ROSCON Japan Workshopマニュアル

「AIジェスチャー認識・人体ボディ認識によるロボット駆動」資料　ROSCon JP 2025 Workshop A

会場のWi-Fi情報

- SSID: RCJP25WS-A
  (2Gと5Gがあります、どちらもOK)
- PASS: togetheROS

全体では
1. 概論：ROSとROS 2、TROS、そしてRDKの基礎知識
2. 教材：Originbot Pro(PD)/RDK X5
3. RDK StudioによるRDK X5(シングルボードコンピュータ）のセットアップ
4. ボディ認識・ジェスチャー認識によるロボット制御体験
5. ジェスチャー認識ノードのカスタマイズとビルド体験
6. ジェスチャー制御プログラムの構成とROS 2的解釈

の6章になります。

ワークショップ情報 内容
https://roscon.jp/#workshop_A

Time	Title
12:30	受付開始
13:00	ROS 2概論・TogetherROS環境、OriginBot教材概要紹介
13:40	OriginBotの設定
14:00	ジェスチャー認識・人体認識の実行

16:00	RDK StudioとVNC-Linux Desktopによるロボット開発方法
16:30	ロボット駆動のプログラミング
17:30	発展的な話題
18:00	終

スイッチサイエンス/金沢大学　高須正和. 著作権はCC BY 4.0

1:概論：ROS/ROS2について

概論：ROSとROS 2、TROS、そしてRDKの基礎知識

この章では、ROSファミリー（ROS, ROS 2)について、初学者にもわかりやすく解説します。実習内容に直結する用語や構成の理解をここで整理しておきましょう。

1. ROS/ROS 2とは

ROS 2公式ドキュメント
https://docs.ros.org/en/humble/index.html

ROSとは？

ROS（Robot Operating System）は、ロボットシステムを効率的に構築するためのオープンソースミドルウェアです。ノード、トピック、サービス、アクションなどの仕組みを通じて、複雑なロボットの機能をモジュール化し、再利用性と保守性を向上させることができます。

ROS 2の特徴

ROS 2は、ROS1の後継としてリアルタイム性、セキュリティ、マルチプラットフォーム対応などの機能が強化された新世代のROSです。産業応用や学術研究、教育用途でも広く使われています。

ROS 2の主な特徴：

- DDS（Data Distribution Service）による高信頼通信
- リアルタイム制御対応（PREEMPT_RTなど）
- セキュリティ（通信の暗号化・認証）
- マルチプラットフォーム対応（Linux/Windows/micro-ROS）
- コンポーネントベース設計（ノードを動的にロード可能）
- Pythonベースの柔軟なLaunchシステム

なぜ**ROS**を使うのか？

項目	**ROS**を使う場合	**ROS**を使わない場合
開発効率	パッケージ再利用、モジュール化により迅速	全機能を自作、開発時間が長い
柔軟性	異種ハード統合が容易	ハードウェアごとに固有実装
コスト	オープンソースで低コスト	ライセンス費や独自開発コストが高い

サポート	豊富なチュートリアルと活発なコミュニティ	自社開発・自力解決が前提
リアルタイム性	ROS 2で強化（RT対応）	専用RTOSや実装が必要

ROS 2の構造と用語 (今回のワークショップに出てくるもの）

ノード（Node）

ROS 2の処理単位。各ノードは独立した機能を持ち、相互に通信します。

トピック（Topic）

非同期・一方向通信でデータを送る仕組み（例：画像、速度など）。

メッセージ（Message）

トピックでやりとりされるデータ構造。例えば `/cmd_vel` トピックでは `geometry_msgs::msg::Twist` 型が使われます。

サービス（Service）

同期通信で、リクエスト・レスポンス型のやり取り（例：座標取得、ファイル保存）。

アクション（Action）

長時間かかる処理を進行状況付きでやりとりする仕組み（例：ナビゲーション）。

ROS 2 Humble:基本的な用語と解説

※今回のワークショップではあまり意識しなくても開発できます

1. **DDS (Data Distribution Service)** の採用とメッセージング**:**
    - ROS 2は、ノード間の通信メカニズムとしてDDSを採用しています。DDSは、リアルタイム性、信頼性、拡張性に優れたデータ配布ミドルウェアであり、産業用アプリケーションでの利用実績も豊富です。
    - これにより、ROS1のTCP/IPベースの通信よりも、より効率的で信頼性の高い通信が可能になりました。
    - メッセージング**:** ROS 2におけるメッセージングはDDSを通じて行われます。ノード間でデータをやり取りする主要な方法は以下の通りです。
        - トピック **(Topics):** 一方向の非同期通信に使用されます。あるノードがメッセージをパブリッシュ（発行）し、そのトピックをサブスクライブ（購読）している複数のノードがメッセージを受信します。センサーデータ（例：Lidarスキャン、カメラ画

像）やロボットの状態（例：現在の位置、速度）などの連続的なデータストリームに適しています。
        - サービス **(Services):** 双方向の同期通信に使用されます。クライアントノードがリクエストを送信し、サービスを提供するサーバーノードがそのリクエストを処理し、レスポンスを返します。特定のタスクの実行（例：ロボットのアームを特定の姿勢に移動する、地図を保存する）など、応答が必要な処理に適しています。
        - アクション **(Actions):** サービスとトピックを組み合わせたような、長時間実行されるタスク（例：目標地点までのナビゲーション、複雑な操作）に使用されます。クライアントはゴールを送信し、サーバーはゴールの状態（進行中、完了、中止など）をフィードバックとして送信し、最終的な結果を返します。
    - 複数のDDS実装（例: eProsima Fast RTPS, RTI Connext DDSなど）を選択でき、用途に応じて最適なものを選ぶことができます。
2. リアルタイム制御の強化:
    - DDSの採用と、OSレベルでのリアルタイムパッチ（例: PREEMPT_RT）のサポートにより、リアルタイム性を要求されるアプリケーションにも対応できるようになりました。
    - これにより、産業用ロボットや自動運転システムなど、厳しい時間制約を持つ分野でのROS 2の活用が進んでいます。
3. セキュリティ機能の強化:
    - ROS 2は、通信の認証、暗号化、アクセス制御などのセキュリティ機能を標準でサポートしています。
    - これにより、ロボットシステムへの不正アクセスやデータの改ざんを防ぎ、より安全なシステムを構築できます。
4. マルチプラットフォーム対応:
    - ROS 2は、Linuxだけでなく、WindowsやmacOS、RTOSなど、より多くのプラットフォームをサポートしています。
    - これにより、開発環境の選択肢が広がり、様々なデバイスでのロボット開発が可能になります。
5. コンポーネントベースのアーキテクチャ:
    - ROS 2では、ノードをさらに小さな「コンポーネント」に分割し、プラグインのようにロード・アンロードできるようになりました。
    - これにより、システムのリソース効率が向上し、より柔軟なアプリケーション開発が可能になります。
6. ローンチシステムの改善:
    - ROS 2のローンチシステムは、Pythonベースで記述され、より強力な機能と柔軟性を提供します。
    - 複数のノードやコンポーネントの起動、パラメータの設定、環境変数の管理などを、より簡潔に記述できるようになりました。
7. 豊富なツールとライブラリ:
    - ROS1と同様に、ROS 2にも可視化ツール (Rviz2)、デバッグツール (rqt_*)、シミュレーション環境 (Gazebo) など、開発を支援する多様なツールとライブラリが提供されています。
    - これにより、開発者はロボットの動作を効率的にテストし、デバッグすることができます。

ROS 2は、これらの特徴により、研究開発から実用的な産業用アプリケーションまで、幅広い分野でのロボット開発を強力にサポートするシステムとなっています。特にDDSによる柔軟で堅牢なメッセージング機構が、複雑なロボットシステムの構築を可能にしています。

2:教材：RDK X5 / Originbot Pro(PD)

教材：Originbot Pro(PD)/RDK X5

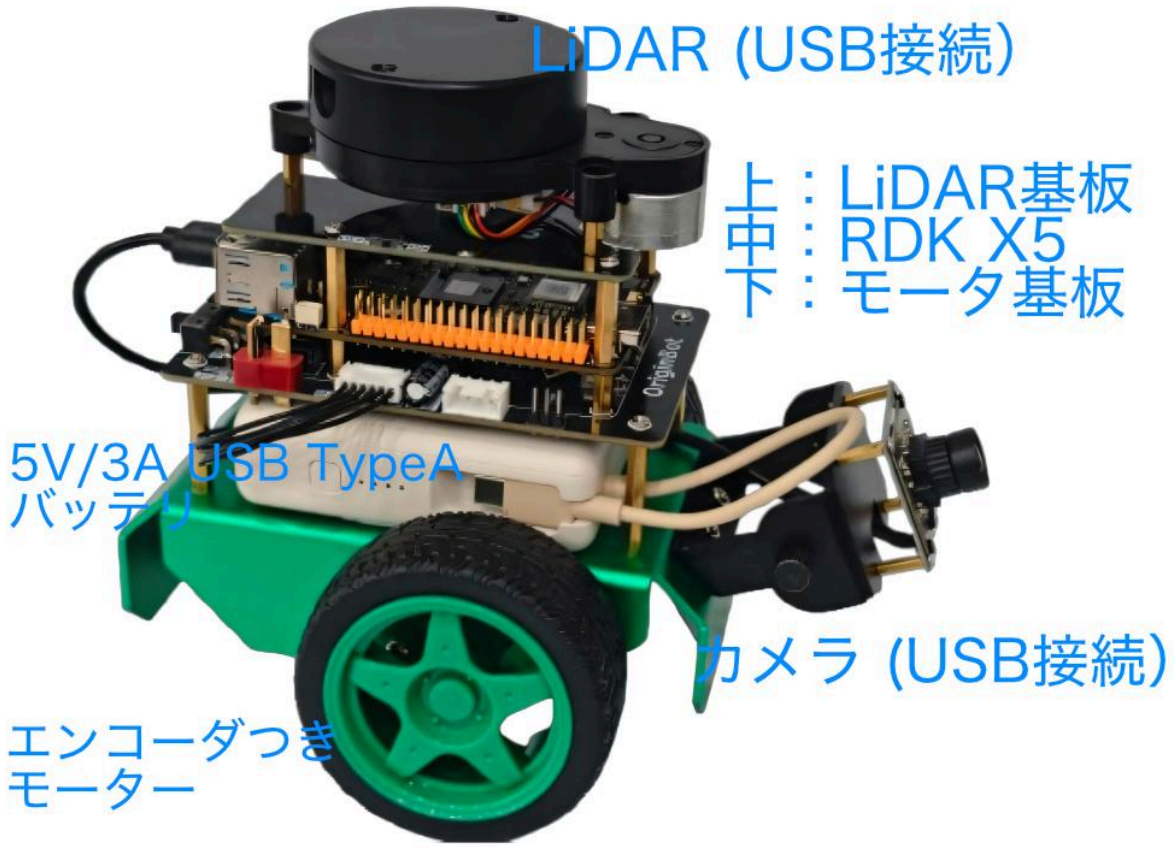
この章では、本ワークショップで使用するロボット教材「Originbot Pro (PD)」と、その中核をなすシングルボードコンピュータ「RDK X5」、そしてROS 2環境として構築されている「TogetheROS」について紹介します。

Originbot Pro PD紹介

Originbot Pro (PD)は、教育・研究用途に最適化されたROS 2対応の小型モバイルロボットです。

主な特徴

- RDK X5を搭載し、**ROS 2 Humble**環境があらかじめ構築済み
- 5V/3Aの市販Powerbankで安全に給電可能（Type-A出力）
- カメラ、LiDAR、エンコーダ付きモーターなどを搭載
- 拡張ピンやUSBにより、3Dカメラやマイクなどを追加可能



