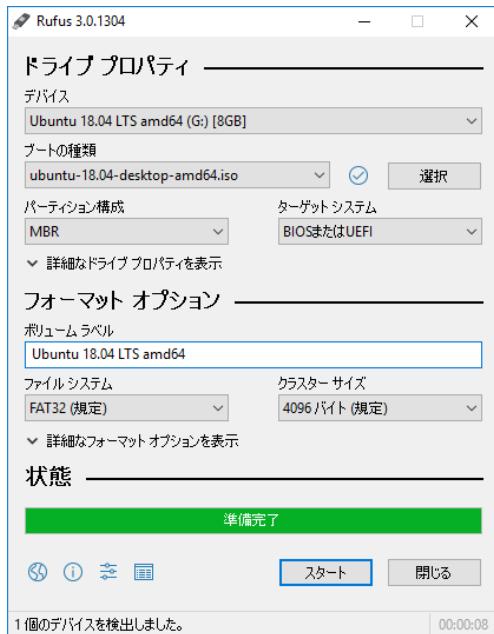


Fig.3.1 Dialog of Rufus

3.1.3 Ubuntu のインストール

作成したライブUSBをNUCに接続し、ディスプレイやキーボード等を繋いでNUCを起動します。「Install Ubuntu」を選択して、Ubuntuのインストールを開始します。Ubuntuのインストールはウィザードの指示に従って作業を行えば大丈夫です。

3.1.4 USB デバイスを使用するための権限設定

RPLiDARやIntel Realsenseなど、USBデバイスをROSノードから使用するためには、ユーザーが dialout グループに属している必要があります。Ubuntuのインストールが完了したら、ユーザーが dialout グループに属しているかどうかを確認してください。コード 3.1 を実行することで確認を行うことができます。

Code 3.1 Check if the User Belongs to the dialout Group

```
groups $USER
```

このコマンドを実行すると、現在のユーザーが所属しているグループの一覧が表示されます。表示された一覧に dialout の文字が無い場合は、ユーザーは dialout グループに所属していません。このままではUSBデバイスにアクセスする権限が無いので、ユーザーを dialout グループに追加します。

コード 3.2 を実行することで、ユーザーを dialout グループに追加することができます。

Code 3.2 Add the User into dialout Group

```
sudo usermod -aG dialout $USER
```

コード 3.2 を実行した後、マシンを再起動してください。

3.2 ROS Melodic のインストール

ROS Melodic Moreniaをインストールします。インストール手順はROS Wiki^{*2}に書かれている内容と同じですが、ビルドシステムに catkin ビルドツールを使用する点が異なります。

^{*2} <http://wiki.ros.org/melodic/Installation/Ubuntu>

3.2.1 ROS 本体のインストール

Ctrl + Alt + T を押してターミナルを開き、以下のコマンドを順番に実行します。

【注意】PDF のコードブロックはコピーペーストした際に文章が崩れる可能性があります。コマンドのコピペは ROS Wiki から行うことをお勧めします。

Code 3.3 リポジトリの鍵の取得

```
sudo sh -c 'echo "deb http://packages.ros.org/ubuntu $(lsb_release -sc) main" \  
> /etc/apt/sources.list.d/ros-latest.list'
```

Code 3.4 鍵の登録

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key \  
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Code 3.5 パッケージリストのアップデート

```
sudo apt update
```

ここではデスクトップ版の ROS をインストールします。

Code 3.6 ROS 本体のインストール

```
sudo apt install ros-melodic-desktop-full
```

Code 3.7 ROS のセットアップスクリプトの読み込みを自動化する

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

Code 3.8 ユーティリティツールのインストール

```
sudo apt install \  
python-rosdep \  
python-rosinstall \  
python-rosinstall-generator \  
python-wstool \  
build-essential
```

Code 3.9 rosdep のインストール

```
sudo rosdep init  
rosdep update
```

3.2.2 catkin ビルドツールのインストール

ROS1 のモダンなビルドツールとして、Catkin Command Line Tools^{*3}が開発されています。

ROS Wiki や ROS の教本などでは catkin_make コマンドを使ってワークスペースの設定やパッケージのビルドを行っていますが、catkin ビルドツールの方が使い勝手が良いです。これから ROS1 を始める際は catkin_make コマンドでなくこちらを使うことをおすすめします。

Catkin Command Line Tools はデフォルトで ROS に含まれていないので、追加でインストールする必要があります。

*3 <https://catkin-tools.readthedocs.io/en/latest/>

Code 3.10 catkin ビルドツールのインストール

```
sudo apt install python-catkin-tools
```

【注意】 catkin コマンドと catkin_make コマンドの間には互換性がありません。そのため、 catkin_make コマンドで初期化したワークスペースを catkin コマンドでビルドすることはできません。もし既に catkin_make コマンドでワークスペースを初期化している場合には、 catkin_ws ディレクトリにある src/以外のディレクトリを削除してから catkin コマンドでビルドし直してください。

