目次

1. はじめに

本講座は川崎重工株式会社が公開している **khi_robot** のパッケージを用いて，GUI からロボットの操作・シミュレーション・プログラムによる操作の紹介をしていきます．実機のロボットを扱っていく際にリアルタイムカーネル(**RT**)の構築が必要になって行く為，その構築をする必要があります．本講座の内容をスムーズに行って頂きたい為，扱っていく環境を DockerEngine にまとめてあります．扱っていく ROS バージョンは Noetic となっており，本講座で作成したプログラムは Python3 のバージョンで実行できるようになっています．最後に本講座は ROS を扱った実機ロボットの操作を目的にしている為，ROS による基本動作やプログラムの操作の紹介後，RS007N タイプの実機ロボットの操作方法を紹介していきます．

本講座で扱った環境の紹介を下記の一覧となっています．

- **計算機(PC) 仕様**
  - **OS : Ubuntu 20.04.5 LTS**
    (Ubuntu 20.04.5 ISO ファイルダウンロード先)
    https://releases.ubuntu.com/20.04/ubuntu-20.04.5-desktop-amd64.iso
  - **CPU :**
  - **RealtimeKernel (RT) : linux-5.4.221**
- **ROS**
  - **Version : Noetic**
  - **Catkin : 最新バージョンを使用**
  - **Python: Version 3**
- **Ros Package**
  - **khi_robot (Github): https://github.com/Kawasaki-Robotics/khi_robot.git**
- **DockerEngine**
  - **Version: 5:20.10.21~3-0**
  - **Compose Version: v2.15.1**

本講座で扱う環境構築スクリプトや実機ロボット操作のパッケージ，ソースコードは，

(**Github**): https://github.com/skrjtech/dockerRosKHIRobot.git

に公開されています．こちらを参照しながら，コマンドのコピー＆ペーストを行うことで快適な操作が可能です．

1.1. 本講座の構成

本講座では実機ロボットの操作を紹介していく為に，環境構築からシミュレータの扱い方，Moveit! Commander を使用したプログラミングを行います．実機ロボットの起動手順や操作手順，センサー類の状態取得の解説を行った後に，プログラミングによるワークの整頓操作を行っていきます．

本講座の構成は下記の通りとなっています．

- **はじめに**
- **KHIDuaro/RS007N ロボットタイプの紹介**
- **Moveit! Commander で RS007N の操作**
- **RS007N の実機ロボット環境紹介と接続手順**
- **教師データの取得と設定方法**
- **RS007N の実機ロボットでワーク整頓作業**
- **付録 A 新規パッケージの作成方法**
- **付録 B 独自プログラムの関数解説**
- **Reference**

1.2. 環境構築

環境構築には本講座で用意したリポジトリを Github からダウンロード，もしくは Git で Clone を行うことにより，本講座同様の環境を再現することができます．新規ターミナルを起動して下記のリンクから，

(Github): https://github.com/skrjtech/dockerRosKHIRobot.git

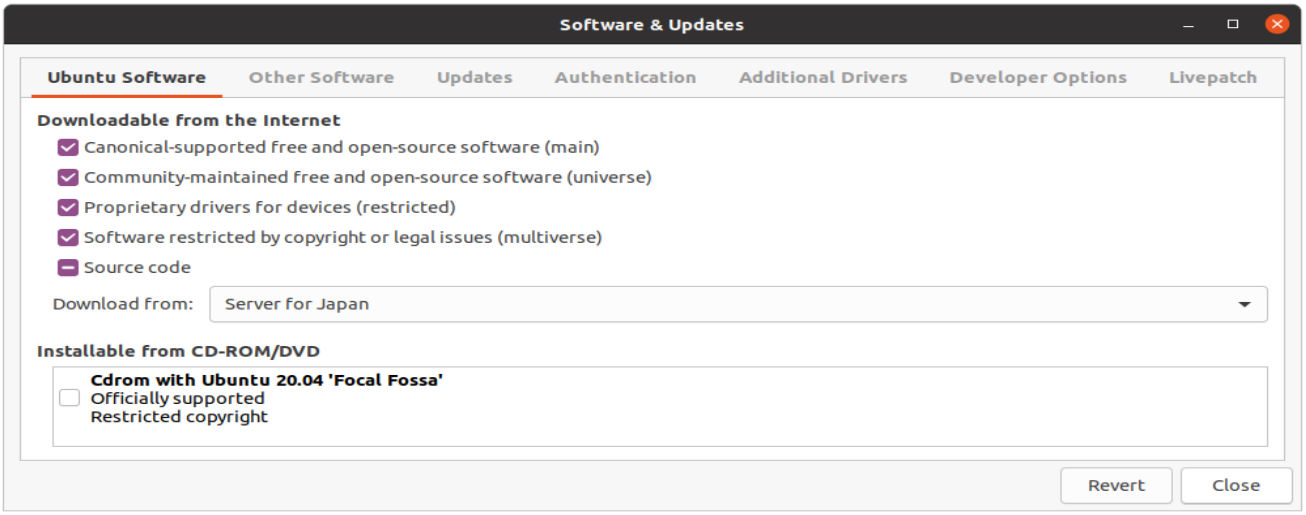
リポジトリを Clone し，dockerRosKHIRobot のディレクトリに移動します．

```
$ git clone https://github.com/skrjtech/dockerRosKHIRobot.git
$ cd dockerRosKHIRobot
```