↑Originbot Pro PD（見やすいように一部のケーブルは外しています）

製品情報

https://www.switch-science.com/products/10505

RDK X5Originbot Proスペックシート

アプリケーション プロセッサ	**RDK X5** CPU：オクタコア ARM Cortex-A55@1.5 GHz BPU：32 Gflops（10 TOPS相当） メモリ：8 GB LPDDR4 RAM ストレージ：32 GB（TFカード）
モーション コントローラ	MCU： STM32F103 Flash：64 kB RAM：20 kB
差動シャーシ	エンコーダ付きTTモーター（2x2） カメラブラケット ユニバーサル車輪とブラケット
カメラ	720P USBカメラ
LiDAR（TOF）	6 Hzスキャン周波数 360°スキャン角度 3000 Hz測距周波数
ジェスチャーセンサ	加速度/角速度/角出力 カルマンフィルタ内蔵
寸法	180 mm x 132 mm x 154 mm（±2）
重量	0.78 kg（±0.05）
最大速度	0.8 m/s（±0.05）
OS	Ubuntu 22.04（Server） ROS 2 Humble TogetheROS 3.0.0
電源	5 V/3 A USB Type-A モバイルバッテリー（別売）

RDK X5の紹介

（Robot Development Kit X5）は、ROS 2とAI推論に対応した教育・開発向けの高性能シングルボードコンピュータです。

主な仕様

- **ROS 2 Humble**対応**OS** をプリインストール可能
- Raspberry Piと互換性のあるピン配置
- AI推論用 **BPU**（**Brain Processing Unit**）を搭載
- CAN FD、RTC、MIPIカメラなどのロボット向け拡張ポート
- Originbot Proの“2階部分”として搭載

このBPUにより、物体検出やジェスチャー認識といったAI処理を高速かつ省電力で実行可能です。

製品情報
https://www.switch-science.com/products/10500

RDK X5シングルボードコンピュータ。Originbot Pro (PD版）の2階部分の基板です。

3. TogetheROSとは？

TogetheROSは、D-Roboticsが開発した**ROS 2 Humble**ベースの統合開発環境です。ROS 2を学びやすくするための設定・ツール・ドライバが最初から組み込まれています。

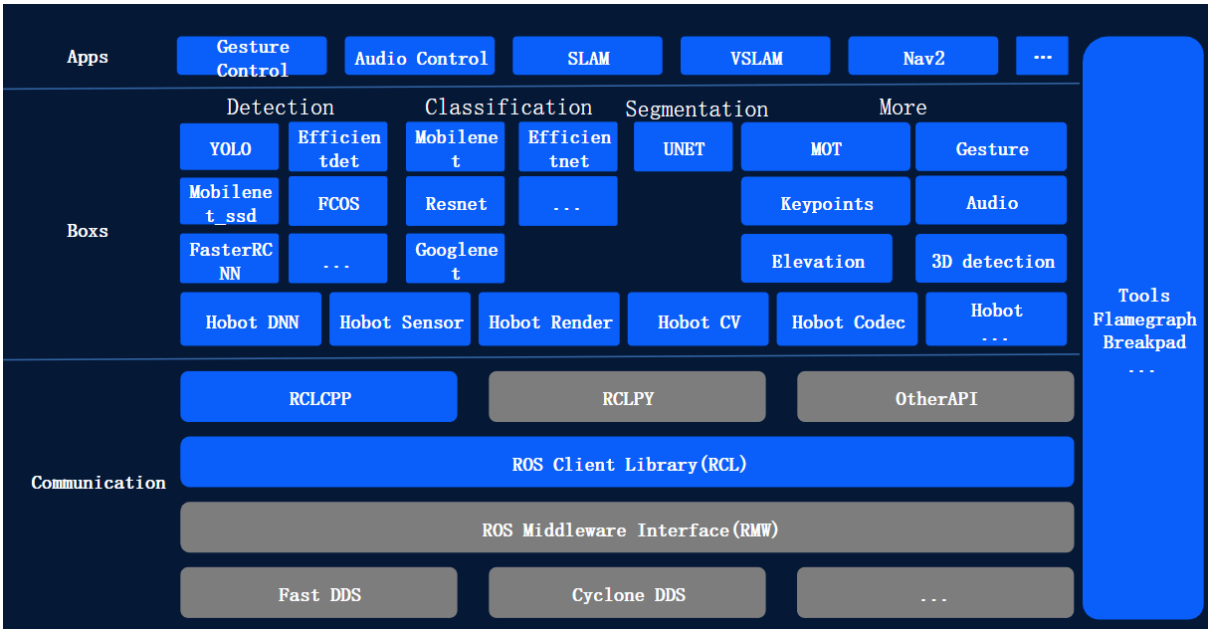
項目	**ROS 2 Humble**	**TogetheROS**
開発元	OSRF（Open Source Robotics Foundation）	D-Robotics（深圳）

対象	全般向け	教育・初学者向けに特化
構成	ROS 2の標準ディストリビューション	ROS 2 Humbleをベースにした独自拡張版（RDK OS）
セットアップ	Ubuntu上にユーザーが構築	すでに構成済み（SDカード/VM）
拡張	ユーザーが自力で導入	GUIやシミュレータ、教材付き
対応ハード	TurtleBot等汎用ロボット	RDK X5搭載機に最適化

TogetheROSでは、カメラ・LiDARのドライバや物体認識モジュールなどが初めから含まれており、セットアップの手間が大幅に削減されています。

TogetheROSを使う利点

TogetheROSは環境の開発も、ハードウェアの開発もD-Robotics社（と親会社のホライゾン・ロボティクス）が行っています。RDK X5に搭載されているX5チップにはBPUというAI強化機能が入っていて、高速・省電力で演算を行うことが行えます。



TogetheROSの概要図

実装構成図（抜粋）

- 青色部分：BPUで動作する高速AI処理領域
- その他：ROS 2ノード群やLaunch設定など

上の図の青い部分はBPUを利用でき、高速・省電力で動作する部分です。ソフトウェアはオープンソースですが、ハードウェア的にはX5に最適化されています。

4. BPUとONNXツールチェーン

RDK X5に搭載されている**BPU**（**Brain Processing Unit**）は、画像認識・音声処理・物体追跡などを高速・省電力で処理可能な**AI**チップです。特に**YOLO**やジェスチャー分類のような**CNN**ベースの推論処理でその力を発揮します。

自分で作成したAIモデル（例：YOLOv5）をこのBPU上で動かすには、D-Robotics提供の **ONNX** ツールチェーン を用いて、モデルをInt8/FP32形式に最適化する必要があります。

- ONNXツールチェーンのドキュメント：
  https://developer.d-robotics.cc/rdk_doc/Advanced_development/toolchain_development/overview
- ONNXを使ってBPU最適化するときに使える演算子/Operator一覧（主だったものはだいたい使える）
  https://developer.d-robotics.cc/rdk_doc/Advanced_development/toolchain_development/intermediate/supported_op_list

まとめ

本ワークショップでは、以下の環境を前提に学習を進めます：

- Originbot Pro (PD版)：ロボット本体
- RDK X5：ROS 2とBPU推論を司る計算プラットフォーム
- TogetheROS：教育用途に最適化されたROS 2 Humble拡張環境

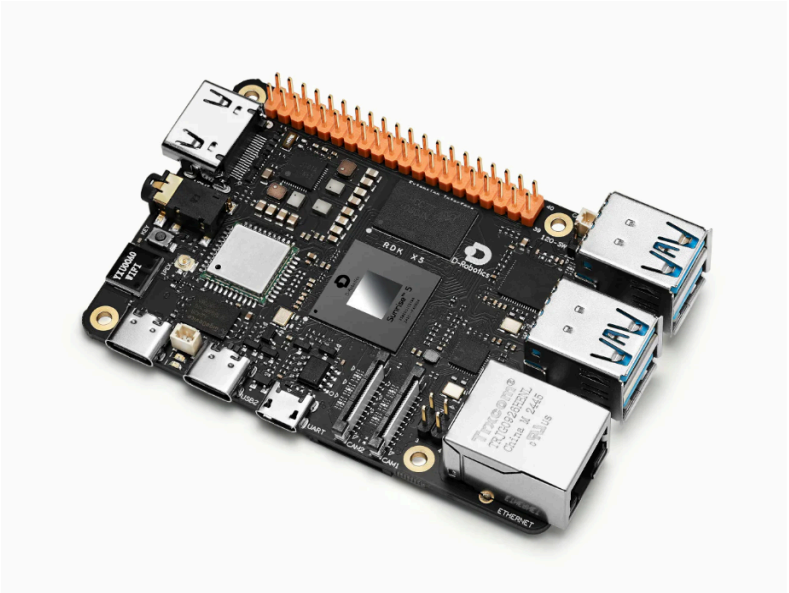
次章では、実際にカメラ映像をBPUで解析し、人（ボディ）認識の結果を**ROS 2**トピックとして受け取る仕組みを学びます。

3:RDK StudioによるRDK X5(シングルボードコンピュータ)のセットアップ

+第3章：RDK Studioによる環境セットアップと活用体験

この章では、ワークショップにおける第1ステップとして必須の「RDK StudioによるRDK X5のセットアップと環境確認」について詳しく解説します。

ROS 2を学ぶ際、シングルボードコンピュータへのインストール作業やトラブル対応は、学習の障壁となることが多くあります。RDK Studioはこの問題を解決するために設計された、GUIベースの開発補助ツールです。



RDK X5シングルボードコンピュータ。Originbot Pro (PD版）の2階部分の基板です。

1. なぜRDK Studioを使うのか

通常、Linux上にROSを構築するには以下の作業が必要です：

- aptやcolconによる依存解決
- シリアル/SSH接続
- ネットワークとWi-Fi設定
- パーミッションと環境変数の調整

RDK Studioはこれらの操作をGUIで統合管理でき、**「ロボット開発をしたい人がLinuxの学習曲線でつまずかないようにする」**ことを目的に設計されています。

RDK X5には、すでにROS 2 Humbleベースの環境（TogetheROS）が構築されたOSイメージがプリインストールされており、すぐに開発を始めることができます。