3.2.3 ワークスペースの初期化

catkin コマンドを使って、ROS のワークスペースを初期化します。

Code 3.11 ワークスペースの初期化

```
mkdir -p ~/catkin_ws/src  
cd ~/catkin_ws  
catkin init
```

これで ROS のインストール作業は終了です。次に、ADAMR2 用の ROS パッケージをインストールする作業を行います。

3.3 ADAMR2 用 ROS パッケージのインストール

ADAMR2 用の ROS パッケージをインストールします。パッケージをダウンロードする際に Git^{*4}が必要になります。git がインストールされていない場合は、追加でインストールしましょう。

Code 3.12 Git のインストール

```
sudo apt install git
```

3.3.1 パッケージのクローン

GitHub^{*5} から ADAMR2 用 ROS パッケージをワークスペースにクローンします。melodic-devel ブランチをクローンしましょう。

Code 3.13 パッケージのクローニング

```
cd ~/catkin_ws/src  
git clone -b melodic-devel https://github.com/fjnkt98/adamr2_ros1_packages.git
```

この時点ではまだビルド時依存パッケージである YP-Spur^{*6} がインストールされていないので、パッケージのビルドはできません。YP-Spur のインストールは次の小節で行います。

3.3.2 依存パッケージのインストール

ADAMR2 用 ROS パッケージが依存しているパッケージを rosdep を用いてインストールします。コード 3.14 のコマンドを実行します。

Code 3.14 依存パッケージのインストール

```
rosdep install -i --from-path ~/catkin_ws/src/adamr2_ros1_packages/
```

このコマンドを実行することで、rosdep が依存関係を解決し、不足しているパッケージを apt-get コマンドを使用して自動的にインストールしてくれます。

*4 <https://git-scm.com/>

*5 https://github.com/fjnkt98/adamr2_ros1_packages/tree/melodic-devel

*6 <https://openspur.org/>

この作業を行うことで、YP-Spur のバイナリも一緒にインストールされます。また、YP-Spur の ROS ラッパーとして `ypspur_ros`^{*7} パッケージも提供されていますが、ADAMR2 では使用しません。

3.3.3 パッケージのビルド

依存関係を解決したので、パッケージをビルドできるようになりました。コード 3.15 のコマンドを実行してパッケージをビルドします。

Code 3.15 パッケージのビルド

```
cd ~/catkin_ws  
catkin build
```

ビルドが完了したら、ワークスペース内の `setup.bash` スクリプトを読み込みます。

Code 3.16 セットアップスクリプトの読み込み

```
source ~/catkin_ws/devel/setup.bash
```

ついでに `~/.bashrc` に書き込んで自動化してしまいましょう。

Code 3.17 セットアップスクリプト読み込みの自動化

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

3.3.4 環境設定

実際にロボットの実機を動かす際は、いくつかの環境設定を行う必要があります。

まずは実行ファイルへの権限付与と実験用ディレクトリの生成を行います。`adamr2_ros1_packages/utility` ディレクトリ内にある `setup_environment.sh` を実行します。

Code 3.18 環境設定スクリプトの実行

```
cd ~/catkin_ws/src/adamr2_ros1_packages/utility  
bash setup_environment.sh
```

このシェルスクリプトは以下の処理を実行しています。

- `ypspur-coordinator` を起動するスクリプト `ypspur_launcher.sh` に実行権限を付与する
- 実験で取得した `rosbag` ファイルや地図ファイルを保存しておくためのディレクトリをホームディレクトリに生成する

手動で行うのが面倒な作業を一括で行ってくれます。

次に、USB デバイス名の固定を行います。T-Frog モータードライバ及び RPLiDAR を PC に接続した状態で、`adamr2_ros1_packages/utility/udev` ディレクトリ内にある `set_udev_rules.sh` を実行します。

Code 3.19 udev rules の設定

```
cd ~/catkin_ws/src/adamr2_ros1_packages/utility  
bash set_udev_rules.sh
```

内部で `sudo` コマンドを実行しているため、このシェルスクリプトを実行する際はパスワードが要求されます。

このシェルスクリプトを実行することによって、T-Frog モータードライバと RPLiDAR のデバイスパスのシンボリックリンク `/dev/tfrog-driver` と `/dev/rplidar` が生成されます。

