RZ/V2H Robotic Development Kit User Manual

Release 0.1

Renesas Electronics Corporation

Nov 05, 2025

Contents

1 Getting Started

1.1 Overview

WS125 Robotic Development Kit is a solution with Renesas new generation RZ/V2H MPU for AI application, which has AI inference processing performance of up to 80TOPS with multi-core CPU to run multiple OS simultaneously for high performance AI image processing.

It is also equipped with many interfaces that make it suitable for development and integration into a variety of robotic applications.

1.1.1 Software Environment

Category	Description
OS Support	Yocto 5.1 (Styhead) and **Ubuntu 24.04 (available in headless)**.
ROS 2 Distribution	**ROS 2 Jazzy (installed on the target system).**

1.1.2 Hardware Environment

Items	Description

RZ/V2H	• **CPU**
	  – 4 × Arm Cortex-A55 (1.8GHz)
	  – 2 × Arm Cortex-R8 (800MHz)
	  – 1 × Arm Cortex-M33 (200MHz)
	• **DRP**
	  – Vision/Dynamically Reconfigurable Processor
	• **DRP-AI**
	  – Hardware AI Accelerator (8 dense TOPS, 80 sparse TOPS)
	• **Package**
	  – R9A09G057H44GBG: 1368-pin FCBGA

Memory	LPDDR4 1600MHz (8GB) × 2
SD Card	64GB SanDisk
QSPI Flash ROM	64MB
Interfaces	• DC Jack (12–24V / 2A) • JTAG (10-pin) • MIPI CSI-2 4-Lane ×2 (22-pin / 0.5mm) • HDMI • USB3.2 Type-A ×2 • USB Micro-B (SCIF) • 10/100/1000 Base-T RJ45 • Micro SD • PCIe 3.0 Root Complex (16-pin / 0.5mm) • CAN-FD ×2 • 40-pin RasPi GPIO Header

For more details about RZ/V2H RDK's specification, visit the WS125 Robotic Development Kit Hardware Manual.

RZ/V2H RDK Board Image View:

The following image shows the top/bottom view of the RZ/V2H Robotics Development Kit (RDK) board, highlighting its main connectors and interfaces.

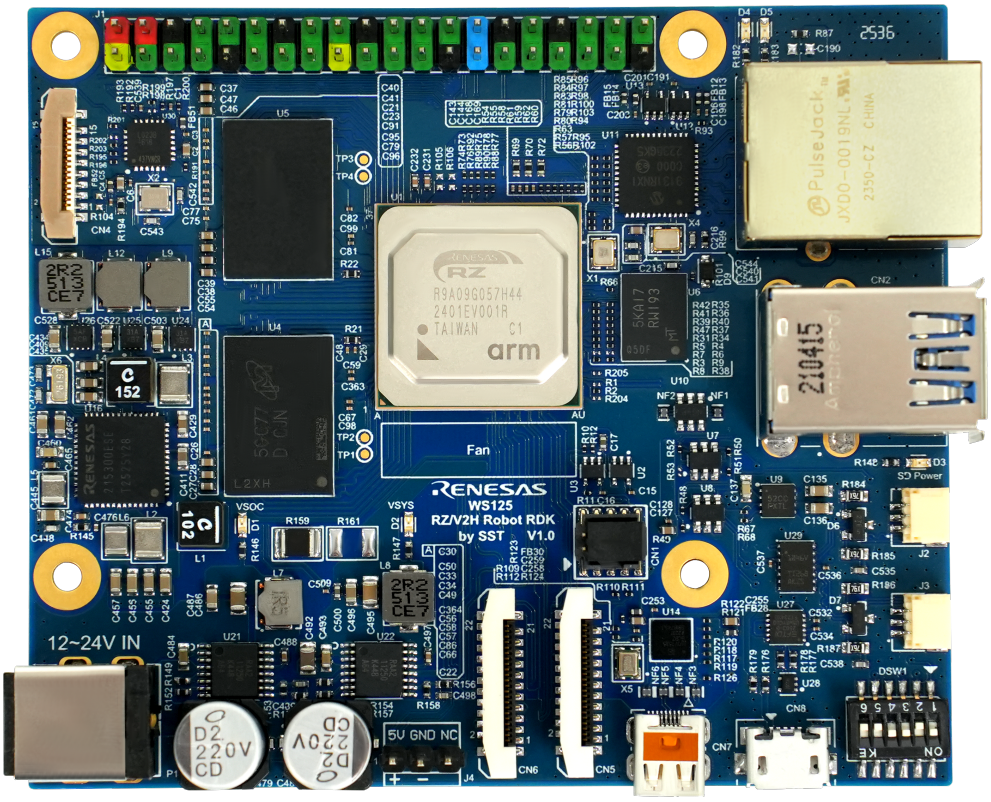

Fig. 1: RZ/V2H RDK Board Top View

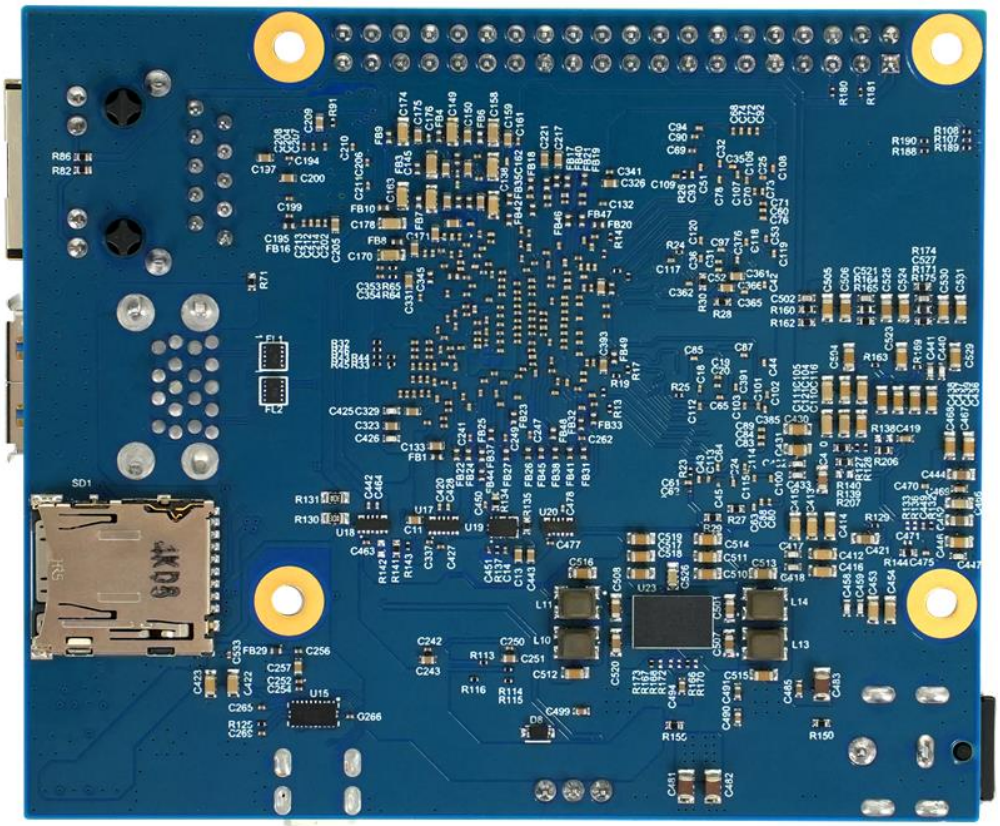

Fig. 2: RZ/V2H RDK Board Bottom View

1.2 Quick setup guide

This quick start guide focuses on booting the board using a **microSD card**, which is the most straightforward method.