2. RDK Studioの導入

対応環境

- Windows / macOS（Linux版も開発中）

ダウンロードサイト

https://developer.d-robotics.cc/rdkstudio

Ubuntuユーザーであれば、シリアル接続やSSHでの操作も可能です。

ダウンロードサイト
https://developer.d-robotics.cc/en/rdkstudio

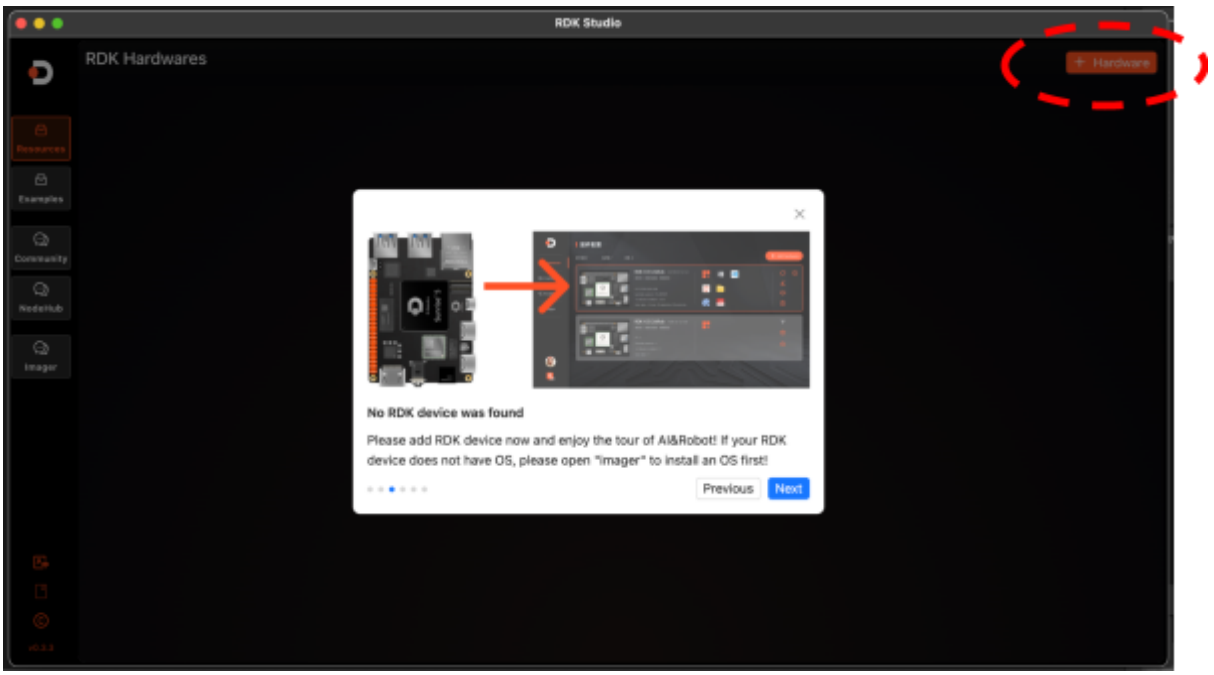
macOSの場合はセキュリティ設定を無効化してアプリを開く必要があります。下記をご参照ください。
https://support.apple.com/ja-jp/guide/mac-help/mh40616/mac

ロボットの電源を入れます。モバイルバッテリーの電源を入れ、次の本体のスイッチを入れます。Ethernetポートが点滅し始めるまで、しばらく待ちます。

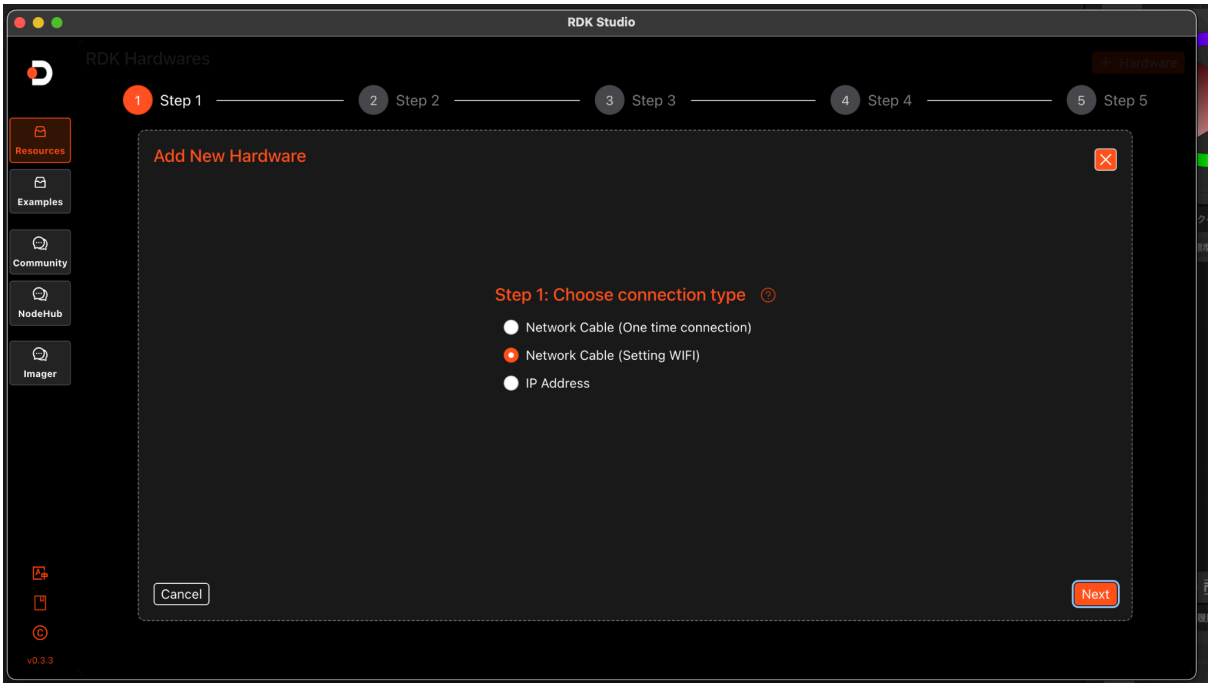
RDK X5へのネットワーク設定

1. まず、RDK X5のイーサネットポートとPC/MacのポートをLANケーブルで繋ぎます。

2. RDK Studioを立ち上げ、画面中央に出てくるチュートリアルをスキップし右上の + HardwareボタンでRDK X5登録を始めます。

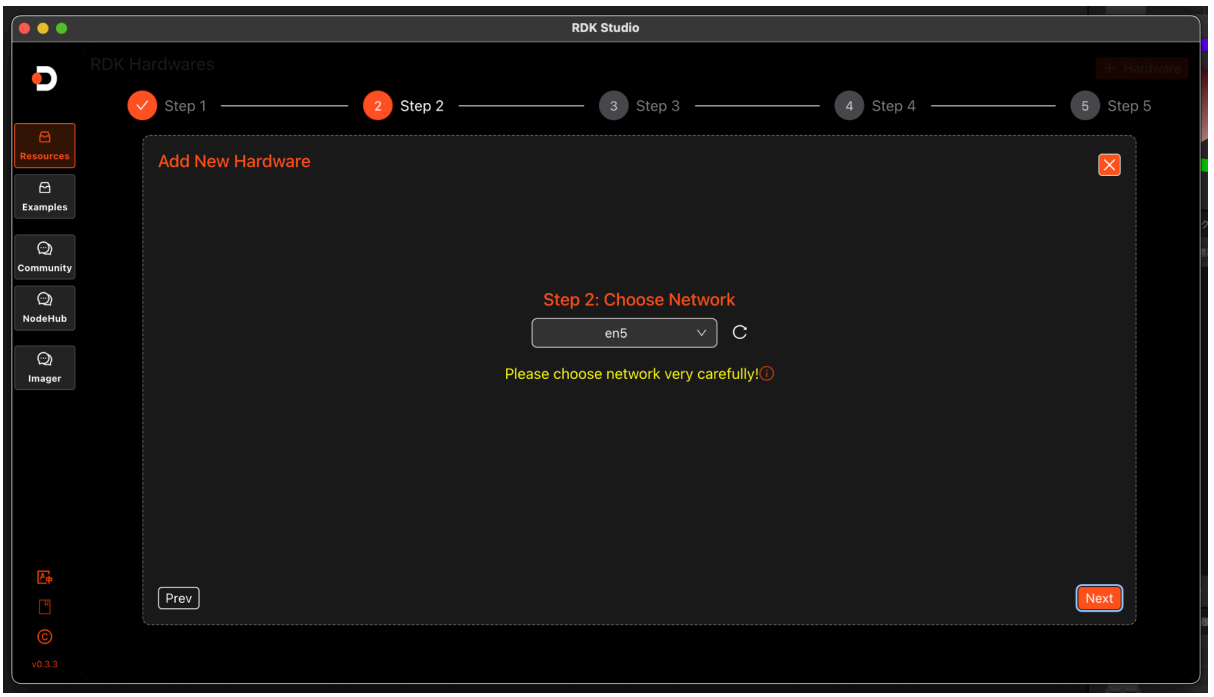


3. Network Cable (Setting WIFI)を選択

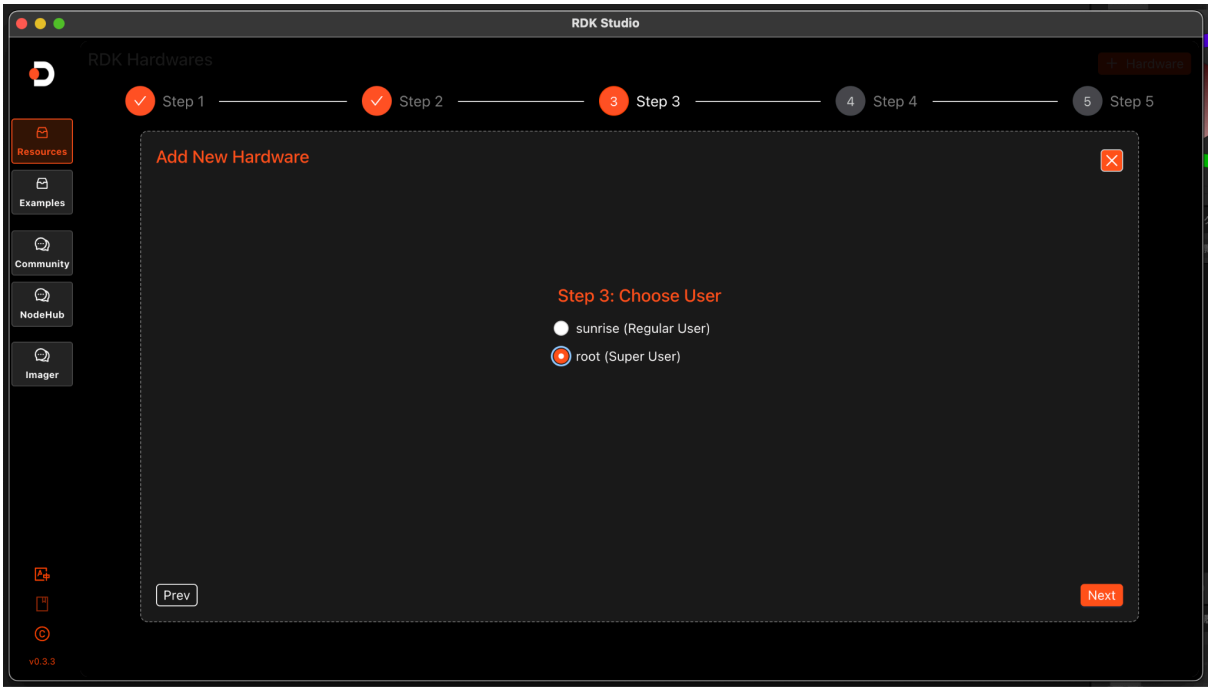


4. Mac:デフォルトで表示されるネットワークポート（例ではen5）をそのままNext。Win:イーサネットx(数字）などが出てくるものを選択する。

   ハードウェアが追加されることで、PC/Macのセキュリティ設定によってはコンピュータそのもののパスワードが要求されます。（RDX X5のために新しいパスワードが必要なのでなく、自分のコンピュータのパスワード）