ここまで手順により、ROS の開発環境を整えることができました。ハードウェアのセットアップが完了しているなら、各パッケージにある `launch` ファイルを実行することでロボットを動かすことができます。

*7 https://github.com/openspur/ypspur_ros

第4章

ADAMR2 の使用方法

本章では ADAMR2 の起動方法について解説します。ADAMR2 はゲームパッドによる手動操縦、SLAM による地図作成、および自律走行を実行するためのアプリケーションを備えています。各種機能は、パッケージに含まれる launch ファイルを起動することで実行することができます。

4.1 ラジコン操縦の手順

ゲームパッドを用いた手動遠隔操縦を行うためには、まず ADAMR2 のモータードライバを起動状態にします。以下の手順に従って、ロボットのハードウェアをセットアップしてください。尚、モータードライバとモータとの接続は既に済んでいるものとします。

1. 図 2.1 を参考にして電装部品の接続を行う。
2. モータードライバと PC を USB ケーブルで接続する。
3. 電源スイッチを ON にし、電源基板およびモータードライバに電源を供給する。
4. PC からモータードライバのデバイスファイルが見えるかどうかを確認する。

モータードライバのデバイスファイルが存在するかどうかを確認するには、コード 4.1 を実行します。

Code 4.1 Check the Device File of the Motor Driver

```
$ ls /dev/ttyACM*
/dev/ttyACM0
```

デフォルトでは、モータードライバ TF-2MD3-R6 は /dev/ttyACM0 として認識されます。デバイスファイルが見つからなかった場合は、USB ケーブルを繋ぎ直すか、電源を再投入する等してください。

モータードライバが PC から認識可能になったら、launch ファイルを起動して走行可能な状態にします。ADAMR2 を走行可能な状態にするには、新しいターミナルを開き、コード 4.2 を実行します。

Code 4.2 Launch adamr2_control.launch

```
roslaunch adamr2_control adamr2_control.launch
```

次にゲームパッドを使用可能にします。ゲームパッドを PC に接続し、新しいターミナルを開いてコード 4.3 を実行します。

Code 4.3 Launch joy.launch

```
roslaunch adamr2_bringup joy.launch
```

これで準備は整いました。緊急停止スイッチを解除し、ゲームパッドの A ボタン（もしくは B ボタン）を押しながら左アナログジョイスティックを動かすことでの、ロボットをラジコン操縦することができます。

尚、ロボットを動作させるときは電気的トラブルに十分注意してください。初めてロボットを起動する場合は、鉛蓄電池を直接繋ぐのではなく、電流保護機能付きの直流安定化電源を使用して、必ず動作テストを行ってから実験を行ってください。^{*1}

ロボットの操作を終了するには、起動した launch ファイルを終了すれば OK です。各 launch ファイルを実

^{*1} 筆者は一度やらかして電源基板とドライバ基板を焼いています。

行しているターミナルで Ctrl+C を入力することでプロセスを終了することができます。launch ファイルを終了させたら、ロボットの電源スイッチを OFF にして電源を切ります。

4.2 SLAM 実験の手順

ADAMR2 を使って slam_gmapping を用いた SLAM による地図作成を行うことができます。SLAM で地図作成実験を行うときは、以下の 2 つのシチュエーションが考えられます。

- オンライン SLAM：ロボットを走行させながらリアルタイムに地図作成を行う
- オフライン SLAM：ロボットの走行データを保存して、それを再生しながら地図作成を行う

本節では 2 つのシチュエーションに分けて、SLAM の実行手順を説明します。

4.2.1 オンライン SLAM

SLAM を行うには、ロボットのオドメトリ情報とレーザースキャン情報が必要になります。オドメトリ情報はモータードライバが提供してくれるので、あとはレーザースキャン情報を追加すれば良いということになります。

まずは 4.1 で説明したように、ロボットを走行可能な状態にします。そして、RPLiDAR を PC に接続し、新しいターミナルを開いてコード 4.4 を実行します。

Code 4.4 Launch rplidar.launch

```
roslaunch adamr2_bringup rplidar.launch
```

RPLiDAR が起動して、レーザースキャンのデータを取得できるようになれば準備完了です。あとは slam_gmapping を起動する launch ファイルを実行します。新しいターミナルを開き、コード 4.5 を実行します。

Code 4.5 Launch gmapping.launch

```
roslaunch adamr2_slam gmapping.launch situation:=real
```

注意点として、実験時のシチュエーションを引数に指定しなければなりません。これは、Gazebo シミュレーションでの SLAM 実験と、実機による実験とで slam_gmapping のパラメータを切り替えるための設定です。situation 引数を設定しないと、gmapping.launch を実行することはできません。

gmapping.launch を実行すると、slam_gmapping ノードと rviz が起動します。rviz の画面を確認して、トピック等に抜けがないかをチェックしてください。特に RPLiDAR は電流不足によるデータの配信ミスが多いです。