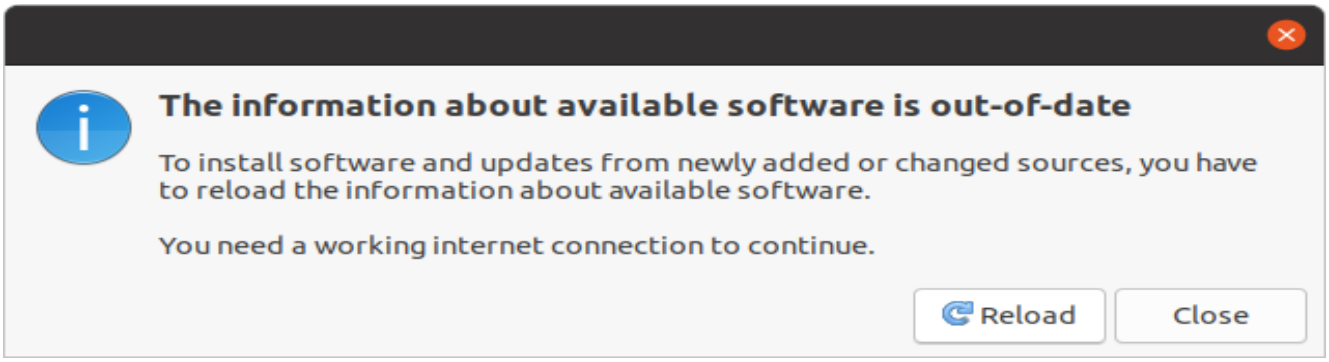
dockerRosKHIRobot の階層については，下記の一覧となっています．

- dockerRosKHIRobot
  - .devcontainer (VSCode で起動するコンテナ情報格納ディレクトリ **本講座では使用しない**)
  - .docker
    - noetic (Dockerfile 格納ディレクトリ)
      - docker-compose-realtime.yml (リアルタイム制御用コンテナ起動情報ファイル)
      - docker-compose.yml (リアルタイム制御無し用コンテナ起動情報ファイル)
      - Dockerfile (コンテナ環境構築用情報ファイル)
  - archives (本講座で作成した独自パッケージとソースコード)
    - khiRobotPack (独自パッケージ格納ディレクトリ)
      - rs007n_type (RS007N 用独自パッケージ 教師データを含めたソースコード)
    - khirobotenv.sh (khi_robot パッケージの環境構築用スクリプト)
  - config (リアルタイムカーネル情報ファイル格納ディレクトリ)
  - docs (本講座内容の公開ディレクトリ)
  - install (環境構築用インストールファイル)
    - docker_install_khirobot_setup.sh (DockerEngine のインストールと Noetic のインストール済み環境イメージのダウンロードのスクリプト)
    - realtime_kernel_install.sh (計算機(PC)のリアルタイムカーネルの構築スクリプト)
  - README.md (本講座概要)
  - build.sh (メイン：環境構築用スクリプト)

以上で dockerRosKHIRobot の内容となります，メインとなる build.sh のスクリプトを起動することで，一連のインストールが開始され環境構築を行うことができます．build.sh の実行の前に「**Software & Update**」をアプリケーション一覧から画面を開き，



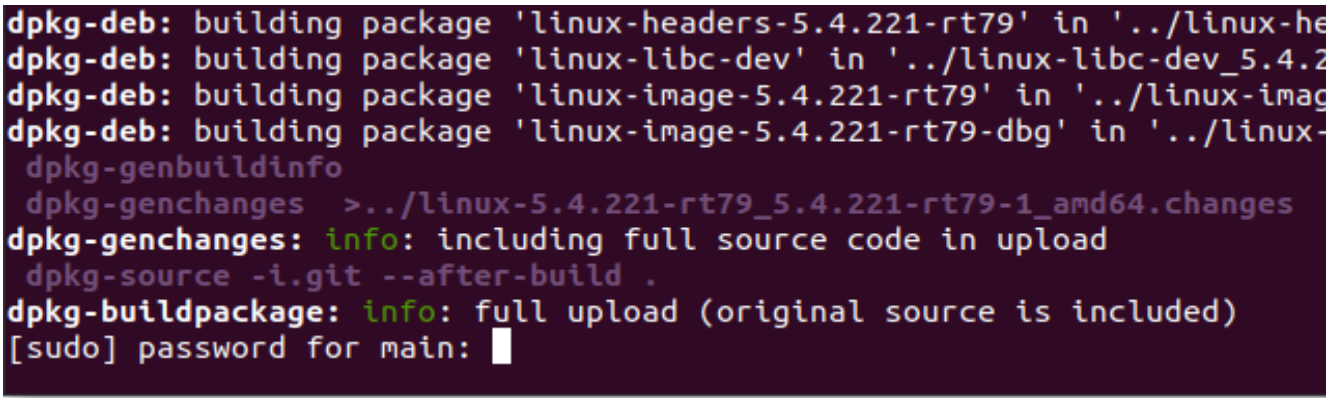
Source Code のチェック欄を有効にし，Close をクリックします．次の画面が出現しますので，**Reload** をクリックし，画面を閉じます．