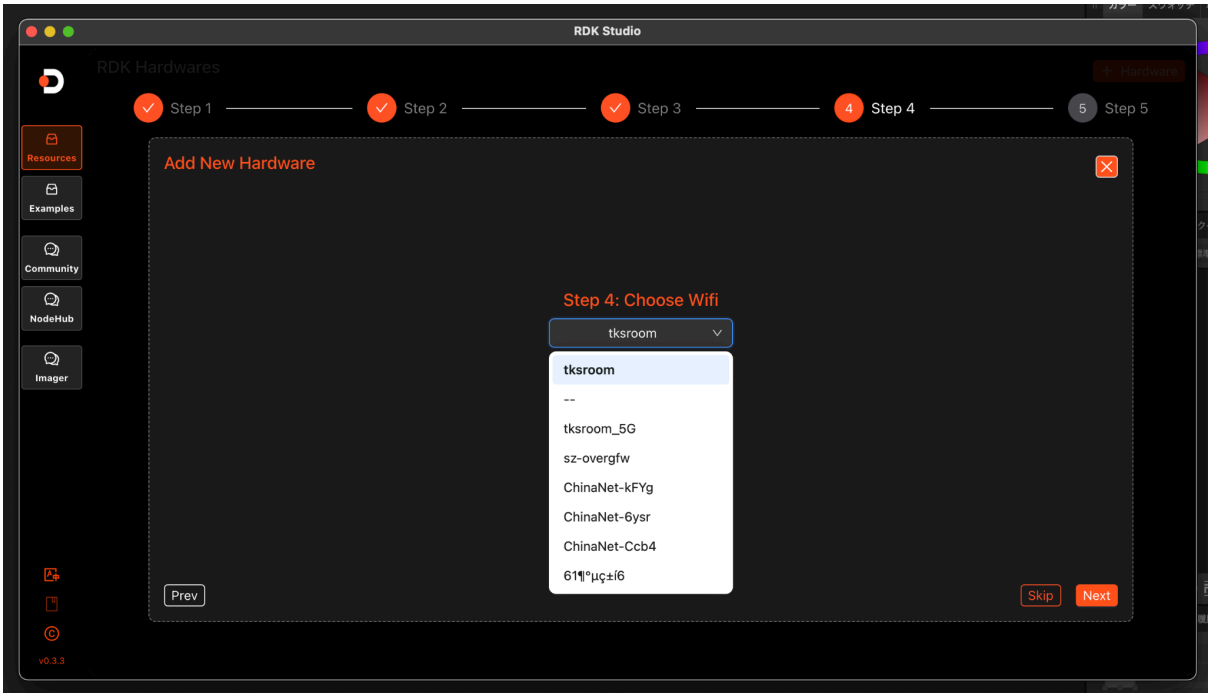
5. RDKシリーズのOS Imageには、スーパーユーザーのroot、ユーザのsunriseの2つがあらかじめ設定されています。(パスワードはそれぞれrootとsunrise)
ログインするユーザを選びましょう。
このワークショップではrootで進行します。



6. RDK X5ボード側のWifiを設定します。スキャンできたSSID一覧が出てくるので、SSIDを選んでパスワードを設定してwifiを支えるようにします。
(ワークショップ会場のWifiはそれぞれ当日に告知)

ここでmacOSの方は「An error occurred!」となって次に進めないことがあります。この場合は、RDK Studio にローカルネットワークへのアクセスを許可する必要があります。

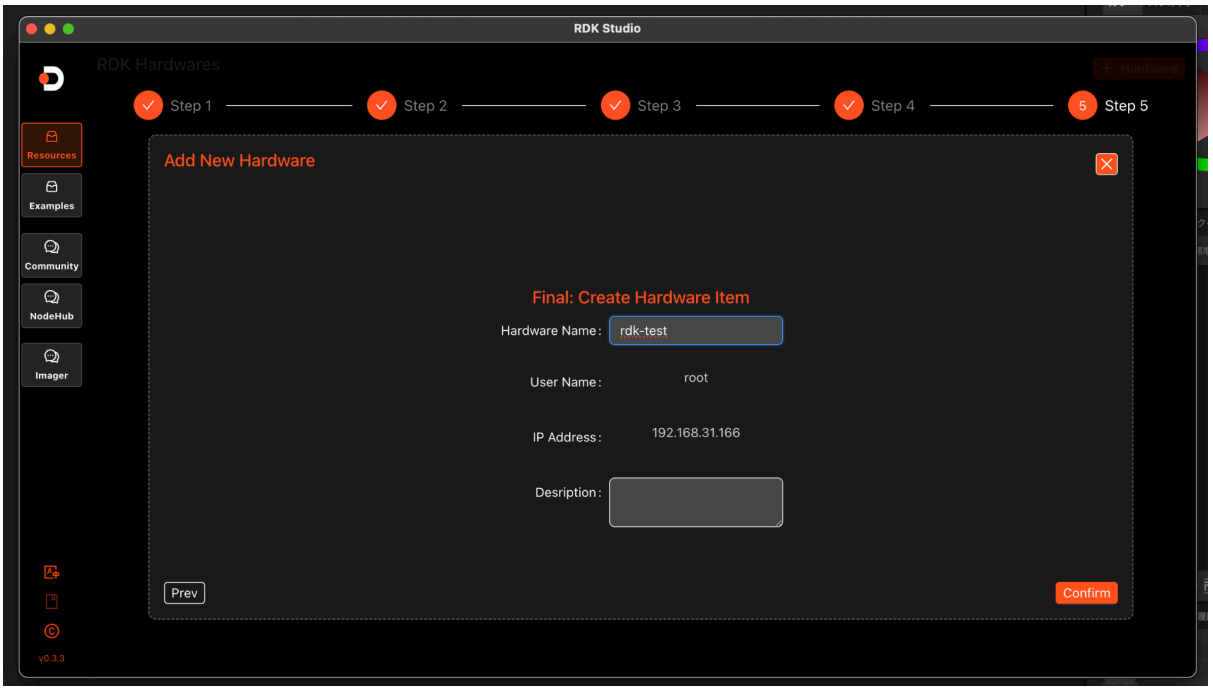
https://support.apple.com/ja-jp/guide/mac-help/mchla4f49138/mac