トピックに問題が無ければ、ゲームパッドによってロボットを操縦します。ロボットの移動に従って、地図がどんどん作成されていくのが確認できます

地図の作成が終わったら、作成した地図を保存しておきます。地図の保存は map_server パッケージの map_saver ノードによって行えます。^{*2}

地図を保存したいディレクトリに移動し、コード 4.6 を実行します。

Code 4.6 Save Map

```
rosrun map_server map_saver
```

コード 4.6 を実行すると、カレントディレクトリに map.pgm と map.yaml の 2 つのファイルが生成されます。

地図の保存が完了したら、ロボットを終了させます。

^{*2} map_server パッケージには map_server ノードと map_saver ノードが存在します。名前が似ていて非常に紛らわしいので、間違えないように注意してください。

4.2.2 オフライン SLAM

オフラインで SLAM を実行する場合は、まずロボットの走行データを Bag ファイルに保存する必要があります。前小節と同様にロボットと LiDAR を起動し、走行データを集めます。

ヘルパースクリプトを使用した走行データの記録

走行データの記録は rosbag コマンドを使用するのですが、ADAMR2 では SLAM 用のデータを一括で記録するためのヘルパースクリプトを作ったので、それを使います。ヘルパースクリプトは adamr2_ros1_packages/utility/bag_tools ディレクトリに置かれています。SLAM 用のデータを集めるスクリプトは slam_real.sh です。slam_real.sh の中身はコード 4.7 のようになっています。

Code 4.7 slam_real.sh

```
#!/bin/bash
set -e

TODAY=$(date "+%Y%m%d")

TARGET_DIR="/home/${USER}/adamr2_experiments/real/slam/${TODAY}"

if [ ! -e $TARGET_DIR ]; then
    mkdir -p $TARGET_DIR
    echo "-- Target directory was successfully generated. --"
fi

cd $TARGET_DIR

DIR_NUM=$(ls -l | grep ^d | wc -l)
TITLE_NUM=$((++DIR_NUM))

mkdir exp$((TITLE_NUM))
cd exp$((TITLE_NUM))

mkdir bag && cd bag

rosbag record /tf \
    /tf_static \
    /adaml2/camera/depth/color/points \
    /adaml2/camera/color/image_raw \
    /adaml2/scan \
    /adaml2/diff_drive_controller/odom
```

slam_real.sh はホームディレクトリに adamr2_experiments というディレクトリを作り、実行時の日付を名前としたディレクトリを作ります。また、slam_real.sh が実行される度にディレクトリを作るようになっているので、ディレクトリ構造がすっきりするようになっています。

slam_real.sh を実行するには、スクリプトが置かれているディレクトリに移動し、コード 4.8 を実行します。

Code 4.8 Run slam_real.sh

```
bash ./slam_real.sh
```

slam_real.sh を実行すると rosbag によるデータの記録が始まります。そのままロボットを走行させ、走行データを収集します。

記録をストップするには、slam_real.sh を実行したターミナルで Ctrl+C を押せば終了します。

bag ファイルによる SLAM

bag ファイルが記録できたら、それを使って SLAM を行います。オフライン SLAM ではロボットを使う必要は無いため、ここからの作業ではロボットの電源を切っておいて構いません。

まず新しいターミナルを開き、ROS Master を起動します。

Code 4.9 Execute roscore

```
roscore
```

更に新しいターミナルを開き、`use_sim_time` パラメータを `true` に設定します。これは、`rosbag` のファイルを再生して処理を行う際に必要となるオプションです。

Code 4.10 Set `use_sim_time` to true

```
rosparam use_sim_time true
```

また、ロボット内部の TF 情報を ROS 側が見えるようにするため、`robot_description` パラメータにロボットの URDF を登録する必要があります。`adamlr2_description` パッケージにそれ用の launch ファイルを準備しているので、それを実行します。

Code 4.11 Launch `adamlr2_description.launch`

```
roslaunch adamlr2_description adamlr2_description.launch
```

何故 URDF の登録が必要かというと、ロボットのベースリンクから LiDAR リンクまでの座標変換が必要になると、ロボットのモデルを `rviz` で表示するためです。

更にもう 1 つのターミナルを開き、`slam_gmapping` を実行します。先ほど作成した launch ファイルを実行します。`slam_gmapping` はデータが送られない限り処理を行わないので、Bag ファイルを再生する前に起動しておきます。`situation` 引数の設定を忘れないようにしましょう。