Other advanced boot methods, such as **xSPI flash**, are also supported.

The **TFTP + NFS boot** method is supported as well but is not covered in detail here.

1.2.1 Preparing the SD Card

To boot the RZ/V2H RDK board using a microSD card, you must first flash a bootable Linux image onto it.

Requirements

- **Balena Etcher:** GUI-based tool to flash image
- **microSD card:** at least 16 GB recommended
- **Provided bootable Linux images:**

File name	Target OS	Host platform support
renesas-core-image-weston.wic.gz	Yocto Linux based Weston Image	Windows / macOS / Linux
ubuntu-core-image.wic.gz	Ubuntu 24.04 headless	Windows / macOS / Linux

Flash using Balena Etcher

Balena Etcher is a user-friendly GUI tool to flash OS images to SD cards and USB drives. It provides a simple and safe method.

1. **Install Balena Etcher**

   Download and install the software from the Balena Etcher Official Website.

2. **Flashing the Image**

   - Once Etcher is open:

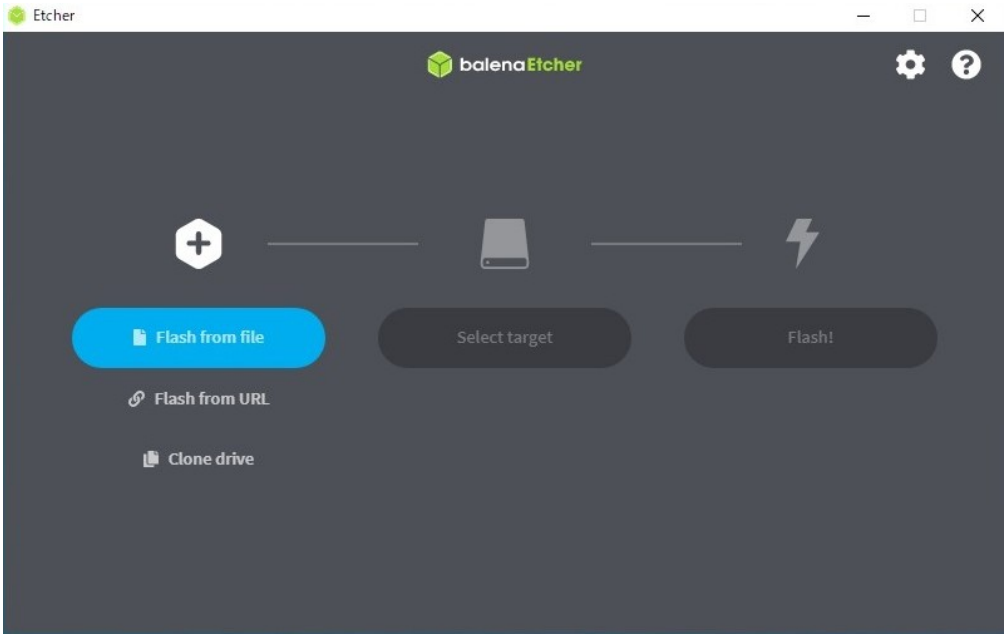

Fig. 3: Balena Etcher Application

   - **Select Image:** Click “Flash from file” and choose your image file (e.g., ubuntu-core-image.wic.gz)
   - **Select Target:** Insert your SD card into the host machine and choose the correct device.

> **Note**
>
> Please confirm the SD card device name carefully. Double-check to avoid overwriting your main disk.

   - **Flashing:** Click “Flash” to begin. Etcher will:
     - Write the image
     - Validate the image
     - Automatically unmount the SD card
   - **Finish:** Remove the SD card safely after Etcher reports successful completion.

1.2.2 Boot Mode Configuration (DIP Switch)

Before powering up the RZ/V2H RDK, make sure the board's boot mode is configured correctly using the DIP switches.

DSW1	RZ/V2H Pin	Default Setting	Operation
1	BTSEL (BOOSTSELCPU)	ON = High: 1	Select the coldboot CPU: • High: **CA55** (*default*) • Low: **CM33**
2, 3	BOOTPLLCA_1 BOOTPLLCA_0	OFF = High: 1 ON = High: 1	Input the CA55 frequency at CA55 coldboot. **BOOT_PLLCA[1:0]:** • Low:Low → 1.1 GHz • Low:High → 1.5 GHz (0.9 V) • High:Low → 1.6 GHz (0.9 V) • High:High → 1.7 GHz (0.9 V) (*default*)
4 5	MD_BOOT1 MD_BOOT0	ON = Low: 0 OFF = Low: 0	Input boot mode select signal. **MD_BOOT[1:0]:** • Low:Low → SD (*default*) • Low:High → eMMC • High:Low → xSPI • High:High → SCIF download
6	MD_BOOT3	OFF = Low: 0	Select JTAG debug mode: • Low: normal mode (*default*) • High: JTAG

Attention

Always power off the board before changing boot switches.

1.2.3 Boot Mode Support

The board supports multiple boot options, including:

Boot Source	Description	DSW1 Setting
microSD	Boot from SD card	SD mode
xSPI	Boot from xSPI flash	xSPI mode

Tip

The serial port is powered by the **board's power supply**, not by the **USB port** from the PC. Early boot messages might not appear automatically in the terminal (including U-Boot console and SCIF terminal). To view them, manually reset the board by connecting **JTAG QRESN (PIN10)** to **GND**, as shown below.

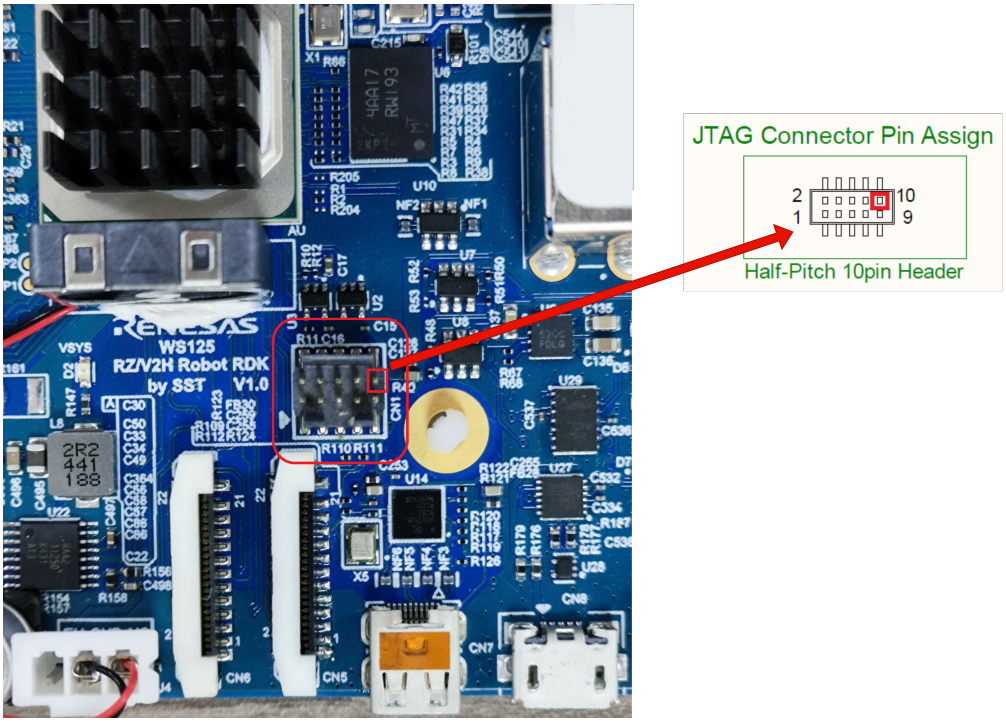

Fig. 4: JTAG Reset Pin Example

Option 1: SD Card Boot Mode

For **SD card boot mode**, the IPLs are already written to the SD card when flashing the image using Balena Etcher.