7. ボードに管理用の名前（各自お好きなものを）をつけて設定完了です。

RDK StudioによるLinuxソフトウェア呼び出し

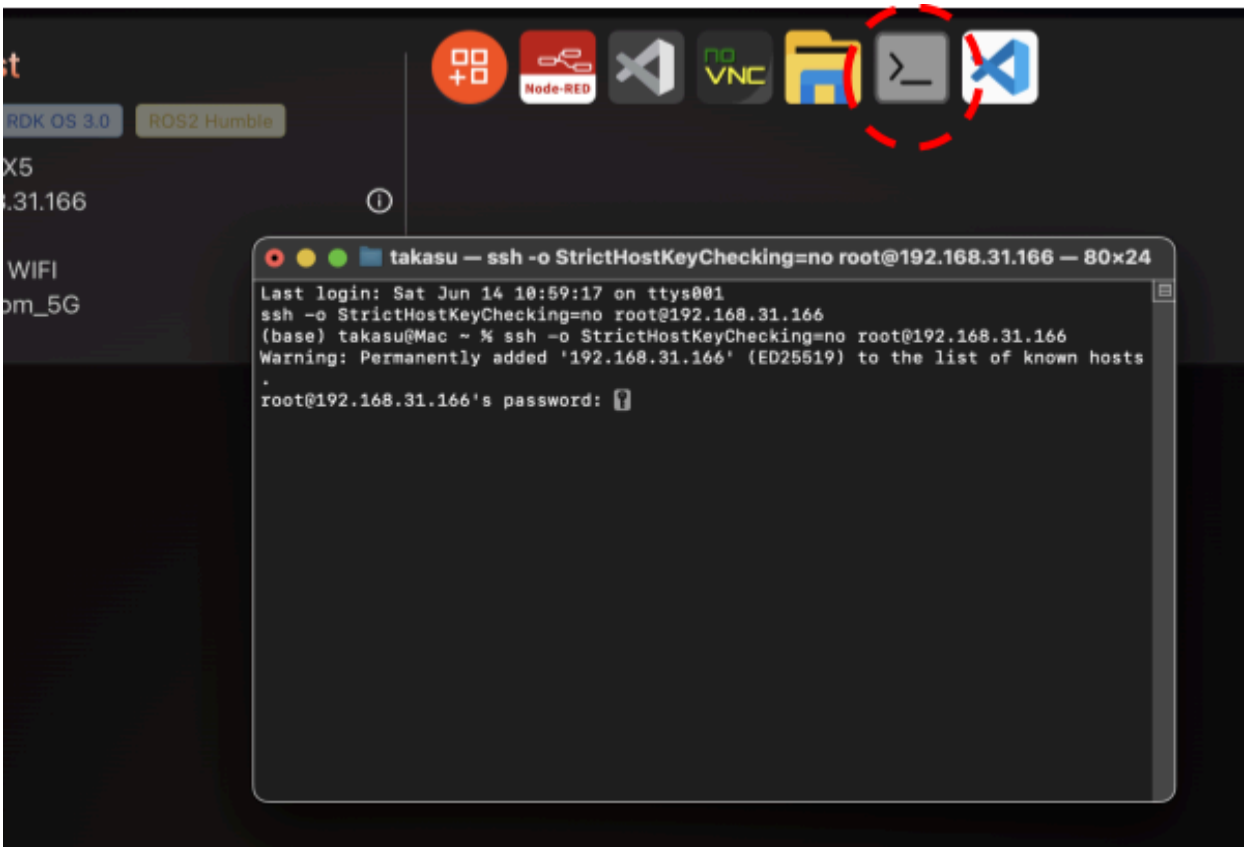
RDK Studio上で、RDK X5内にソフトウェアをインストールす



る、PC/Macから呼び出すことができます。

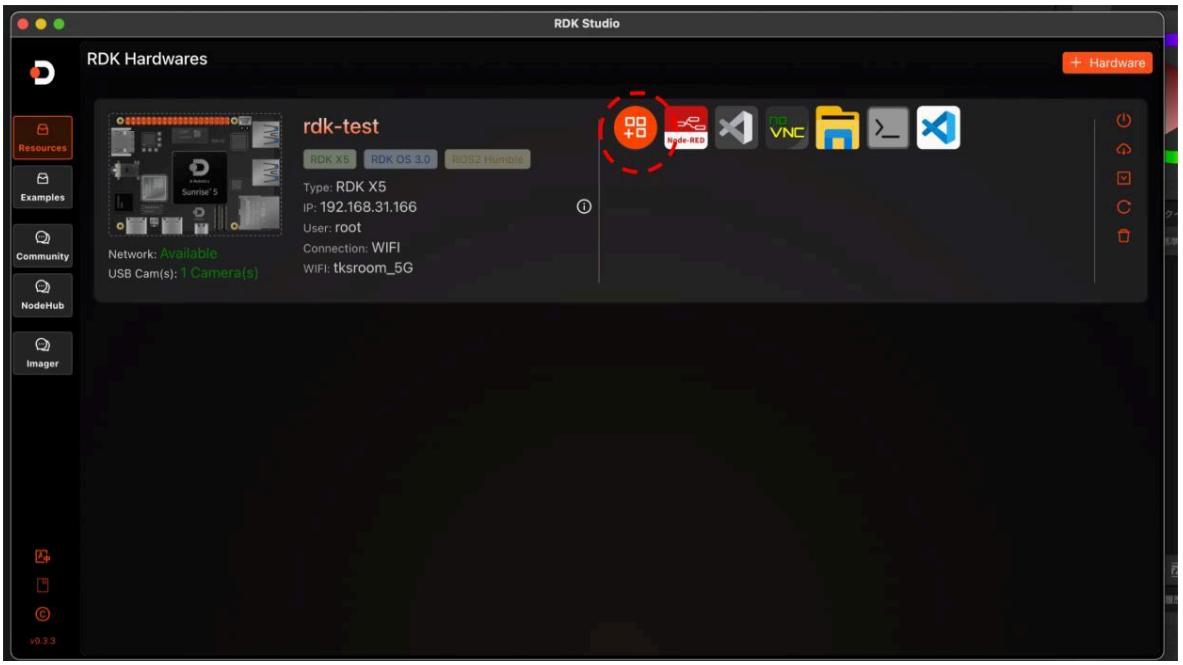
ターミナル呼び出し

このアイコンでterminalを呼び出すことができます。root,sunriseそれぞれのユーザでログインできます。

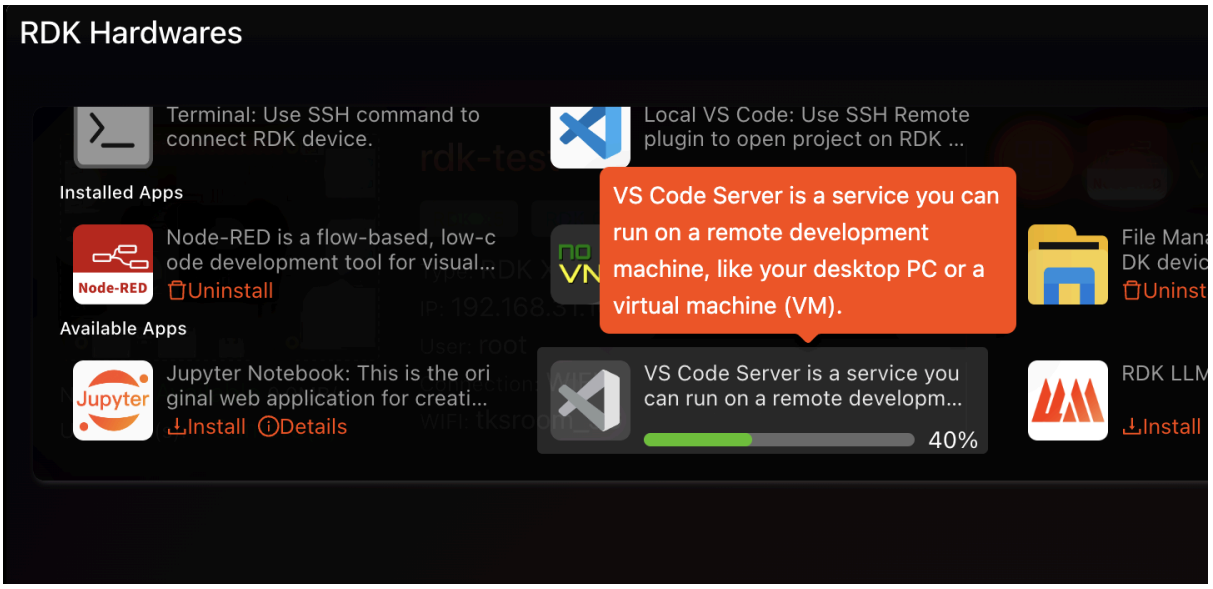


VS Code(サーバ版）

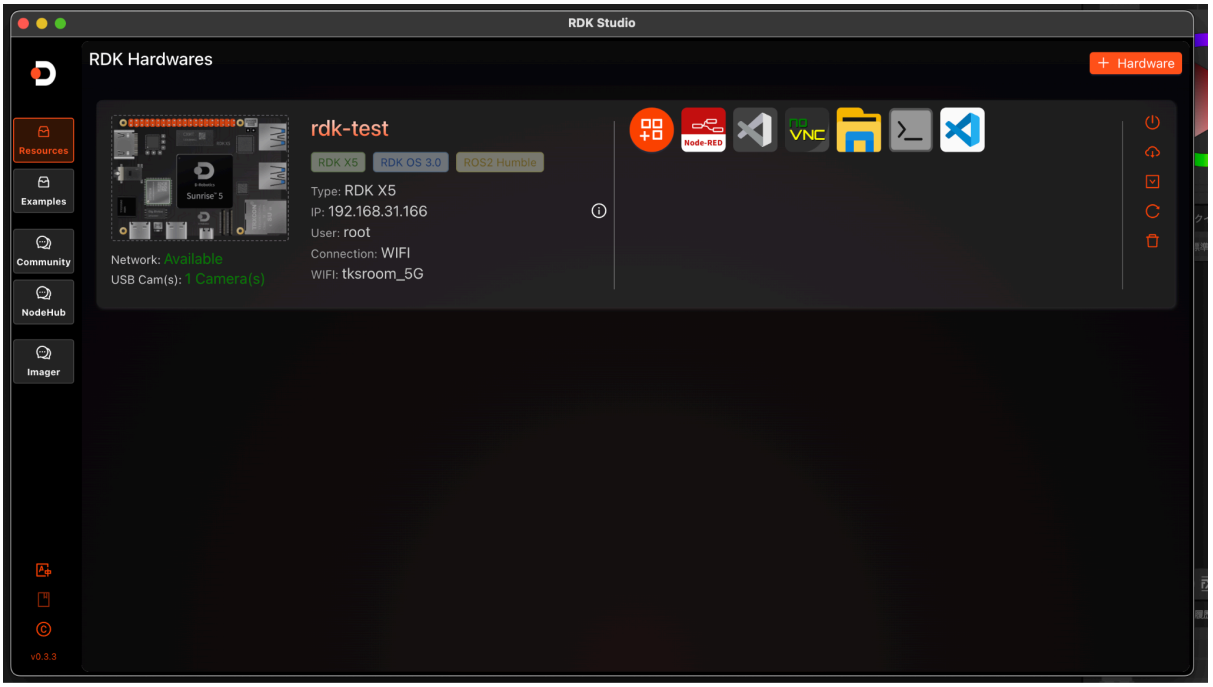
RDK Studio上でRDK X5のLinux環境にソフトウェアをインストールしていくことができます。
（ワークショップ用のSD ImageではすでにVS Codeがインストール済みです）



ワークショップで使うVS Code Serverをインストールしましょう。



インストールが終了すると、RDK Studio内にこのようにVS Code Serverのアイコンが出てきます。
(灰色のほうがワークショップで使用するVS Code Serverのアイコン、白青のものはPC/Mac内のVSCodeと連携するもので、今回は使用しない)



この画面まで表示されたら、次は実際にロボットを動かす部分に入ります。

3. RDK Studioを使ってできること

機能	説明
ノード・トピックの起動管理	`ROS 2 launch` などをGUI操作で代替可能
ROS 2ログ確認	ターミナル・出力ウィンドウでログ確認
VS Codeサーバ開発	BPUノードなどのコード改変と再ビルド体験に最適
ファイル操作支援	FTPやVNCでLinuxファイル管理がGUI操作で可能

この章では、ROS 2を効率的に学ぶための環境として、RDK Studioがいかに重要かを解説しました。

次章では、RDK X5に搭載されたBPUとカメラを使い、人（ボディ）の認識結果を**ROS 2**のトピックで観察する実験を行います。

おまけ：RDK Studioで様々なLinuxソフトウェアインストール

VNCによるLinux Desktop呼び出し

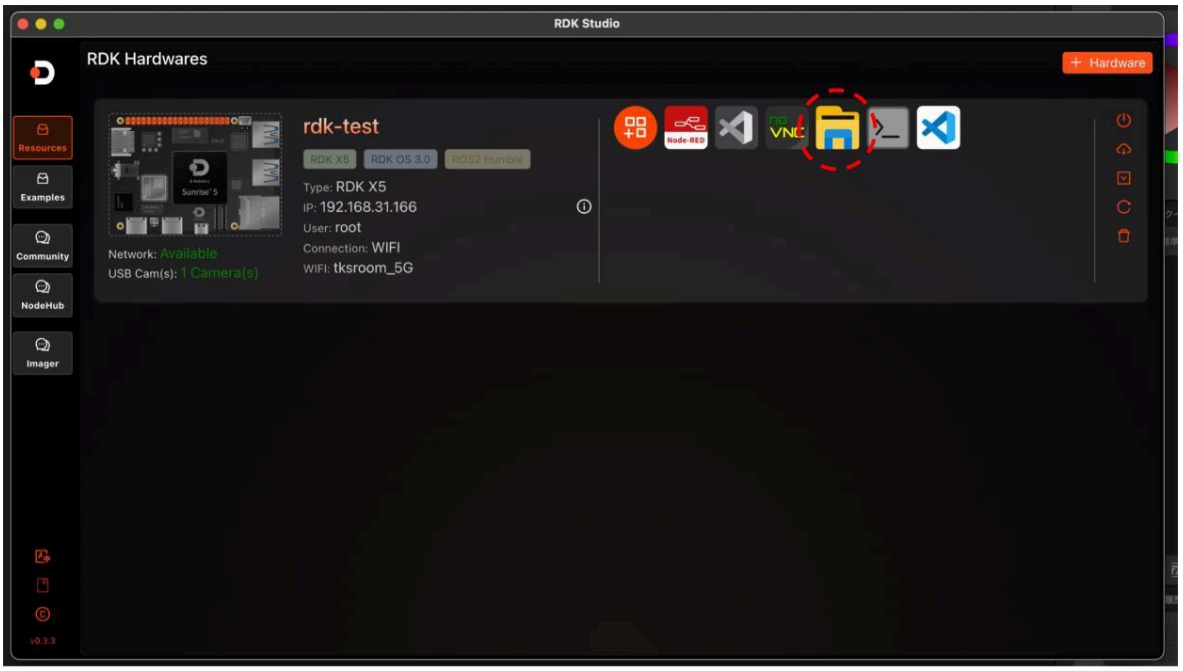
RDK StudioソフトでnoVNCソフトをインストールすることで、RDK X5上のLinux Desktopにリモートログインして、PC/Macから操作することができます。
(ワークショップ用のOS Imageではインストール済みです)

初期パスワードは88888888

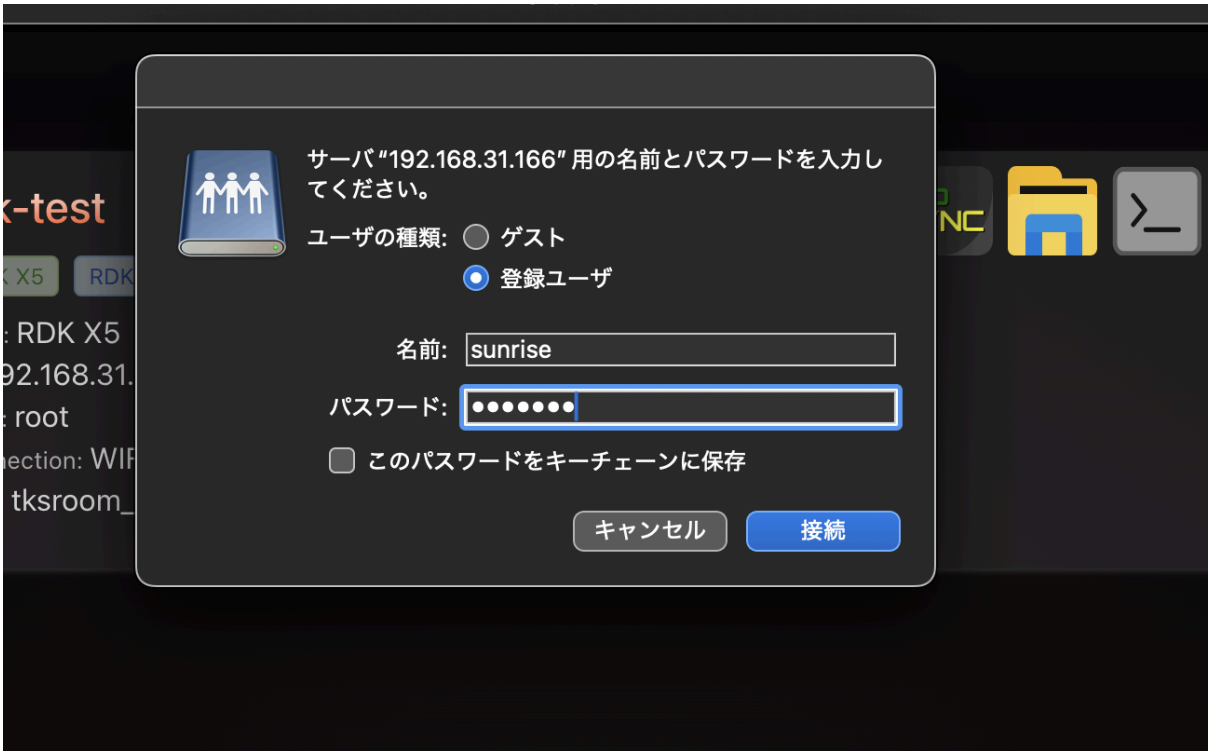
↓PC/Macの画面にLinux Desktopが表示され、そのまま操作できます。ネットワーク越しの操作なので反応が少し遅くはありますが、ファイルマネージャなどで直接シングルボードコンピュータ内のファイルが確認でき、エディタで開けることや、Linuxの設定画面にアクセスできるのは楽です。