build.sh を実行して環境構築を開始します．

$. build.sh

途中でパスワード入力が求められるので，使用しているアカウントのパスワードを入力し，先に進みます．リアルタイムカーネル(RT)のインストール時にも下記の様にパスワードが求められるので，こちらもパスワードを入力して行きます．



すべてのインストールが終了しますと，自動的に再起動が実行されます．

再起動されましたらリアルタイムカーネル(RT)で起動するように画面一覧から，下記のように選択します．

- Ubuntu 起動メニュー
  - Ubuntu
  - Advanced options for Ubuntu　<-- こちらを選択します．
    - Ubuntu, with Linux 5.15.0-57-generic
    - Ubuntu, with Linux 5.15.0-57-generic (recovery mode)
    - Ubuntu, with Linux 5.15.0-56-generic
    - Ubuntu, with Linux 5.15.0-56-generic (recovery mode)
    - Ubuntu, with Linux 5.xxx-xxx-rtxxx　<-- こちらを選択し RT で起動
    - Ubuntu, with Linux 5.xxx-xxx-rtxxx (recovery mode)
  - Memory Test (memtest86+)
  - Memory Test (memtest86+, serial consol 115200)
  - ...

1.3. Docker Engine

本講座では，DockerEngine を ROS 環境の土台として使用していきます．Docker Engine を使用するのは，独自パッケージの作成の操作不良や，何かしらの不具合で動作が出来なくなってしまった時に Docker Engin のコンテナの作り直しをすることで，環境の再構築することなく元の環境を呼び出せるメリットとして使用していきます．前節で解説した環境構築で DockerEngine がインストールされると同時に本講座で扱う**コンテナイメージファイル**がダウンロードされます．dockerRosKHIRobot ディレクトリの下で新規ターミナルを起動し，次のコマンド入力でイメージファイルが存在しているかを確認します．

```
$ docker images
```

上記のコマンドで下記のように表示されます．

skrjtech/khi_robot:noetic のイメージファイルの確認が取れましたら，コンテナの起動をしていきます．

```
$ docker compose -f .docker/noetic/docker-compose.yml up -d
```

コマンド実行後下記のように表示されます．

次のコマンドで，コンテナ一覧を表示し実際にコンテナが起動されてかを確認します．

```
$ docker ps -a
```

次のように表示されましたら，コンテナが起動されている状態となります．

次にコンテナ内に入るには，

```
$ docker exec -it ros_noetic bash
```

のコマンドで中に入ることができ下記のようになります．

コンテナから抜けるには，exit と入力し，元のターミナルに戻ります．

最後にコンテナの削除や環境のリセットを行う際には，コンテナの削除動作をします．下記のコマンドで，

```
$ docker rm -f ros_noetic
```

コンテナの削除を行うことができ，もう一度コンテナ起動を行うと初期状態のコンテナを起動することができます．

1.4. ROS Moveit!・Gazebo シミュレータについて

本講座で扱う ROS シミュレータの種類の紹介をしていきます.

- Ros シミュレータ
  - Moveit! : ロボット操作用モーションプランニング・ナビゲーション
  - Gazebo : ロボットの姿勢や動作の可視化・仮想空間で疑似障害物の構築

Moveit!では，GUI の操作を行うことで，実機ロボットによるモーション動作や移動などのプランニングを行うことができ，また，Gazebo と連携させることで仮想空間でロボットのモーションの確認を行うことができます.