On the RZ/V2H RDK board, configure the **DSW1** switches as shown below:

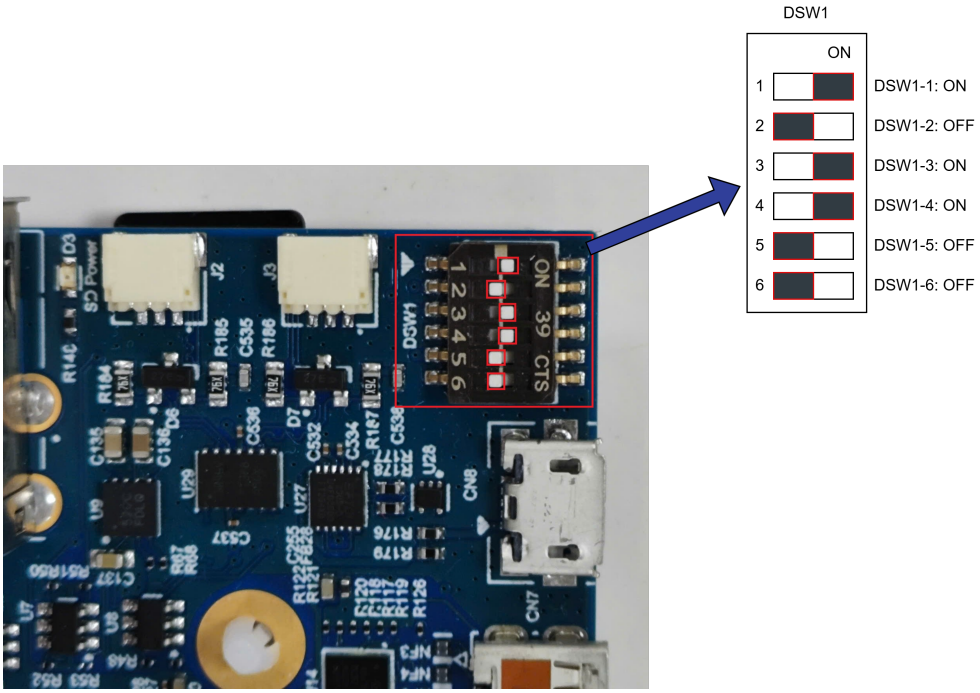

Fig. 5: DSW1 SD Card Boot Mode

After that, insert the SD card and connect the power supply (**Max 24V/5A**) to the board.

Open a terminal emulator (e.g., **Tera Term**) and connect to the **COM** port.

The COM port settings are the same as described in **Step 3** of *Write bootloaders to board*.

The board will start the boot process.

Tip

If there is no output from the terminal, do *the JTAG reset tip* first, then reset the U-Boot environment variables:

```
env default -a
saveenv
boot
```

If you intend to use **SD card boot mode only**, proceed to *First Time Boot Setup* to complete the setup.

Option 2: xSPI Boot Mode

Board Setup Procedure

Follow the instructions below to set up the board.

1. **Install Terminal Emulator**

Note

If already installed, skip this step.

- **Terminal Emulator:** Tera Term
- **Operating Environment:** Windows

2. **Install the Serial Port Driver**

> **Note**
>
> If already installed, skip this step.

- The serial communication between the Windows PC and **RZ/V2H RDK** requires: FTDI Virtual COM Port (VCP) driver

  Download and install the Windows version (*.exe*).

3. **Write Bootloaders to the Board**

Copy the bootloaders file to your Windows PC.

File Name	Description
Flash_Writer_SCIF_RZV2H_DEV_INTERNAL_MEMORY.mot	Flash writer for RZ/V2H (used in SCIF download mode)
bl2_bp_spi-rzv2h-rdk.srec	Boot loader stage 2 binary
fip-rzv2h-rdk.srec	Firmware Image Package for RZ/V2H

- Connect the **Windows PC** and **Board** using a **Serial-to-MicroUSB** cable.
- Change the **DSW1** setting to **Boot Mode 3 (SCIF download)**.

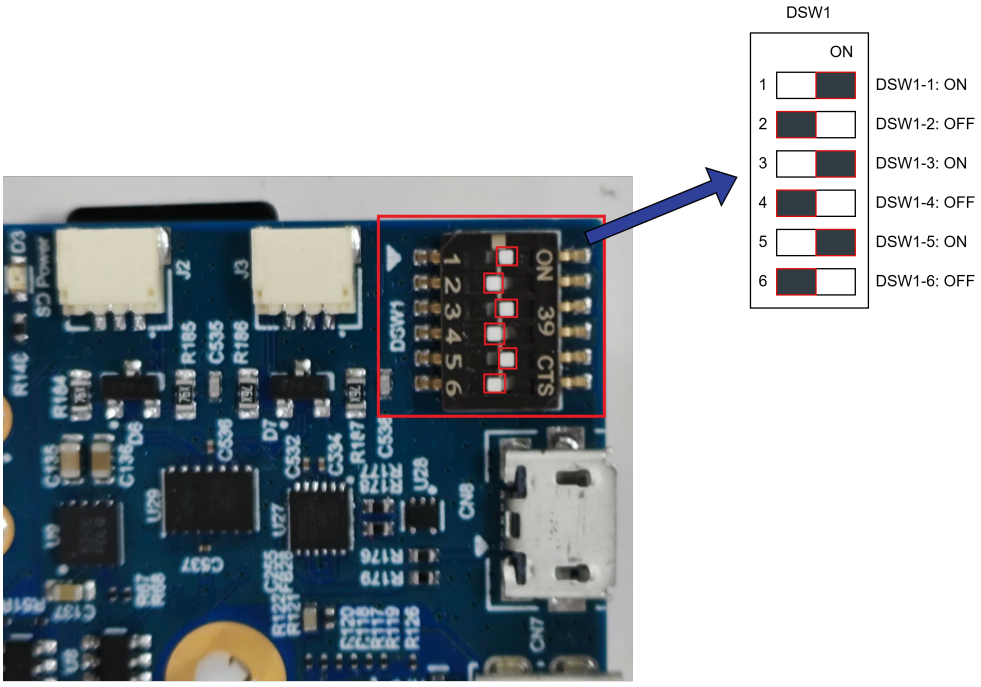

Fig. 6: DSW1 SCIF Download Mode

- Connect the power cable (**Max 24V/5A**).

- Open **Tera Term** and configure:

  Setup → Terminal:

Item	Value
New-line	Receive: Auto / Transmit: CR

  Setup → Serial Port:

Item	Value
Baud rate	115200
Data	8-bit
Parity	None
Stop	1-bit
Flow control	None
Transmit delay	0 msec/char

- Send files using “File → Send file…” and follow on-screen messages.

  (Keep the original command sequences as-is for flashing.)