FTPサーバによるPC/Macとのファイルやりとり

フォルダアイコンのようなものは、FTPサーバ機能です。この機能をインストールすると、RDK X5内のファイルをPC/Macで確認できます。
（アップロード、ダウンロードもできるはずだが、このドキュメント執筆時6/16/2025ではパーミッションの問題でできない）



↓ID/PasswordはsunriseIPアドレスは設定されているRDK X5のアドレス



↑Mac/PCの画面でRDK X5内のフォルダを確認できる

4:ロボットの動作

第4章：ボディ認識・ジェスチャー認識によるロボット制御体験

この章では、RDK StudioとOriginbot Proを使って、実際にロボットを動かす体験を行います。前半では人（ボディ）認識、後半では手のジェスチャー認識を通じて、カメラとAI、ROS 2ノード群がどのように連携してロボットを動かすかを確認します。

1:ボディ認識を行う

モータードライバ基板を起動させる

RDK Studioからターミナルを開き、以下のコマンドを実行すると、Originbotのモータドライバ基板が動作します。（最初にピーっと音が鳴ります）

```
Shell
ros2 launch originbot_bringup originbot.launch.py
```

ボディ認識ノードの起動

もう一枚、別のターミナルを開き、ボディ認識のプログラムを起動します。

```
Shell
cd /userdata/dev_ws

ros2 launch body_tracking body_tracking_without_gesture.launch.py
```

これで、Originbot上でボディ認識のプログラムが走ります。
RDK Studioで自分のOriginbotのIPアドレスを確認して、ブラウザから
http://{IPアドレス}:8000
と入力してください。ブラウザでボディ認識の様子をみることができます。
このWebページが「This site can’t be reached / このサイトにアクセスできません」などで開けない場合は、SafariやEdgeなど他のWebブラウザでも試してみてください。

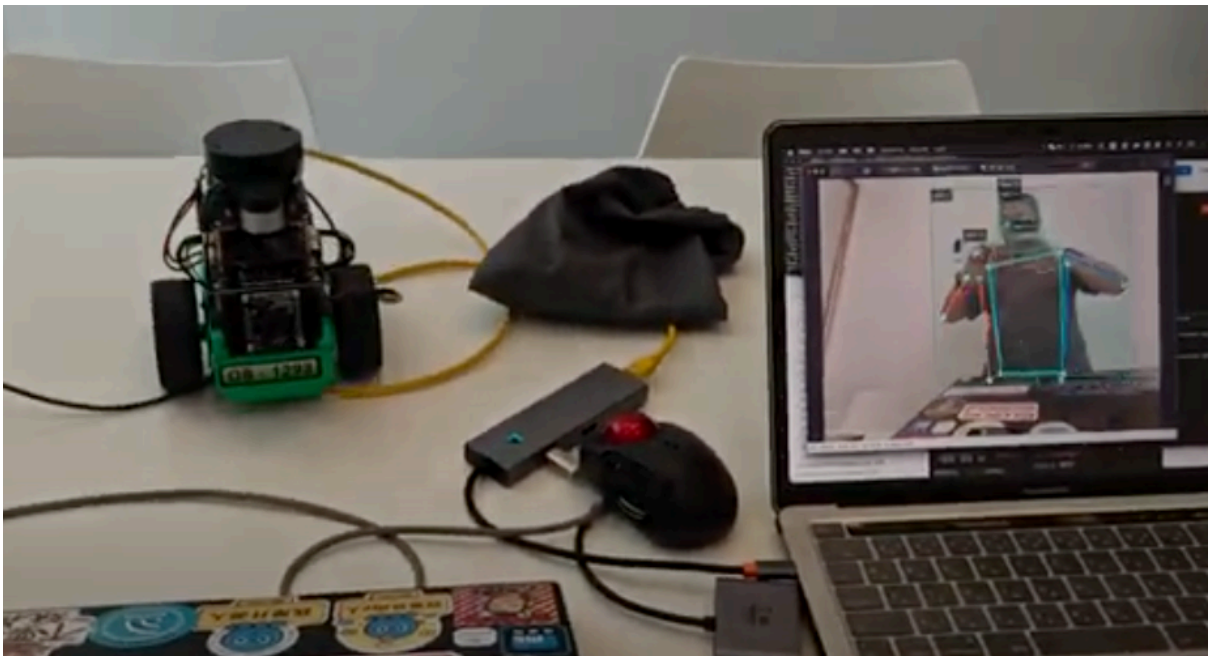
安全上の注意

- Originbotが急に動くことがあります。机の端に置かないでください。
- 認識が不安定な時は光の条件や距離、カメラ角度を調整してみましょう。

動画：Originbot Proボディ認識
https://youtu.be/WLDcYCxSIg8

実行中の注意点

- デモ終了後は 必ず `Ctrl + C` でノードを終了 してください(ターミナルが2枚開いているので、2枚とも）



- 複数のノードを残したままにすると、リソースを圧迫してロボット動作が遅くなる場合があります

2:ジェスチャー認識を行う

ボディ認識と同様に、モータ起動 → ジェスチャーノード起動 → Webで確認、という流れです。

モータードライバ基板を起動させる

RDK Studioからターミナルを開き、以下のコマンドを実行すると、Originbotのモータドライバ基板が動作します。（最初にピーっと音が鳴ります）

```shell
ros2 launch originbot_bringup originbot.launch.py
```

ジェスチャー認識ノードを起動させる

もう一枚、別のターミナルを開き、ジェスチャー認識のプログラムを起動します。

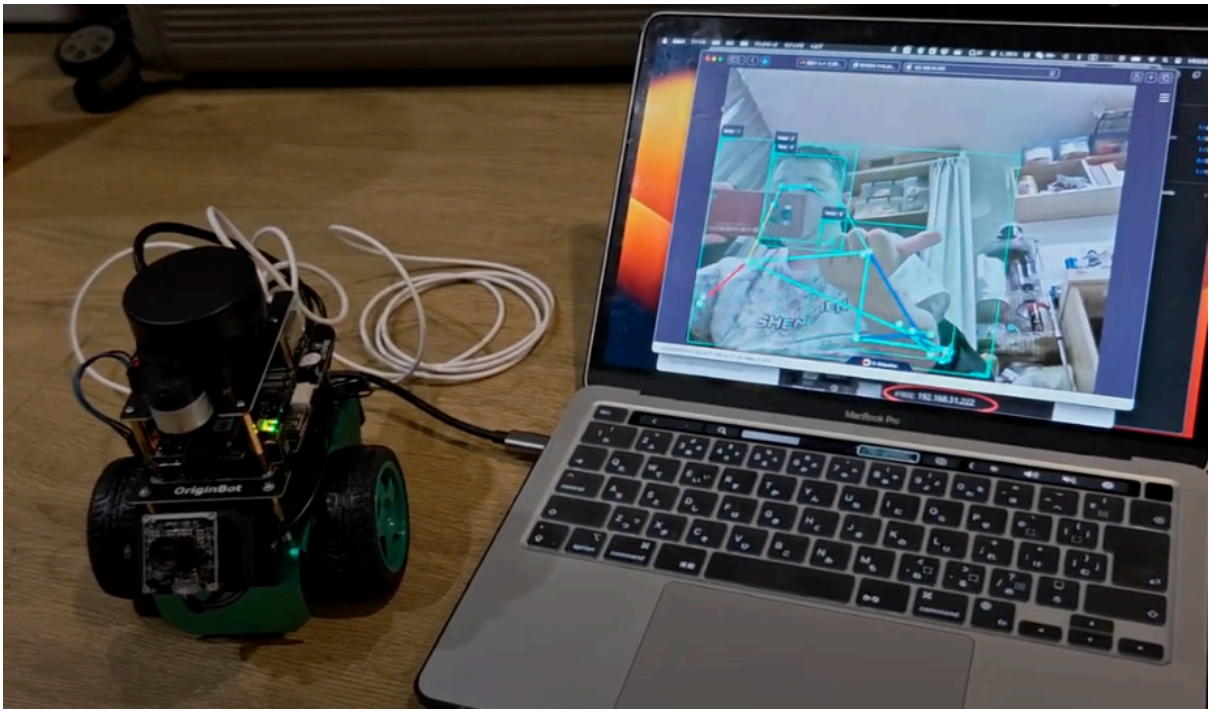
```shell
cd /userdata/dev_ws

ros2 launch gesture_control gesture_control.launch.py
```

これで、Originbot上でジェスチャー認識のプログラムが走ります。
RDK Studioで自分のOriginbotのIPアドレスを確認して、ブラウザから
http://{IPアドレス}:8000
と入力してください。ブラウザで物体認識の様子をみることができます。

安全上の注意

- Originbotが急に動くことがあります。机の端に置かないでください。
- 認識が不安定な時は光の条件や距離、カメラ角度を調整してみましょう。

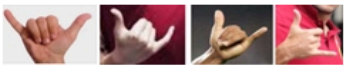


動画：Originbot Proジェスチャー認識
https://youtu.be/d5HasT-tkLw

※デモ終了後はctrl + c でターミナルでのプログラムを終了する（文字が流れなくなる）、その後ターミナルを閉じるようにしてください。

そうでないとジェスチャー認識のプログラムが複数走り続けてRDK X5のリソースを消費し、動作がどんどん遅くなります。

認識できるジェスチャー

手势名称	功能定义	手势动作举例
666手势/Awesome	前进 前進	
yeah/Victory	后退 後退	
大拇指向右/ThumbRight	右转 右回り	
大拇指向左/ThumbLeft	左转 左回り	
OK/Okay	唤醒 wakeup	
手掌/Palm	重置 restart	

3:ジェスチャー制御プログラムの構成とROS 2的解釈

ワークショップで使用した `gesture_control_circle` パッケージのソースコード構造を読み解き、ROS 2におけるノード・トピック・メッセージの観点からプログラムがどのように構成されているかを解説します。

---

1. パッケージの全体構成

```
None
gesture_control_circle/
├── CMakeLists.txt
├── package.xml
├── launch/
│   └── gesture_control_circle.launch.py
├── src/
│   ├── gesture_control_engine.cpp
│   ├── gesture_control_node.cpp
```

```
|    ├── param_node.cpp
|    └── main.cpp
├── include/
|    ├── gesture_control_engine.h
|    ├── gesture_control_node.h
|    └── param_node.h
```

---

2. ノードとクラスの関係

クラス名	役割	ファイル
`GestureControlNode`	`ai_msgs::PerceptionTargets` を受信し `/cmd_vel` を出力	gesture_control_node.cpp
`GestureControlEngine`	ノードの構築と `RunStrategy()` による動作判断	gesture_control_engine.cpp
`ParametersClass`	`move_step` や `rotate_step` を定期的に読み取る	param_node.cpp