2. KHIDuaro/RS007N ロボットタイプの紹介

Github で公開されている khi_robot のパッケージ内には，複数のロボットタイプが存在します．今回はその中で KHIDuaro と実機制御として使用する RS007N のロボットタイプを紹介していきます．また，紹介の際に，GUI による操作で Gazebo と Rviz のツールを使用して，Gazebo で起動したロボットタイプの制御を Rviz で行っていく紹介をしていきます．

2.1. KHIDuaro

KHIDuaro のシミュレーションを行って行く前に，Docker コンテナとホストとなる計算機(PC)の間で GUI の通信を行えるように設定をしていきます．dockerRosKHIRobot のディレクトリの下で新規ターミナルを開き，

```
$ xhost +local:
```

のコマンドを入力します．次に，コンテナを起動していきます．

```
$ docker compose -f .docker/noetic/docker-compose.yml up -d
```

コンテナの起動を行ったらコンテナ内に入ります．

```
$ docker exec -it ros_noetic bash
```

Docker よる GUI の起動を確認します．次のコマンドで，

```
$ xeyes
```

xeyes のアプリケーションが起動され下記の画面が出現します．

画面を閉じる際は，画面から閉じるか，もしくは，ターミナルから強制停止の(Ctrl + C)でアプリケーションを閉じることができます．

ここまで正常に動作が行えたら，KHIDuaro を立ち上げていきます．複数のターミナルを開いていく為，tmux のツールを使用していきます．次のコマンドで，

```
$ tmux
```

次の画面に切り替わります．

切り替わった画面で KHIRobot のノードを起動します．

```
$ roslaunch khi_robot_bringup duaro_bringup.launch
```

下記のようにノードが起動されていきます．

別のターミナルを開いて，Gazebo の起動をします．ターミナル上で「Ctrl + b」を押した後に「C」を押すと新規タブのターミナルが開いていきます．

次のコマンドで，

```
$ roslaunch khi_duaro_gazebo duaro_world.launch
```

ロボット情報が含めた Gazebo 環境が開きます．

次に Rviz を起動し，GUI のよる動作制御をしていきます．次のコマンドで，

```
$ roslaunch khi_duaro_moveit_config moveit_planning_execution.launch
```

Rviz が起動されます．

ロボットの右アーム先端の水色球体をマウス操作で移動させ，赤枠内の「**Plan&Execute**」をクリックすると動か

した方向にアームが移動されます.

最後に起動された GUI とノードを閉じます．新規タブのターミナルを起動し，

```
$ tmux kill-server
```

のコマンドを入力することにより，tmux 上で立ち上げたすべてのターミナルが強制的に閉じていきます.
以上で KHIDuaro の紹介を終わります.

2.2. RS007N

KHIDuaro の起動方法と同様に RS007N の起動をしていきます．RS007N では今後実機制御の解説をしていく為, 単純な起動方法を紹介していきます. はじめに, tmux を開いて, RS007N のノードを起動します.

```
$ tmux
```

新規タブのターミナルでノードの起動,

```
$ roslaunch khi_roboto_bringup rs007n_bringup.launch
```

起動後に新規タブを立ち上げ Gazebo と Rviz を起動していきます.

```
$ roslaunch khi_rs_gazebo rs007n_world.launch
```

新規タブで Rviz を起動.

```
$ roslaunch khi_rs007n_moveit_config moveit_planning_execution.launch
```

3. Moveit! Commander で RS007N の操作

紹介したロボットタイプの制御にはGUIによる制御の他にプログラムによる制御ができます. Moveit！のAPIを使用することでロボットの各関節の制御や位置制御，姿勢制御が行うことができ，また，プランニングによる動作計画を決めることで一連の作業を行うことができます.

次のリンク先で,

https://robo-marc.github.io/moveit_tutorial/

NEDO 特別講座として Moveit!のチュートリアルが詳細に解説されていますので，詳しく知りたい方はこちらをご参照ください．本講座では実機として動かす RS007N の単純な XYZ 座標の移動や目的の位置に向かう制御方法のプログラムを紹介していきます.

RS007N の関節制御をしていく為に，マスター情報の起動をしていきます．tmux を起動し,

```
$ tmux
```

roslaunch で RS007N マスターを起動します.

```
$ roslaunch khi_robot_bringup rs007n_bringup.launch
```

新規タブを開き，Gazebo を起動します.

```
$ roslaunch khi_rs_gazebo rs007n_world.launch
```

最後に各関節の情報を取得する為に Rviz を起動します.

```
$ roslaunch khi_rs007n_moveit_config move_group.launch
```

対話的なプログラムを行っていく為，Python でよく使われる Jupyter Lab を使っていきます.