4. **Setup U-Boot Configuration**
   1. Insert the microSD card to the board.
   2. Change DSW1 to **Boot mode 2 (xSPI boot)**:

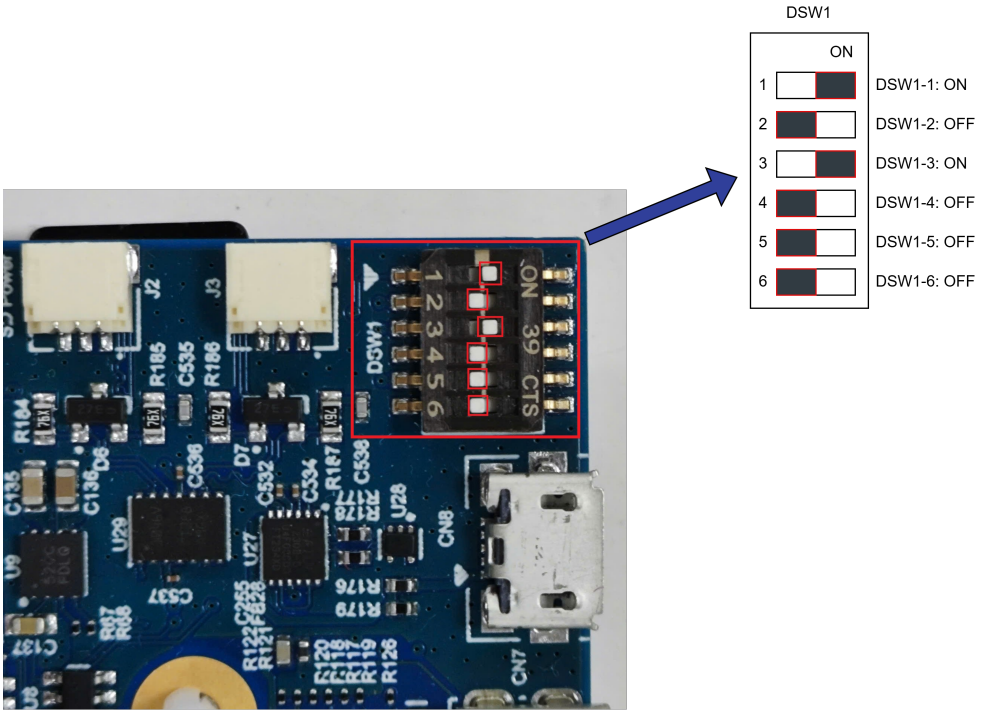

Fig. 7: DSW1 xSPI Boot Mode

3. Connect via **USB Serial to MicroUSB** cable.
4. Power on the board.
5. Open the terminal emulator and connect to the **COM** port (same configuration as before).
6. The board will boot.

> **Tip**
>
> If there is no output from the terminal, do *the JTAG reset tip* first, then reset the U-Boot environment variables:

```
env default -a
saveenv
boot
```

1.2.4 First Time Boot Setup

The default user credentials for the provided Ubuntu images are as follows:

Table 1: Default Login Information

Image Type	Username	Password	Root Password
Ubuntu 24.04 (Headless)	rz	*(none)*	*(none)*

After powering on the board **for the first time**, connect to the serial console and check the boot log to verify that Ubuntu boots successfully.

> **Note**
>
> This operation is required **only once**, immediately after flashing the root filesystem and booting the board for the first time.

Connect an Ethernet cable to the board and run:

```
# Check network
ping 8.8.8.8 -c 3
ping bing.com -c 3
```

1. Perform apt update and resize the SD card:

```
sudo apt update
sudo apt install parted
sudo parted /dev/mmcblk0
```

Inside parted terminal:

```
> print
> resizepart 2 100%
> print
> quit
```

Resize root filesystem:

```
sudo resize2fs /dev/mmcblk0p2
```

For example, this is the sample output after resizing:

```
$ sudo parted /dev/mmcblk0
sudo: unable to resolve host localhost.localdomain: Name or service
not known
GNU Parted 3.6
Using /dev/mmcblk0
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: SD SN64G (sd/mmc)
Disk /dev/mmcblk0: 63.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
1      1049kB  211MB   210MB   primary               lba
2      211MB   4855MB  4644MB  primary  ext4

(parted) resizepart 2 100%
(parted) print
Model: SD SN64G (sd/mmc)
Disk /dev/mmcblk0: 63.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
1      1049kB  211MB   210MB   primary               lba
2      211MB   63.9GB  63.7GB  primary  ext4

(parted) quit
Information: You may need to update /etc/fstab.

$ rz@localhost:~$ sudo resize2fs /dev/mmcblk0p2
sudo: unable to resolve host localhost.localdomain: Name or service
not known
resize2fs 1.47.0 (5-Feb-2023)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing
required
old_desc_blocks = 1, new_desc_blocks = 8
The filesystem on /dev/mmcblk0p2 is now 15540480 (4k) blocks long.
```

2. Setup **rosdep** for ROS2 package dependency management:

```
sudo rosdep init
rosdep update
```

3. Setup user groups for: serial port and video access, otherwise some applications may not work properly due to insufficient permissions:

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

Please log out and log back in for the group changes to take effect.

4. (Optional) Add ROS2 workspace setup to bashrc:

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

This completes the **Quick Setup Guide** for the RZ/V2H RDK board.

1.2.5 Reference

- Advanced Boot Options (xSPI): Renesas RZ/V AI SDK Developer Guide
- Balena Etcher Official Website: https://www.balena.io/etcher

1.3 RZ/V ROS2 Demos

ROS2 packages designed for computer vision applications on Renesas RZ/V MPU platforms, specifically targeting the RZ/V2H.

The packages provide AI-accelerated vision capabilities including object detection, pose estimation, and visualization tools.

1.3.1 Available Demo Applications

- *Vision Based Robotic Arm Teleoperation*: Vision-based control using MediaPipe hand tracking.
- *Vision Based Dexterous Hand*: Simultaneous mimic control of virtual and physical dexterous hands.
- *Rock Paper Scissors Game*: Legacy RPS game using hand gesture recognition.
- *Static Object Detection*: General object detection using YOLOX models.
- *Static / Camera-based Hand Landmark Estimation*: Hand detection and landmark estimation using YOLOX and MediaPipe model.

2 System Configuration

2.1 Common system configuration

This section describes how to customize and rebuild the Linux system for the RZ/V2H RDK board, including kernel, device tree, and loadable modules.

2.1.1 Overview

The RZ/V2H RDK uses a Linux kernel **version 6.10**, which is built from the **Yocto Styhead Project**. This section provides instructions on how to customize and rebuild various components of the system by using the **Yocto Extensible SDK (eSDK)** environment.

The Yocto Project eSDK provides tools that allow developers to:

- Add new applications and libraries to an image
- Modify and rebuild the source of existing components
- Test software changes directly on the target hardware

To begin working with the Extensible Software Development Kit (eSDK) in Yocto, consult the official documentation provided by the Yocto Project.

This guide offers comprehensive instructions on configuring and utilizing the eSDK effectively.

Access the official eSDK documentation by following this URL: Using the Extensible SDK.

2.1.2 Prerequisites

Before proceeding, ensure that the following prerequisites are in place:

Item	Description / Link
Docker	Must be installed on the Host PC. Refer to the Docker Official Installation Guide.
Yocto eSDK for RZ/V2H RDK	Required for extensible development and rebuilding components. Obtain the installer: `poky-glibc-x86_64-renesas-core-image-weston-cor-texa55-rz-cmn-toolchain-ext-5.1.4.sh`
Yocto SDK for RZ/V2H RDK	Standard SDK tool-chain for building applications. Obtain the installer: `poky-glibc-x86_64-renesas-core-image-weston-cor-texa55-rz-cmn-toolchain-5.1.4.sh`
RZ/V2H RDK X Compile Docker	Preconfigured cross-compilation Docker environment. Refer to the *RZ/V2H RDK X Compile Docker repository*.

2.1.3 Quick set up guide

Common docker environment setup

To streamline the development process, it is recommended to use a Docker container that has been preconfigured for cross-compiling applications as well as eSDK development for the RZ/V2H RDK board.

This section provides a quick guide to setting up the Docker environment.