`main.cpp` ではこれらを統合し、`rclcpp::spin()` によってノードを実行します。

---

3. Launchファイルとパラメータ構成

```
None
Node(
    package='gesture_control_circle',
    executable='gesture_control_circle',
    parameters=[
        {"ai_msg_sub_topic_name": "/tros_perc_fusion"},
        {"twist_pub_topic_name": "/cmd_vel"},
        {"move_step": 0.5},
        {"rotate_step": 0.5}
    ]
)
```

これらのパラメータは `param_node` 経由でノードに共有され、ジェスチャーによる移動や回転のステップ量として使用されます。

---

4. ROS 2トピック通信の流れ

```
None
graph LR
  Camera --> GES[hand_gesture_detection (外部ノード)]
  GES -->|/tros_perc_fusion| CTRL[gesture_control_node]
  CTRL -->|/cmd_vel| ROBOT[Originbot モーター制御]
```

- `/tros_perc_fusion`:BPUによって認識されたジェスチャー情報が流れるトピック(型:`ai_msgs::PerceptionTargets`)
- `/cmd_vel`:速度指令(型:`geometry_msgs::msg::Twist`)

---

5. `gesture_control_engine.cpp` の詳細解説

このファイルは**ジェスチャー認識結果に応じてロボットの動作を決定する戦略部(エンジン)**であり、以下のような構成になっています。

(1)クラス構成と初期化

```
None
GestureControlEngine::GestureControlEngine()
```

- `GestureControlNode` や `ParametersClass` をインスタンス化
- SmartMessage用のキュー(`smart_queue_`)を生成し、別スレッドで非同期に処理

(2)Smartメッセージの受信と処理

```
None
void GestureControlEngine::FeedSmart(const
PerceptionTargets::ConstSharedPtr &msg)
```

- `/tros_perc_fusion` から受信したBPU出力(ジェスチャー分類)をキューに追加
- キューが一杯のときは古いメッセージを捨ててリソース保護

(3) RunStrategy：ジェスチャーに応じた動作決定

```
None
void GestureControlEngine::RunStrategy(const
PerceptionTargets::ConstSharedPtr &msg)
```

- `ProcessSmart()` により `track_info_` にジェスチャー情報を格納
- `track_info_.gesture` をもとに `DoMove()` または `DoRotate()` を実行

```
None
if (gesture == GestureCtrlType::ThumbRight) → DoRotate(RIGHT)
if (gesture == GestureCtrlType::ThumbLeft) → DoRotate(LEFT)
if (gesture == GestureCtrlType::PinchRotateClockwise) →
DoMove(FRONT)
if (gesture == GestureCtrlType::PinchRotateAntiClockwise) →
DoMove(BACK)
```

(4) DoMove / DoRotate

```
None
void DoMove(int direction, float step);
void DoRotate(int direction, float step);
```

- `geometry_msgs::msg::Twist` を生成し、指定された方向とステップで `/cmd_vel` に送信

---

6. ROS 2学習者向けのポイント

- 本プログラムでは **OpenCV**や**Deep Learning Framework**は一切登場せず、BPUにより生成された推論結果（SmartMessage）をROSノードで扱う形になっています。YOLOなどをBPUで扱う方法については3章で付記しました。
- `gesture_control_node` はシンプルな `Subscriber → Callback → Publisher` の構成
- ノードを設計する際の良い設計パターン（処理ロジックを `engine` に分離）

---

7. 拡張へのヒント

- `gesture_control_engine.cpp` に新しい `GestureCtrlType` を追加して、任意のジェスチャーに対応可能
- `/cmd_vel` 以外の出力（音声、LED、他ノードとの連携）への拡張も可能
- `rqt_graph` で動作中のノード間構成を可視化し、ROS 2の全体像を理解

`gesture_control_engine.cpp` を通じて、AI推論→ROSトピック受信→ロボット制御という流れが、どのように設計されているかを理解できたはずです。

4.デモプログラム構成の理解

ボディ認識とジェスチャー認識は、それぞれ異なるノード構成を持っています。

ボディ認識構成

```
None
graph LR
    Camera -->|画像入力| body_tracking_node
    body_tracking_node -->|人物矩形など| WebServer8000
```

- 使用ノード：`body_tracking_node`
- トピック出力：`/hobot_mono2d_body_detection`（内部でWebに転送）

ジェスチャー認識構成

```
None
graph LR
    Camera -->|画像入力| hand_gesture_detection
    hand_gesture_detection -->|/tros_perc_fusion|
gesture_control_node
    gesture_control_node -->|/cmd_vel| OriginbotMove
```

- 使用ノード：`hand_gesture_detection`（BPU）、`gesture_control_node`
- トピック構成：
  - `/tros_perc_fusion`：ジェスチャー情報（ai_msgs::PerceptionTargets）
  - `/cmd_vel`：ロボットの動作指令（Twist）

この構成により、AI推論（BPU）とROS 2ノード群が連携し、視覚情報を元にリアルタイムな制御が行われます。

この章では、実際にロボットを動かす体験を通じて、**「ROS 2ノード → トピック通信 → ロボット制御」**という流れを体感しました。次章では、ジェスチャー認識ノードの一部コードを変更し、`colcon build` を用いた再構築を体験します。

5:Advanced:ソースを書き換えて動作を変える

第5章：ジェスチャー認識ノードのカスタマイズとビルド体験

この章では、ワークショップの最後のステップとして、ジェスチャー認識ノードの一部コードを改変し、自分の意図した動作を実現する方法を学びます。

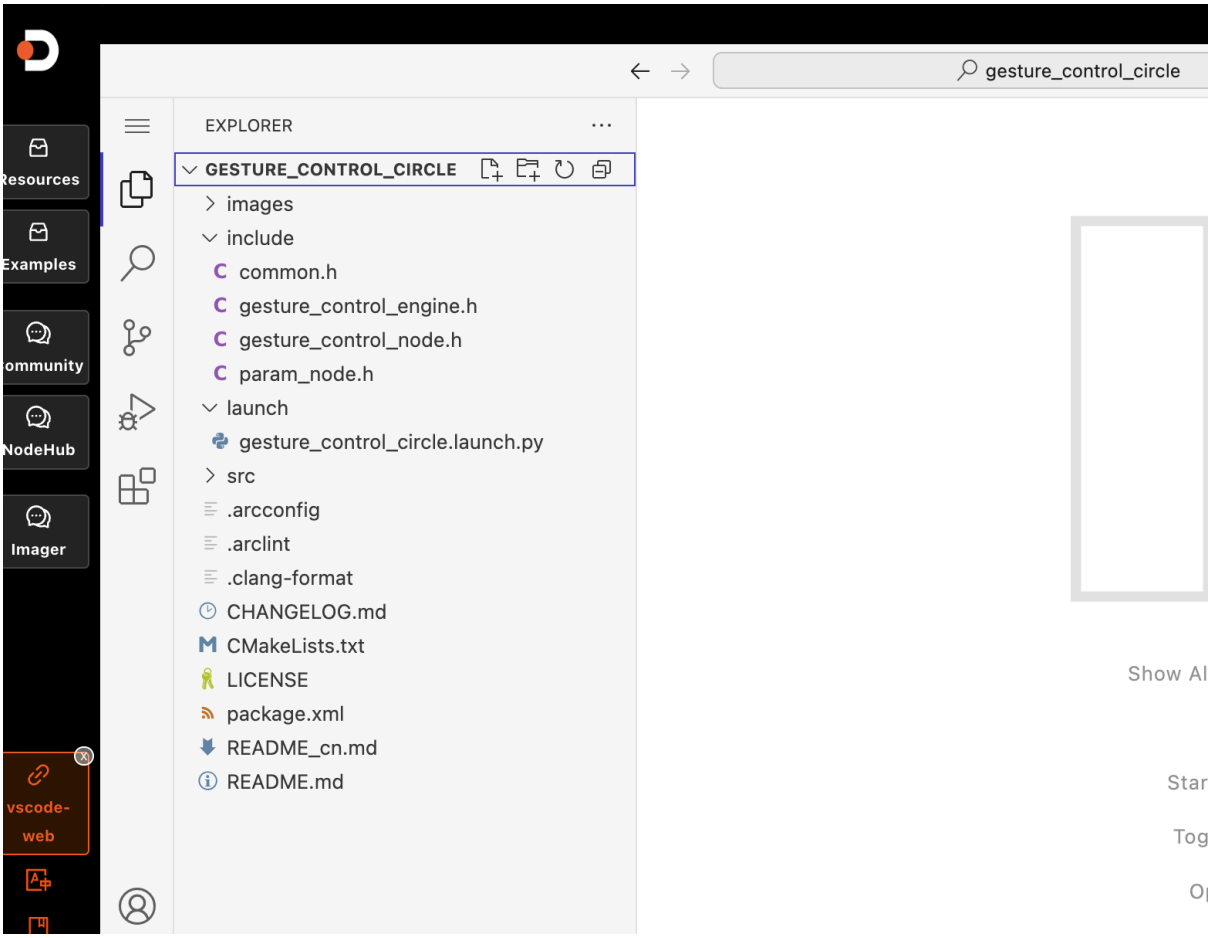
RDK Studioに内蔵された VS Code Server を使って、ROS 2プログラムの編集 → ビルド → 実行までの一連の開発サイクルを体験します。

1.Originbot内のファイル(ソースコード)を開く

RDK StudioのVS Code Serverを開き、「open folder」から

```shell
/userdata/dev_ws/src/gesture_control_circle
```

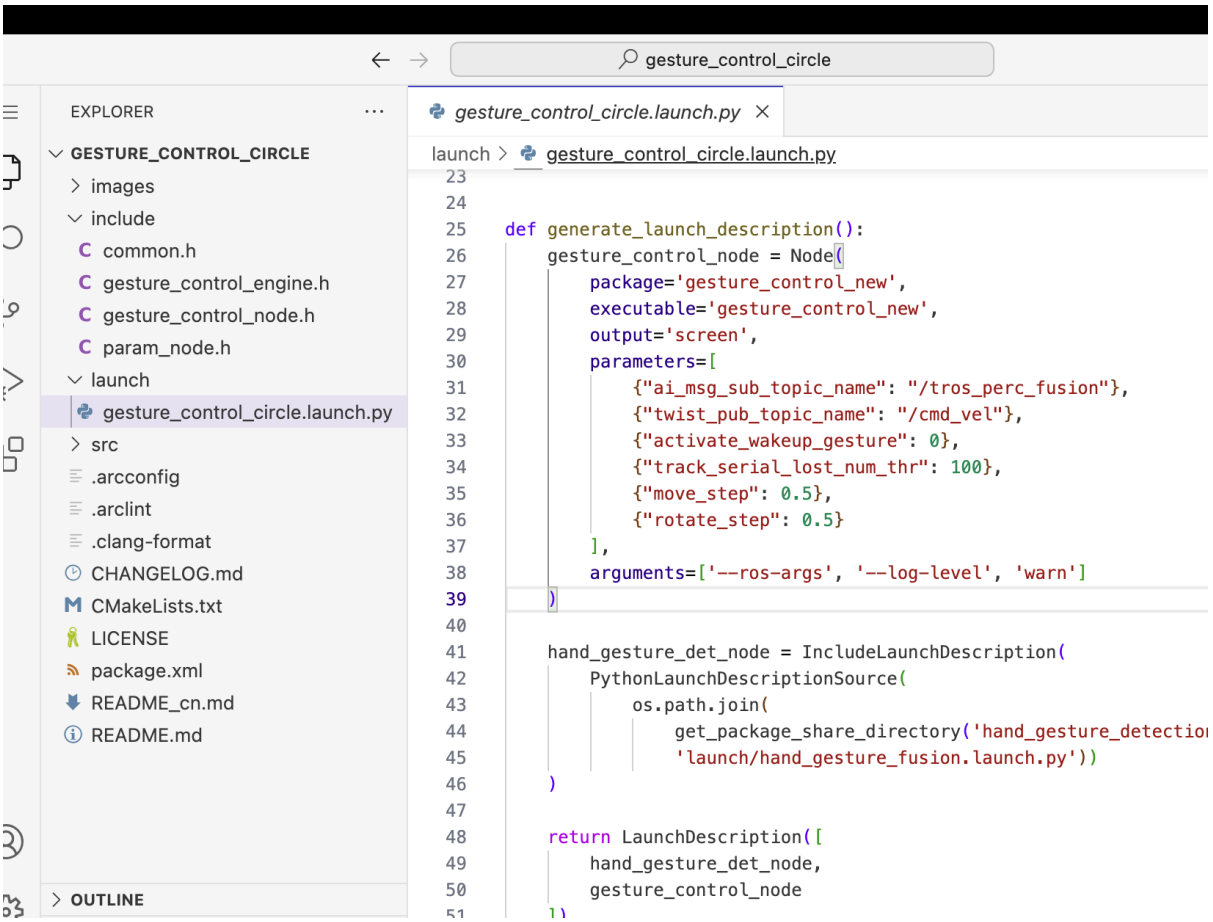
フォルダを開きます。フォルダ内の
gesture_control_circle.launch.py を開きます