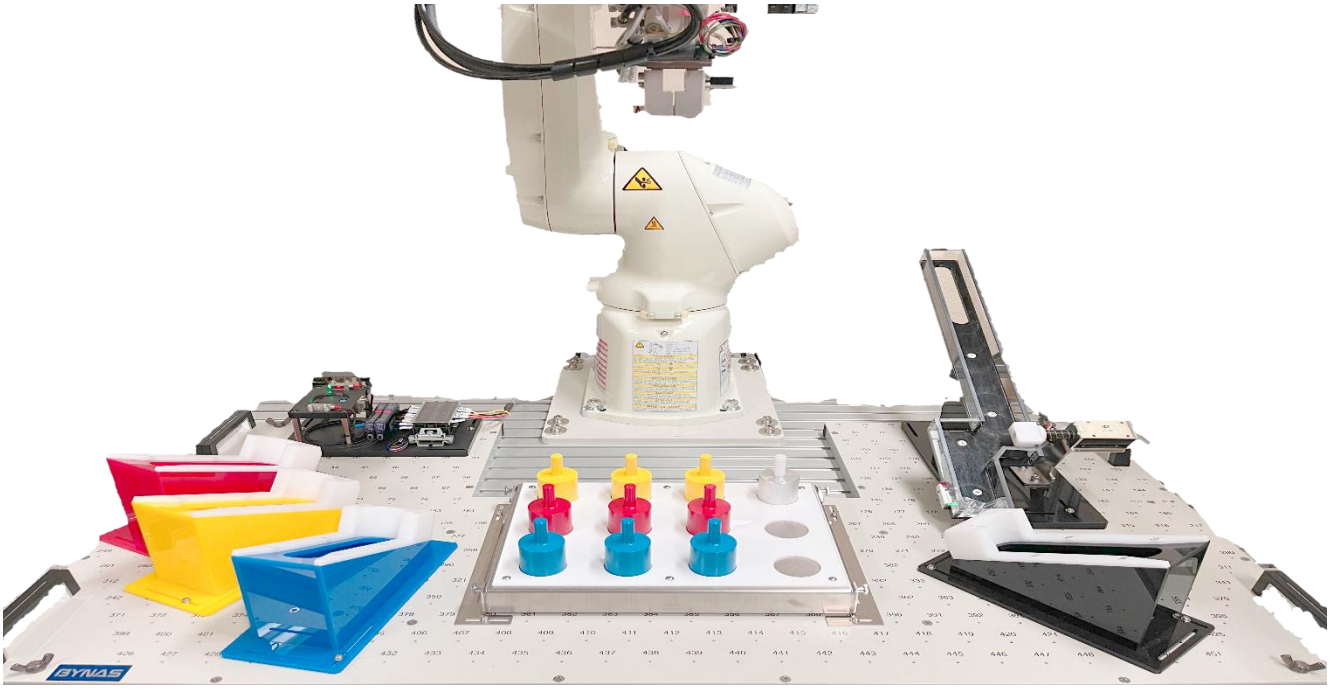
新規タブで次のコマンドを入力していきます.

```
$ cd /khi_robot/src/khiRobotPack/JupyterDir
$ jupyter-lab --allow-root --NotebookApp.token=''
```

JupyterLab の起動を行うと自動的にブラウザが起動します．もし自動的に起動しない場合は，ブラウザを起動させ，検索欄で「http://localhost:8888/lab」と検索します.