Obtain all files from the Prerequisites section, and following the step below to set up the Docker environment:

1. Clone the *x_compilation_docker* repository to your Host PC.

```
renesas@builder-pc:~$ git clone https://partnergitlab.renesas.solutions/
sst1/industrial/ws078/utility/x_compilation_docker.git
```

2. Copy the **Yocto SDK** installer to the Docker build context directory. Please replace the paths below with your actual file locations.

```
renesas@builder-pc:~$ cp poky-glibc-x86_64-renesas-core-image-
weston-cortexa55-rz-cmn-toolchain-5.1.4.sh ~/x_compilation_docker/
```

> **Hint**
>
> *Why copy the Yocto SDK installer into the Docker build context directory?*
>
> This step is required to prepare the cross-compilation environment for cross-compiling the ROS2 applications targeting the RZ/V2H platform.
>
> The setup of this cross-compilation workflow will be introduced in a later section.

3. Build the Docker Container

   Navigate to the x_compilation_docker directory and build the Docker image using the following command:

```
renesas@builder-pc:~$ cd x_compilation_docker/
renesas@builder-pc:~/x_compilation_docker$ ./setup_ros2_cross_env.
sh <path_to_ros2_workspace> [name_of_docker_container]
```

- <path_to_ros2_workspace>: Path to your ROS2 workspace, mounted inside the container.
- [name_of_docker_container]: Optional container name (default: rzv2h_ros_xbuild).

After complete this step, the Docker image (name: rzv2h_ros_xbuild:latest) and container will be created.

Tip

- In the *Common docker environment setup* section, we can create a new Docker container by replace the `[name_of_docker_container]` placeholder with a custom name.
- In case you need to rebuild the Docker image (for example, after modifying the SDK), use the following command to rebuild the image:

```
renesas@builder-pc:~/x_compilation_docker$ ./setup_ros2_cross_env.sh .
[name_of_docker_container]
Docker image 'rzv2h_ros_xbuild:latest' already exists.
Do you want to create the container based on this image or create a new
Docker image?
1) Use existing image
2) Rebuild new image
Enter your choice [1 or 2]:
```

  Enter **2** and provide a tag name to rebuild the image. The script will back up the existing image using the provided tag before rebuilding.

What does this script do?

- Copying the Yocto SDK tool-chain scripts into the build directory lets the Dockerfile install the Yocto SDK inside the image.
- The setup script usually runs docker build (to produce the image) and docker run (to create/start a container) with proper mounts, environment variables and volumes.
- Inside the container the image will have the cross compilers, sysroot and a configured environment (CMake toolchain file, sourced toolchain setup) so colcon/CMake can cross-compile ROS2 for the target.

4. Enter the Docker Container

   Use the following command to enter the Docker container:

```
renesas@builder-pc:~$ docker exec -it [name_of_docker_container] /bin/
bash
```

eSDK Setup

The purpose of the Yocto eSDK (Extended Software Development Kit) is as follows:

- **Rebuild the Linux kernel and device tree** to customize or modify system behavior for the RZ/V2H platform.
- **Add or update kernel configurations** to enable additional hardware drivers or features.
- **Install missing ROS 2 packages** required for the cross-build workflow to ensure compatibility and completeness of the development environment.

Note

If you do not require any of the purposes listed above, you may **skip the eSDK setup** for now. You can always return and perform the eSDK setup later when kernel rebuilding, driver configuration, or additional ROS 2 package installation becomes necessary.

To get started, extract the eSDK and install the tool-chain.

Please replace the paths below with your actual file locations.

Tip

There is no requirement to use the Yocto eSDK within the Docker environment.

However, using Docker can simplify the setup process and ensure a consistent build environment. Since Yocto SDK/eSDK installations can be complex, Docker provides a preconfigured environment that minimizes setup time and potential configuration issues.

Additional, ensure that the **Yocto eSDK installer file** has been **copied into the container** so it can be executed from within that environment.

```
renesas@builder-pc:~$ docker cp poky-glibc-x86_64-renesas-core-image-
weston-cortexa55-rz-cmn-toolchain-ext-5.1.4.sh [name_of_docker_container]:/
home/ubuntu
```

To set up your environment:

1. Install the Yocto eSDK tool-chain.

For example, to install the tool-chain, run the following command.

```
renesas@docker-pc:~$ sh ./poky-glibc-x86_64-renesas-core-image-weston-
cortexa55-rz-cmn-toolchain-ext-5.1.4.sh
```

Note

You **cannot install the eSDK as the root user** because BitBake does not run with root privileges. Attempting to install the extensible SDK as root will cause the installation to fail or behave unpredictably.

If the installation is successful, the following messages will appear in the terminal output.

```
renesas@docker-pc:~$ sh ./poky-glibc-x86_64-renesas-core-image-weston-
cortexa55-rz-cmn-toolchain-ext-5.1.4.sh
Poky (Yocto Project Reference Distro) Extensible SDK installer version 5.1.4
==========================================================================
Enter target directory for SDK (default: ~/poky_sdk):
You are about to install the SDK to "/home/renesas/poky_sdk/5.1.4". Proceed
[Y/n]? Y
Extracting SDK.............done
Setting it up...
Extracting buildtools...
Preparing build system...
Parsing recipes: 100% |#########################################| Time:
0:00:52
Initialising tasks: 100% |######################################| Time:
```

(continues on next page)

(continued from previous page)

```
0:00:00
Checking sstate mirror object availability: 100% |###############| Time:
0:00:00
Loading cache: 100% |#########################################| Time:
0:00:00
Initialising tasks: 100% |#####################################| Time:
0:00:00
done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source
the
environment setup script e.g.
$ . /home/renesas/poky_sdk/5.1.4/environment-setup-cortexa55-poky-linux
```

2. Set up cross-compile environment.

The following command assumes that you installed the SDK in *~/poky_sdk/5.1.4*

```
renesas@docker-pc:~$ source ~/poky_sdk/5.1.4/environment-setup-cortexa55-poky-
linux
SDK environment now set up; additionally you may now run devtool to perform
development tasks.
Run devtool --help for further details.
```

Note

The user needs to run the above command once for each shell session.

2.1.4 Using devtool in the Yocto eSDK

The devtool utility is part of the Yocto Project's Extensible SDK (eSDK). It provides an isolated workspace for modifying, testing, and maintaining recipes without altering upstream metadata.

This section focuses on the core commands used in day-to-day development, especially for Linux kernel, device trees, and driver modifications on Renesas RZ Common System.

1. devtool modify

   The devtool modify command checks out the source code for a recipe into a local workspace, allowing changes without touching upstream layers. This is usually the first step in customizing the kernel, device trees, or drivers.

   Example: To modify the Linux kernel recipe:

   ```
   renesas@docker-pc:~$ devtool modify linux-yocto
   ```

   This command sets up a workspace where you can make changes to the Linux kernel source code.

2. devtool build

   After making changes to the source code, use this command to build the modified recipe.

   Example: To build the modified Linux kernel:

   ```
   renesas@docker-pc:~$ devtool build linux-yocto
   ```

   This command compiles the changes and prepares them for deployment.

3. devtool reset

   If you want to discard your changes and revert to the original recipe, use this command.

   Example: To reset the Linux kernel recipe:

   ```
   renesas@docker-pc:~$ devtool reset linux-yocto
   ```

   This command removes your modifications and restores the original source code.

4. devtool build-image

   The devtool build-image command builds a complete target image that includes all recipes currently under modification.

   This is useful to verify integration of changes into a full root filesystem, not just individual components.

   Example: To build a new renesas-core-image-weston:

   ```
   renesas@docker-pc:~$ devtool build-image renesas-core-image-weston
   ```

   This command generates an updated image that can be deployed to the target hardware.