Code 4.12 Run `gmapping.launch`

```
roslaunch adamlr2_slam gmapping.launch situation:=real
```

最後に Bag ファイルを再生します。ターミナルを新しく開き、Bag ファイルがあるディレクトリに移動し、コード 4.13 を実行します。

Code 4.13 Replay `slam.bag`

```
rosbag play --clock XXXXXXXXXXXXXXXXX.bag
```

Bag ファイル名は適切な名前を入力してください。Tab キーを押せば保管されるはずなので、入力は簡単です。

Bag ファイルの再生に伴い、SLAM が実行され、地図が生成されます。Bag ファイルの再生が終了したら地図を保存しましょう。地図の保存場所は、Bag ファイルがあるディレクトリと紐付けておくのが良いでしょう。

4.3 自律走行実験の手順

自律走行実験を行うためには、SLAM で作成した地図が事前に必要となります。自律走行を行いたい場所の地図をあらかじめ作成しておき、適当なディレクトリに保存しておいてください。

Navigation Stack を使用した自律走行実験を行うには、`adamlr2_navigation` パッケージに置かれている launch ファイルを使用します。自律走行を行うためには、まずロボットを起動して走行可能な状態にします。

SLAM のときと異なり、自律走行を行う際は低い位置の障害物を検知するために Realsense D435 による点群情報が必要になります。そのため、`adamlr2_control.launch`, `rplidar.launch` の他に、`adamlr2_bringup` パッケージにある `realsense.launch` を起動する必要があります。これらの launch ファイルを個別に実行してもよいのですが、3 つのターミナルを開いてそれぞれを実行するのは少々手間です。`adamlr2_bringup` パッケージには、これら 3 つの launch ファイルを一括で実行する launch ファイルである `adamlr2_bringup.launch` が用意されています。これを実行することで自律走行に必要な準備を行うことができます。

Code 4.14 Launch `adamlr2_bringup.launch`

```
roslaunch adamlr2_bringup adamlr2_bringup.launch
```

次に自律走行のためのノード群を起動するための launch ファイルを実行します。`adamlr2_navigation` パッケージに置かれている `adamlr2_navigation.launch` を実行することで、Navigation Stack のノードを一括で実

行することができます。

`adamr2_navigation.launch` は、実行時に 3 つの引数を取ります。`situation`, `map_file`, そして `environment` です。`situation` は実験時のシチュエーションを指定する引数で、SLAM のときと同様に `real` もしくは `sim` を指定します。`map_file` は実験で使用する地図のパスを指定する引数です。パスの指定にはフルパスを入力する必要があります。いちいちフルパスを入れるのはさすがに面倒なので、対象の地図があるディレクトリに移動し、UNIX の `pwd` コマンドを活用すると良いでしょう。`environment` はロボットが動作する環境(屋内か屋外か)を指定します。屋内環境と屋外環境では、環境のスケールやセンサに混入するノイズ等のパラメータが変わってしまうので、`adamr2_navigation` パッケージではそれぞれの環境に合わせたパラメータファイルを用意しています(`config/`ディレクトリ以下)。`environment` には `indoor` もしくは `outdoor` のどちらかを指定します。具体的には、[4.15](#) のように実行します。

Code 4.15 Launch `adamr2_navigation.launch`

```
cd path/to/the/map/file
roslaunch adamr2_navigation adamr2_navigation.launch \
situation:=real map_file:='pwd'/'map.yaml environment:=indoor
```

`path/to/the/map/file` には実際の地図があるディレクトリを指定してください。Linux コマンドラインでは、バッククオートでコマンドを囲むことでコマンドの実行結果をその場所に展開することができます。`pwd` コマンドはカレントディレクトリのパスを出力するコマンドなので^{*3}、結果として地図へのフルパスを `map_file` 引数に指定できることになります。これら 3 つのパラメータにはデフォルト値を設定していません。つまり、`adamr2_navigation.launch` はこの 3 つの引数を指定してやらないと起動しないので注意してください。

`adamr2_navigation.launch` を起動すると、Navigation Stack の各種ノードと `rviz` が起動します。あとは `rviz` 上でナビゲーションの目的地を指定してやれば、自律走行が開始されます。

ちなみに、`adamr2_navigation.launch` で起動するグローバルプランナーは `global_planner`、ローカルプランナーは `base_local_planner` です。起動するプランナーを変更したい場合は、`adamr2_navigation.launch` の中の `arg` タグのうち、`global_planner_param_file` および `local_planner_param_file` の項目を編集して、読み込むパラメータファイルを変更してください。

^{*3} Print Working Directory の略です。