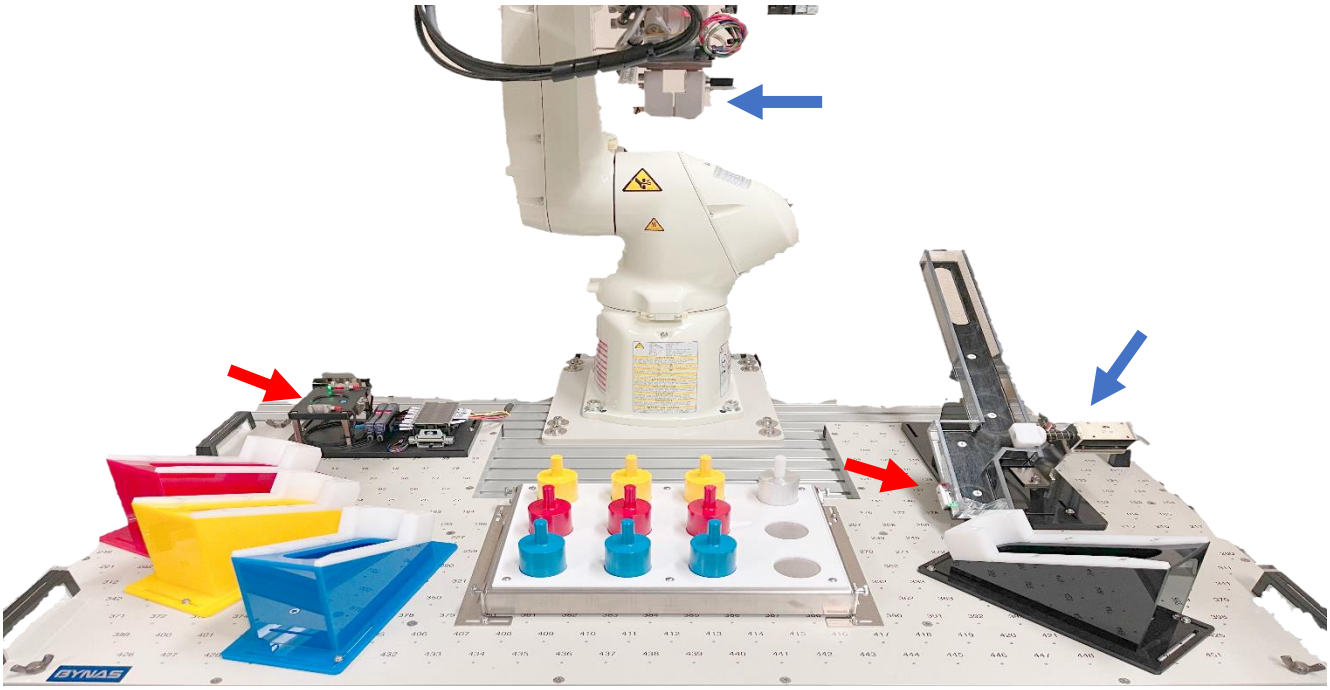
4.　RS007N の実機ロボット環境紹介と接続手順

本講座の目的である実機の起動と接続方法，他に各種センサーの状態取得の解説をしていきます．



上記のイメージは，本講座で使用した RS007N の環境で， Gazebo のシミュレーション上にはなかった**ハンド**がロボットに取り付けられています．またその周辺にはセンサー類が設置されており，センサーの状態を取得する関数が khi_robot のパッケージに用意され，関数に引数を指定することでシグナルに接続されているセンサーの状態を取得することが出来ます．

下記のイメージで**赤い矢印**はセンサーを示しており，**青い矢印**はロボットハンドと供給ユニット(ソレノイド)を示しています．これらの装置を用いて 3 色のワークとアルミワークの色判別，そして，ハンドの開閉やワークの供給の操作を紹介していきます．



4.1. RS007N 実機ロボットの起動手順

実機ロボットの起動手順の紹介を行う前に，下記のイメージのコントローラの**非常停止ボタン**が押されている状態になっているかを確認します．もし押されていなければ非常停止ボタンを押していきます．次に，「**ティーチモード**」と「**リピートモード**」の切り替えスイッチがティーチモードになっているかを確認します．こちらも，ティーチモードになっていなければ切り替えていきます．

次にティーチペンダントもコントローラと同様に非常停止ボタンが押されている状態とティーチモードになっているかを確認します．

どちらとも確認がとれたら，下記のイメージのようにコントローラのレバーを右にシフトし，実機ロボットを起動させます．

起動が完了しましたら，コントローラの非常停止ボタンを解除し，ティーチモードからリピートモードに切り替えます．

4.2. ROS と実機ロボットの接続方法

実機ロボットの起動を行ったら，次に ROS と接続をしていきます．接続の際に計算機とコントローラの通信がローカルであることが条件です．ティーチペンダントから IP アドレスを確認していきます．ティーチペンダントの画面操作から「システムメニュー」「ネットワーク設定」の順に移動し，IP アドレスの欄を確認します．

次に，ティーチペンダントの非常停止ボタンを解除し，ティーチモードをリピートモードに切り替えます．

上記の手順の確認ができたら，計算機から新規ターミナルを開き，tmux を起動します．

```
$ tmux
```

tmux を起動しましたら，確認した IP アドレスを下記のコマンドのように指定し，マスタノードを起動します．

```
$ roslaunch khi_robot_bringup rs007n_bringup.launch ip:=(確認した IP アドレスを入力)
```