Known Issue

When working with the **Yocto eSDK**, you might encounter the following warning:

```
WARNING: You are using a local hash equivalence server but have configured an
sstate mirror.
This will likely mean no sstate will match from the mirror.
You may wish to disable the hash equivalence use (BB_HASHSERVE), or use a hash
equivalence server alongside the sstate mirror.

The ros2-control:do_package_qa sig is computed to be
ea8f7e910d566912b932cbe602d93b93502064e293d1f4f1f569a67ee49f1c72,but the sig
is locked to fd89fc1eb9961fd4ccddf16ea2ca1b73b5480ce1670ebb07a8075603bb645bc8
in SIGGEN_LOCKEDSIGS_t-cortexa55
```

> **Note**
>
> This warning can be **safely ignored**. It does not affect the build process or output artifacts when using the Yocto eSDK.

2.1.5 Custom Linux Kernel and Device Tree

This section describes how to customize and rebuild the Linux kernel or device tree blob for the RZ/V2H RDK board using the Yocto eSDK and devtool.

> **Note**
>
> Before proceeding, ensure that you have set up the eSDK environment as described in the *eSDK Setup* section above.

1. modify the Linux kernel recipe:

   Setup the eSDK environment in the current terminal session:

```
renesas@docker-pc:~$ source ~/poky_sdk/environment-setup-cortexa55-poky-
linux
```

Use the devtool modify command to check out the linux-yocto recipe for modification:

```
renesas@docker-pc:~$ devtool modify linux-yocto
```

When executed, this:

- Creates a workspace copy of the kernel source under: *~/poky_sdk/workspace/sources/linux-yocto/*
- Generates a .bbappend for linux-yocto in the workspace's recipe area.
- Prepares the environment for kernel modifications.

2. Applying Patches for linux-yocto

Unlike most recipes, linux-yocto in this BSP is implemented as an out-of-tree kernel.

- Kernel modifications are stored as patches inside workspace/sources/linux-yocto/.kernel-meta/
- The kernel default configuration (renesas_defconfig) is also managed out-of-tree.
- To ensure the workspace matches the recipe, patches must be applied after running devtool modify.

Procedure, applying patches to the kernel linux-yocto source

```
renesas@docker-pc:~$ cd ~/poky_sdk/workspace/sources/linux-yocto/.kernel-meta
renesas@docker-pc:~/poky_sdk/workspace/sources/linux-yocto$ git am $(cat
patch.queue)
```

This applies the patch series defined in .kernel-meta/patches/ to the kernel workspace.

After applying the patches, developers may perform the following steps:

- Provide kernel configuration fragments (`.cfg` files) to adjust features or enable additional built-in kernel drivers.
- Modify the kernel source code as needed for custom functionality.
- Continue with the `devtool build linux-yocto` command to compile the kernel with the applied modifications.

3. Adding Kernel Configuration Fragments

> **Tip**
>
> If you're unsure whether the config has been added to the kernel configuration,
>
> you can use *zcat /proc/config.gz | grep <CONFIG_NAME>* in the target machine to check whether it is enabled.

There are two possible methods to add kernel configuration for linux-yocto:

- Edit the out-of-tree defconfig directly: *~/poky_sdk/layers/meta-renesas/recipes-kernel/linux/rz-cmn/common/kernel-common.cf*
- **Adding a configuration fragment (.cfg) file**
  - Create the append structure

```
renesas@docker-pc:~$ mkdir -p ~/poky_sdk/workspace/appends/linux-
yocto/
```

- Create a configuration fragment file, e.g., *myconfig.cfg*, inside the *~/poky_sdk/workspace/appends/linux-yocto/* directory. For example:

```
CONFIG_USB_SERIAL=y
CONFIG_USB_SERIAL_FTDI_SIO=y
```

> **Note**
>
> Please include all dependency configs as well.

- Edit the *~/poky_sdk/workspace/appends/linux-yocto_6.10.bbappend* file to include the new configuration fragment at the end of file:

```
renesas@docker-pc:~$ vi ~/poky_sdk/workspace/appends/linux-yocto_6.10.
bbappend
```

- Add the following line to the bbappend file:

```
SRC_URI:append = " file://myconfig.cfg"
```

4. Modify the DTS file:

```
renesas@docker-pc:~$ cd ~/poky_sdk/workspace/sources/linux-yocto/
renesas@docker-pc:~$ vi arch/arm64/boot/dts/renesas/r9a09g057h4-rdk-ver1.
dts
```

Make the necessary changes to the Device Tree Source file (`.dts`) according to your requirements.

5. Build the Modified Kernel

After making changes in the workspace, use devtool build to compile the recipe.

```
renesas@docker-pc:~$ devtool build -c linux-yocto
renesas@docker-pc:~$ export DISTRO=ubuntu-tiny
renesas@docker-pc:~$ devtool build linux-yocto
```

> **Attention**
>
> Before building, always **clean the previous build artifacts** to avoid conflicts.
>
> ```
> renesas@docker-pc:~$ devtool build -c linux-yocto
> ```
>
> Always set the `DISTRO` variable to `ubuntu-tiny` to ensure compatibility with the Ubuntu-based root filesystem.
>
> Otherwise, the generated artifacts may **not be compatible** with the Ubuntu image.

6. Collect the Built Kernel Artifacts

Once the build is complete, collect the built kernel artifacts for deployment to the target hardware.

When building `linux-yocto`, the generated artifacts can be collected from the following locations: *~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image*

Please copy those files to the appropriate boot media (e.g., SD card) as per your deployment requirements.

Tip

1. Make sure to copy the updated kernel image and device tree blob to the correct locations in the root file system on the SD card.

   For example:

   Table 1: File Mapping Between eSDK Build Output and Target RootFS

eSDK Build Output Path	Target RootFS Location
~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/boot/Image-6.10.14-yocto-standard	/boot/Image-6.10.14-yocto-standard
~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/boot/dtb/renesas/r9a09g057h4-rdk-ver1.dtb	/boot/dtb/renesas/r9a09g057h4-rdk-ver1.dtb
~/poky_sdk/workspace/sources/linux-yocto/oe-workdir/image/usr/lib/modules	/usr/lib/modules

2. On the target device, run the following command to update the module dependencies after deployment:

```
$ sudo depmod -a
```

2.2 Ubuntu System with RZ/V2H RDK

This section provides usage information about the interfaces available on the RZ/V2H RDK when running the Ubuntu system.

For more details about specification of each interface, refer to the RZ/V2H Group User's Manual: Hardware.

2.2.1 Overview

The RZ/V2H RDK supports multiple peripheral interfaces that allow users to connect and control external devices for various robotic and industrial applications. These interfaces include:

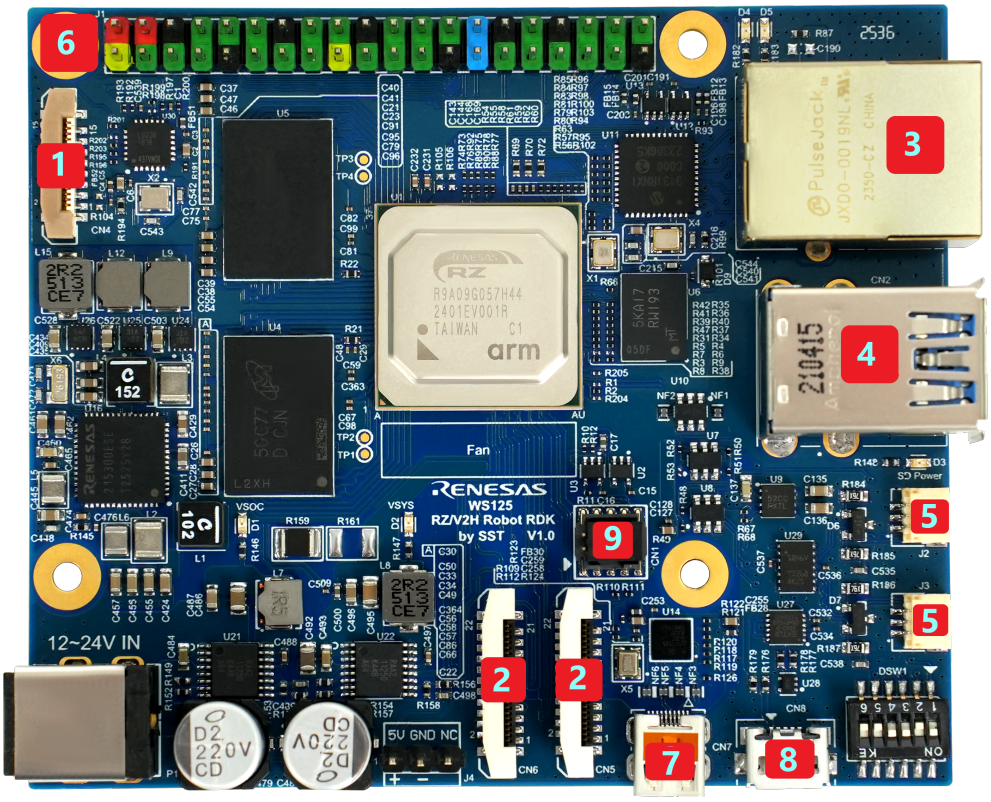

Fig. 1: RZ/V2H Robotic Development Kit Interfaces

2.2.2 Main Interfaces

The main interfaces available on the RZ/V2H Robotic Development Kit are listed below.

Table 2: High-Speed Interfaces

No	Interface Name	Description
1	PCIe 3.0 16-pin connector	High-speed peripheral expansion interface.
2	MIPI-CSI 22-pin connector ×2	Dual camera input interface for image sensors.
3	1000M RJ45	Gigabit Ethernet LAN port for network connectivity.
4	USB 3.0 Type A ×2	USB host ports for external devices such as mouse, keyboard, or USB camera.

Table 3: Communication Interfaces

No	Interface Name	Description
5	CAN-FD ×2	Controller Area Network Flexible Data-Rate communication ports.
6	RasPi GPIO 40-pin Header (I2C, SPI, UART, GPIO, PCM)	General purpose I/O interface compatible with Raspberry Pi pin layout.

Table 4: Other Interfaces

No	Interface Name	Description
7	Mini-HDMI	Video output interface for external display.
8	USB-UART	Serial console interface for debugging.
9	JTAG 10-pin	JTAG interface for debugging and programming.

Each subsection provides details on how to identify, configure, and access these interfaces within the Ubuntu environment.

High-Speed Interface

The RZ/V2H Robotic Development Kit is equipped with several high-speed interfaces that enable users to connect a variety of peripherals and expansion modules.

This sections describes the High-Speed Interface unit of this Kit.

1. PCIe 3.0 16-pin connector

The PCIe 3.0 interface on the RZ/V2H RDK allows for high-speed data transfer and connectivity with compatible PCIe devices.

For example, you can connect a PCIe NVMe SSD to enhance storage performance. Following are the steps to set up and use a PCIe NVMe SSD with the RZ/V2H RDK.

- Hardware Requirements (recommended):
  - PCIe TO M.2 Board (D).
  - M.2 NVMe SSD.
- Hardware Setup:
  1. Power off the RZ/V2H RDK.
  2. Connect the PCIe TO M.2 Board to the PCIe 3.0 16-pin connector on the RZ/V2H RDK.
  3. Insert the M.2 NVMe SSD into the PCIe TO M.2 Board.
  4. Connect 5V power and GND (from GPIO 40 Pins) to the PCIe TO M.2 Board.
  5. Power on the RZ/V2H RDK.

Important

- Ensure that the PCIe TO M.2 Board is properly powered, as the RZ/V2H RDK does not supply power to PCIe devices.
- Make sure to handle the M.2 NVMe SSD with care to avoid damage from static electricity.
- Make sure you connect the PCIe TO M.2 Board to the correct PCIe 3.0 16-pin connector on the RZ/V2H RDK.

Usage example with pciutils:

First, install the pciutils package if it is not already installed:

```
$ sudo apt install pciutils
```

To list all PCIe devices connected to the system, use the following command:

```
$ lspci
```

Example output:

```
root@localhost:~# lspci
00:00.0 PCI bridge: Renesas Technology Corp. Device 003b
01:00.0 Non-Volatile memory controller: Samsung Electronics Co Ltd NVMe SSD
Controller 980 (DRAM-less)
root@localhost:~#
```

To check the NVMe SSD is recognized by the system, use the following command:

```
$ lsblk
```

Example output:

```
root@localhost:~# lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
mtdblock0    31:0    0 116.5K  1 disk
mtdblock1    31:1    0   1.8M  1 disk
mtdblock2    31:2    0   128K  1 disk
mtdblock3    31:3    0    14M  0 disk
mmcblk0     179:0    0  29.7G  0 disk
|-mmcblk0p1 179:1    0   100M  0 part
`-mmcblk0p2 179:2    0   2.4G  0 part /
nvme0n1     259:0    0 465.8G  0 disk
```

Mount the NVMe SSD:

```
$ sudo mkdir /mnt/nvme
$ sudo mount /dev/nvme0n1 /mnt/nvme
```

Unmount the NVMe SSD:

```
$ sudo umount /mnt/nvme
$ sudo rmdir /mnt/nvme
```

2. MIPI-CSI 22-pin connector ×2

The RZ/V2H RDK features dual MIPI-CSI connectors that support camera input for applications requiring image capture and processing.

See also

For information about available partner Camera module list on RZ/V2H, see [RZ/V2H] AVAILABLE PARTNER CAMERA MODULE LIST.

The default RZ/V2H RDK device tree supports the OV5645 camera module connected to the MIPI-CSI interface.

Hint

It is required to change the dts file to use the MIPI-CSI interface with other camera module. Refer to: *Modify the DTS file* section in the Build Kernel chapter for more details about customize the dts file.

Set up the MIPI-CSI interface

Before using the MIPI-CSI interface, we have to configure the property of the camera first.

For example, to use the OV5645 camera module, create and run the v4l2_init.sh script in the terminal:

This script detects the connected camera module and sets the desired resolution.

For other camera modules, please modify the script accordingly.

Usage example with v4l2-ctl

First, install the v4l-utils package if it is not already installed:

```
$ sudo apt install v4l-utils
```

List all connected cameras:

```
$ v4l2-ctl --list-devices
```

List all supported formats for selected camera /dev/video0:

```
$ v4l2-ctl -d /dev/video0 --list-formats-ext
```

To capture an image from the camera using **Renesas Core Image Weston**, use the following command:

```
$ gst-launch-1.0 v4l2src device=/dev/video0 ! videoconvert ! waylandsink
```

3. 1000M RJ45 - Gigabit Ethernet Port

The Gigabit Ethernet (1000M RJ45) port on the RZ/V2H RDK provides high-speed network connectivity for data communication and internet access.

Connect the network cable to (3) before using the Ethernet interface.

The Ubuntu installer configures the system to obtain its network settings via DHCP.