フルパスだと
/userdata/dev_ws/src/gesture_control_circle/launch/gesture_control_circle.launch.py

のファイルを開きます。

2. 動作パラメータの編集



ファイル内には以下の記述があります：

```
None
        {"move_step": 0.5},
        {"rotate_step": 0.5}
```

この2行は、ジェスチャー認識によってロボットが移動・回転するステップ量を制御しています。

パラメータ名	説明
move_step	前進・後退の距離（m単位）
rotate_step	回転の角度ステップ（rad単位）

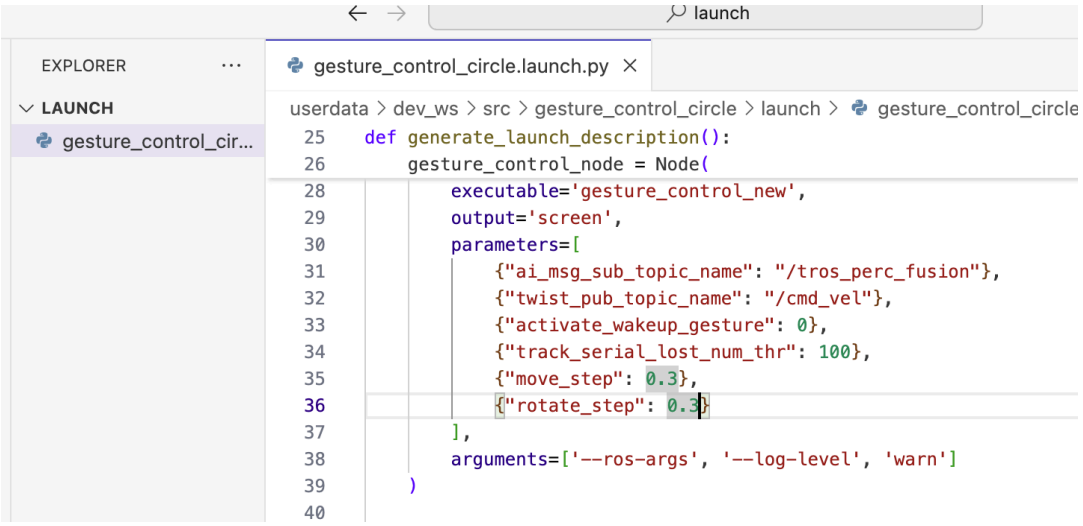
ファイル内の

```python
            {"move_step": 0.5},
            {"rotate_step": 0.5}
```

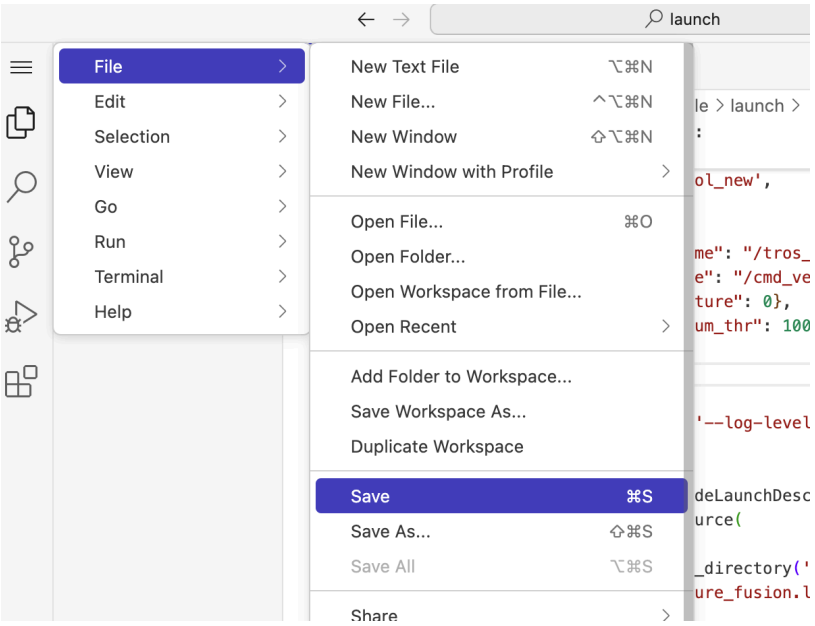
の2行はジェスチャーコントロール時の動作を決める2行です。

それぞれmove_stepが動作距離、rotate_stepがロボット回転の回転の距離を決めます。初期値0.5を大きくすると大きく動くように、小さくすると微妙に動くようになります。

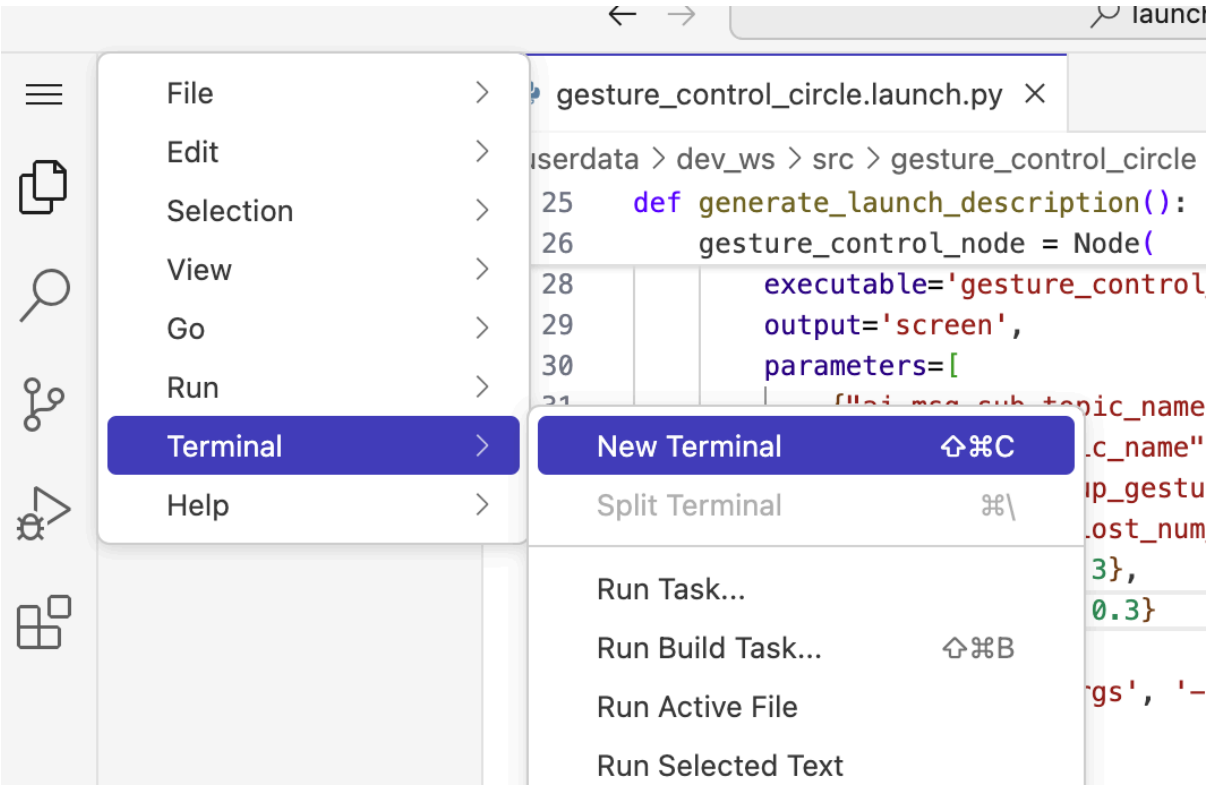
試しに0.3と変更しましょう。



数値を変更したら、saveします。



3. colcon build による再ビルド



VS Code内のTerminalから、編集したファイルを再びビルドします。

```shell
# /userdata/dev_ws
cd /userdata/dev_ws

# root user (sunriseユーザの場合はsudoをつけます)
colcon build --packages-select gesture_control_circle
```

数秒でcolcon buildは終了するはずです。

4.実行して動作を確認する

再度、RDK Studioのターミナルを使ってジェスチャー操作を実行します

再度ターミナルからジェスチャー操作を実行

VS Codeを終了し、RDK Studioからターミナルを開いて、

```shell
#モータードライバボード起動
ros2 launch originbot_base robot.launch.py
```

もう1枚別のターミナルを開き、先ほど編集しbuildしたノードを起動します。

```shell
#ジェスチャーコントロール実行
# /userdata/dev_ws
cd /userdata/dev_ws

ros2 launch gesture_control_circle gesture_control_circle.launch.py
```

※デモ終了後はctrl + c　でターミナルでのプログラムを終了する（文字が流れなくなる）、その後ターミナルを閉じるようにしてください。
そうでないとジェスチャー認識のプログラムが複数走り続けてRDK X5のリソースを消費し、動作がどんどん遅くなります。

まとめ

この章では、ROS 2の開発サイクルの基本（編集→ビルド→実行）を体験しました。特に、**ROS 2**ノードが外部パラメータや**Launch**ファイルを通じてどのように動作制御されているかを、実際に手を動かして理解することができたはずです。

これで、RDK X5上での開発・実行・AI認識とロボット制御の一連の流れを体験できました。

Appendixでは、参考資料などを整理します。

6:Appendix 今回のプログラム構造

—--追加するかも------
今回のワークショップのOS Image
https://drive.google.com/file/d/1t4Nw3a3aRSa5E_T8-ggoYDa93F9qOENd/view?usp=sharing

- Oribinbotで拡張できるROS機能 (SRAMできるとか。デモできるとよりよし）

もっと詳しいマニュアル(ただし中国語）

https://horizonrobotics.feishu.cn/wiki/QzUOwA3JlihBJwkO04LcM4aJn6e

RDKシリーズのBPUで使える（DepthToSpace、Gemmなど）命令と制限一覧
https://developer.d-robotics.cc/rdk_doc/Advanced_development/toolchain_development/intermediate/supported_op_list#rdk-x5%E6%94%AF%E6%8C%81%E7%9A%84onnx%E7%AE%97%E5%AD%90%E5%88%97%E8%A1%A8

路面探索、SLAM, 単眼カメラでのDepth検出など
https://developer.d-robotics.cc/rdk_doc/Robot_development/boxs/driver/parking_perception

Nodehub(開発事例、サンプルコード）
https://developer.d-robotics.cc/nodehub

Yolo 10vで115FPS出した例
https://developer.d-robotics.cc/nodehubdetail/1835246307463438338

タブ 8