上記のコマンドの実行で，

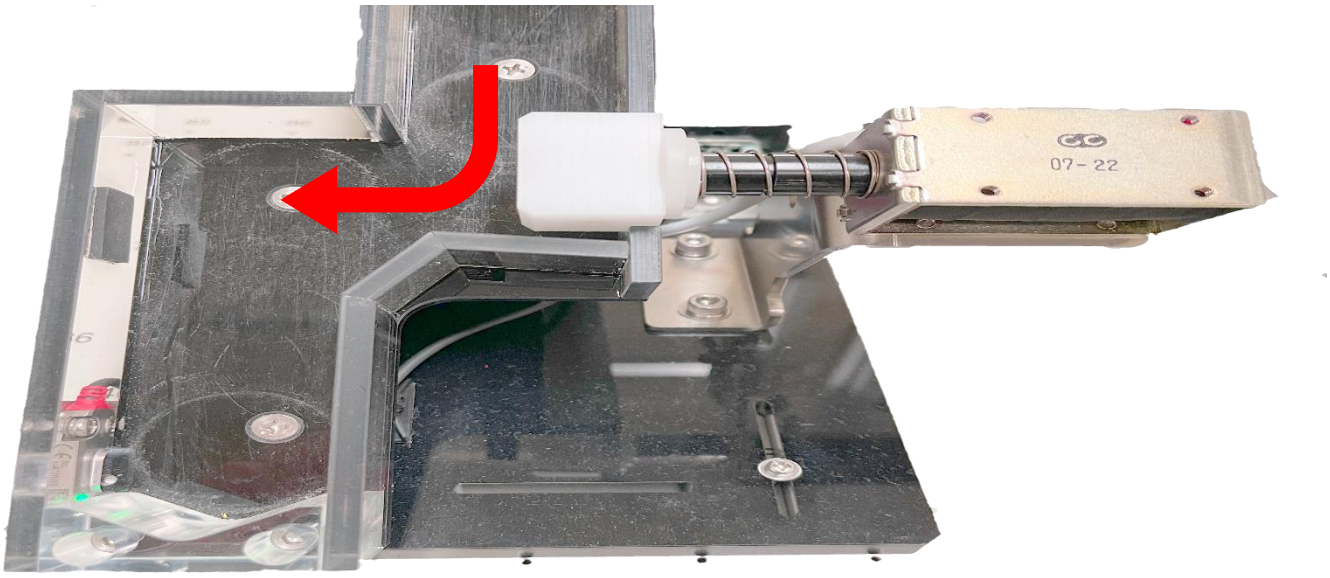
のように表示され，「ACTIVE」と表示されれば接続成功です．もし，表示されなかった場合は，LAN ケーブルがポートの奥まで正常に接続されているかを確認します．他に，コントローラとティーチペンダントの非常停止ボタンが解除，もしくは，ティーチモードがリピートモードになっているかを確認します．

今回起動したマスタノードを次の解説で扱って行く為，この状態をキープしていきます．

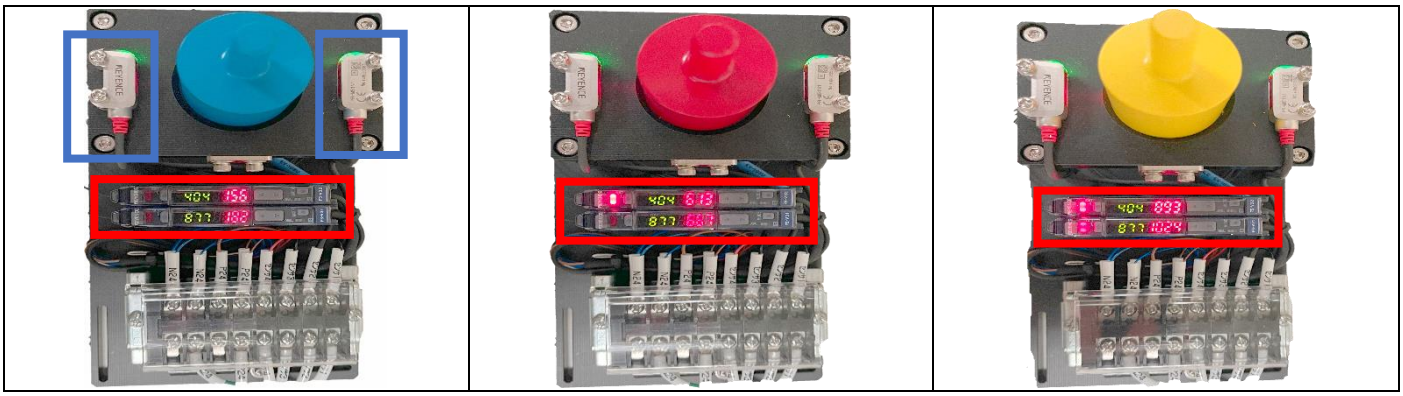
4.3.　周辺装置とワークの紹介

次に環境内にある周辺装置の紹介をしていきます．環境内には，在荷センサー，カラーセンサー，カラー台在荷センサーの 3 つのタイプのセンサーと供給ユニット(ソレノイド)とハンドの装置を紹介していきます．

上記のイメージは 4 つの色に分かれたワークで，これらを収納レーンや収納ボックスに移動，または在荷センサー，カラーセンサーで識別の状態を取得するのに扱っていきます．下記のイメージは収納レーンで供給ユニットと在荷センサーの装置が取り付けられています．収納されたワークを取り出すには，供給ユニットを起動させることでワークが在荷センサーに移動され，ワークの存在を確認することができます．



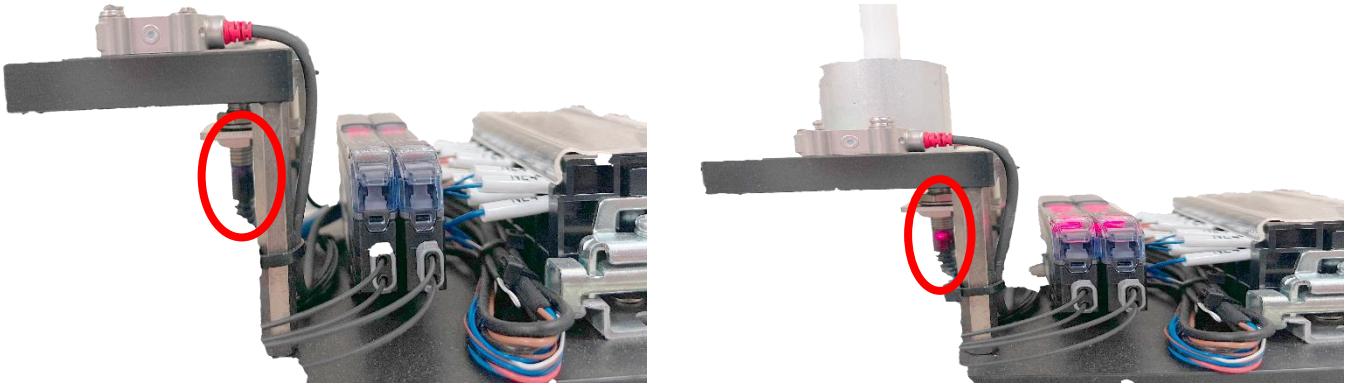
下記のイメージはカラー台在荷センサーとカラーセンサーの３つの色を判別している状態のイメージです.



青い枠のセンサーは在荷センサーでワークが設置されると緑の光が点灯します．そして，赤い枠で示しているのは，３つの色の判別状態です．カラーセンサーは色を判別しているわけではなく，ワークの表面の反射率で判別しています．それぞれの色を判別していくには，下記の表の状態で確認できます.

在荷センサー	色判別１	色判別２	金属検知	判定
IN9	IN10	IN11	IN12	
OFF	OFF	OFF	OFF	ワーク無し
ON	OFF	OFF	OFF	青色ワーク
ON	ON	OFF	OFF	赤色ワーク
ON	ON	ON	OFF	黄色ワーク
ON	ON	ON	ON	アルミワーク

表の項目の在荷センサーはカラー台在荷センサーで，それぞれの反射率の状態示す色判別１と２が上記のイメージとなっています．また，金属検知のセンサーはカラー台の下に設置されており，下記の赤い枠が点灯している状態でアルミのワークを認識することができます.



上記の表の IN9~12 のセンサーの状態を取得することによって，ワークの有無や色の識別を行うことができます．後にこの表を用いて，プログラムの条件分岐で扱っていきます．最後に各色のワークを収納するボックスがあり，色ごとに収納することができます.

4.4. 周辺装置状態取得

環境内にある周辺装置の紹介解説をしてきました．次に周辺装置の状態の取得方法について解説していきます．状態の取得を行っていく為に対話的なプログラムが行える JupyterLab を起動していきます．起動しているマスタノードから(Ctrl+b 押した後に C で) tmux の新規タブを開き，

$ jupyter-lab --allow-root --NotebookApp.token='' --notebook-dir=src/khiRobotPack/JupyterDir

のコマンドを入力し，JupyterLab のブラウザを開きます．

ファイル一覧から，GetSignalDriver.ipynb をクリックして開きます．

GetSignalDriver.ipynb を開きましたら，上から順に「Shift + Enter」で各行を実行することで，行で記述されたプログラムの結果を取得することができます．初期の行は，khi_robot のパッケージにある関数を代用しています．この関数を使用していくことで，シグナルといった周辺装置の入出力の状態を取得することができます．また今回の環境で配線されている周辺装置のシグナル値は，下記の表となっています．

IN	ロボット IN	OUT	ロボット OUT	ユニット
1 (1001)	PB1	1	SL1	操作パネル
2 (1002)	PB2	2	SL2	
3 (1003)	PB3	3	SL3	
4 (1004)	PB4	4	SL4	
5 (1005)	在荷センサー	5		ワーク供給機
6 (1006)		6		
7 (1007)		7		
8 (1008)		8		
9 (1009)	センサー(在荷センサー)	9		
10 (1010)	センサー(色判別１)	10		
11 (1011)	センサー(色判別２)	11		
12 (1012)	センサー(金属センサー)	12	ソレノイド	ワーク供給機
13 (1013)		13		
14 (1014)		14		
15 (1015)		15		
16 (1016)		16		

cmdhandler_client 関数の使い方や内容としては，第一引数に「driver」もしくは「as」の操作タイプと，第二引数にはシグナルの状態取得と状態セットを指定することで，ロボットのハンド開閉やワーク供給ユニットのオン・オフそして，センサーの状態を取得することができます．

5. 教師データの取得と設定方法

6. RS007N 実機ロボットでワーク整頓作業

付録 A　新規パッケージの作成方法

付録 B 独自プログラムの関数解説

Reference