After connecting the Ethernet cable, use the following command to confirm the network configuration:

To list all network interfaces and their IP addresses:

```
$ ifconfig
```

To test network connectivity reach an external server, use the ping command:

```
$ ping -c 4 bing.com
$ ping -c 4 8.8.8.8
```

Set a static IP address

In Ubuntu, the network is configured with Netplan, if you need to set a static IP address for the Ethernet interface (example: use a static IP address 192.168.0.100), follow these steps:

- Open the network configuration file with vim:

```
$ sudo vi /etc/netplan/01-netcfg.yaml
```

- Modify the file to set a static IP address. For example:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
version: 2
renderer: NetworkManager
ethernets:
    end0:
    dhcp4: no
    dhcp6: no
    addresses: [169.254.43.99/24]
    routes:
      - to: default
        via: 169.254.43.86
    nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

- Apply the changes with the following command:

```
$ sudo netplan apply
```

Set DHCP

In case you want to revert back to DHCP configuration, modify the `/etc/netplan/01-netcfg.yaml` file as follows:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    end0:
      dhcp4: yes
      dhcp6: yes
```

- Apply the changes with the following command:

```
$ sudo netplan apply
```

4. USB 3.0 Type A ×2

The RZ/V2H RDK includes two USB 3.0 Type-A ports that support high-speed data transfer for connecting various USB peripherals, such as external storage devices, cameras, and input devices.

To use these devices, simply connect them to the USB 3.0 Type-A ports.

Verify USB 3.0 functionality

To verify that the USB 3.0 ports are functioning correctly, you can use the following command to list USB devices and check their connection speed:

```
$ lsusb -t
```

Example output:

```
root@localhost:~# lsusb -t
/:  Bus 001.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
480M
/:  Bus 002.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
20000M/x2
/:  Bus 003.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
480M
/:  Bus 004.Port 001: Dev 001, Class=root_hub, Driver=xhci-renesas-hcd/1p,
20000M/x2
root@localhost:~#
```

USB-WIFI Adapter Support

The following USB-WIFI adapters have been tested and are compatible with the RZ/V2H RDK:

- Ralink Technology, Corp. MT7601U Wireless Adapter
- AC1300 Tp-Link T3U Nano

Please refer to the *USB-WIFI Adapter Support* section for more detailed instructions on driver and firmware installation for these adapters.

Usage example

- Check USB Devices

  First, connect the USB-WIFI adapter to the RZ/V2H RDK.

  Then, run the following command to list all connected USB devices:

  ```
  $ lsusb
  ```

  Example output:

  ```
  Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
  Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
  Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
  Bus 003 Device 003: ID 2357:0138 TP-Link 802.11ac NIC
  Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
  Bus 005 Device 003: ID 148f:7601 Ralink Technology, Corp. MT7601U
  Wireless Adapter
  ```

- Check Interface Name

  ```
  $ ip a | grep wl
  ```

  Example output:

```
7: wlx98ba5f1918cf: <BROADCAST,MULTICAST,DYNAMIC> mtu 1500 qdisc
noqueue state DOWN group default qlen 1000
```

- Check Device Status

```
$ nmcli dev status
```

Example output:

```
DEVICE           TYPE      STATE                   CONNECTION
end0             ethernet  connected               Wired
connection 1
lo               loopback  connected (externally)  lo
wlx98ba5f1918cf  wifi      unavailable             --
can0             can       unmanaged               --
can1             can       unmanaged               --
dummy0           dummy     unmanaged               --
sit0             iptunnel  unmanaged               --
```

- Enable the WiFi Interface

If the WiFi interface is disabled, the STATE will show as "unavailable".

We have to enable the WiFi interface first:

```
$ sudo nmcli radio wifi on
```

- Recheck Device Status

```
$ nmcli dev status
```

Example output:

```
DEVICE           TYPE      STATE                   CONNECTION
end0             ethernet  connected               Wired
connection 1
lo               loopback  connected (externally)  lo
wlx98ba5f1918cf  wifi      disconnected            --
can0             can       unmanaged               --
can1             can       unmanaged               --
dummy0           dummy     unmanaged               --
sit0             iptunnel  unmanaged               --
```

- List Available WiFi Connections

```
$ nmcli dev wifi list
```

Example output:

```
IN-USE  BSSID              SSID             MODE   CHAN  RATE
  SIGNAL  BARS  SECURITY
        C8:7F:54:E1:45:10  MY_SSID         Infra  8     270 Mbit/s
 100     ▂▄▆█  WPA2
```

- Connect to WiFi Network

```
# Replace MY_SSID and yourpassword with the actual WiFi SSID and
password.
$ sudo nmcli dev wifi connect MY_SSID password "yourpassword"
```

- Example output:

```
sudo: unable to resolve host localhost.localdomain: Name or
service not known
Device 'wlx98ba5f1918cf' successfully activated with '0fde1c53-
876d-4752-adba-8b6f758d9c51'.
```

- Test Network Connectivity

```
$ ping -I wlx98ba5f1918cf bing.com
```

Example output:

```
PING bing.com (150.171.27.10) from 192.168.19.177 wlx98ba5f1918cf:
56(84) bytes of data.
64 bytes from 150.171.27.10: icmp_seq=1 ttl=120 time=420 ms
64 bytes from 150.171.27.10: icmp_seq=2 ttl=120 time=482 ms
--- bing.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 419.837/451.166/482.495/31.329 ms
```

- Disconnect from WiFi Network

```
$ sudo nmcli con down id MY_SSID
```

Communication Interfaces

This section provides usage examples of the communication interfaces available on this kit.

5. CAN-FD ×2

The RZ/V2H RDK is equipped with two CAN-FD (Controller Area Network Flexible Data-Rate) ports that enable high-speed communication for automotive and industrial applications.

> **Tip**
>
> To use the CAN-FD interfaces, ensure that you have the required CAN connectors.
>
> We recommend using the: **Bus Cable, 1 Female Connector, 3 Contacts, 1.0 mm Pitch, 10 cm Length**.
>
> The RZ/V2H RDK is equipped with an onboard CAN transceiver (**TCAN1046VDMTRQ1**) and an integrated **120 Ω termination resistor**, eliminating the need for any external CAN transceiver circuitry.

Connect the CAN-FD ports to your CAN network using appropriate connectors and cables. CAN-H and CAN-L lines should be connected accordingly.

Usage examples:

Verify that the CAN interfaces are recognized:

```
$ ip link show | grep can
```

Example output:

```
5: can0: <NOARP,ECHO> mtu 72 qdisc noop state DOWN mode DEFAULT group default
qlen 10 link/can
6: can1: <NOARP,ECHO> mtu 72 qdisc noop state DOWN mode DEFAULT group default
qlen 10 link/can
```

Bring up the CAN interface (e.g., 500 kbps nominal, 2 Mbps data):

```
$ sudo ip link set can0 down
$ sudo ip link set can0 type can bitrate 500000 dbitrate 2000000 fd on
$ sudo ip link set can0 up
```

Check the interface status:

```
$ ip -details link show can0
```

Send and Receive CAN Messages

You can use the **can-utils** package for testing CAN-FD communication.

1. Install can-utils (if not already installed):

   ```
   $ sudo apt install can-utils
   ```

2. On one terminal, listen for incoming CAN-FD frames:

   ```
   $ candump can0
   ```

3. On another terminal, send a test frame:

   ```
   $ cansend can0 123##01122334455667788
   ```

6. RasPi GPIO 40-pin Header

The Raspberry Pi GPIO 40-pin header on the RZ/V2H RDK provides a versatile interface for connecting various peripherals and expansion boards compatible with the Raspberry Pi pin layout. This header includes multiple communication protocols such as I2C, SPI, UART, GPIO, and PCM.

Support the following communication protocols:

- I2C (Inter-Integrated Circuit)
- SPI (Serial Peripheral Interface)
- UART (Universal Asynchronous Receiver/Transmitter)
- GPIO (General Purpose Input/Output)
- PCM (Pulse Code Modulation)

Pin Out Diagram

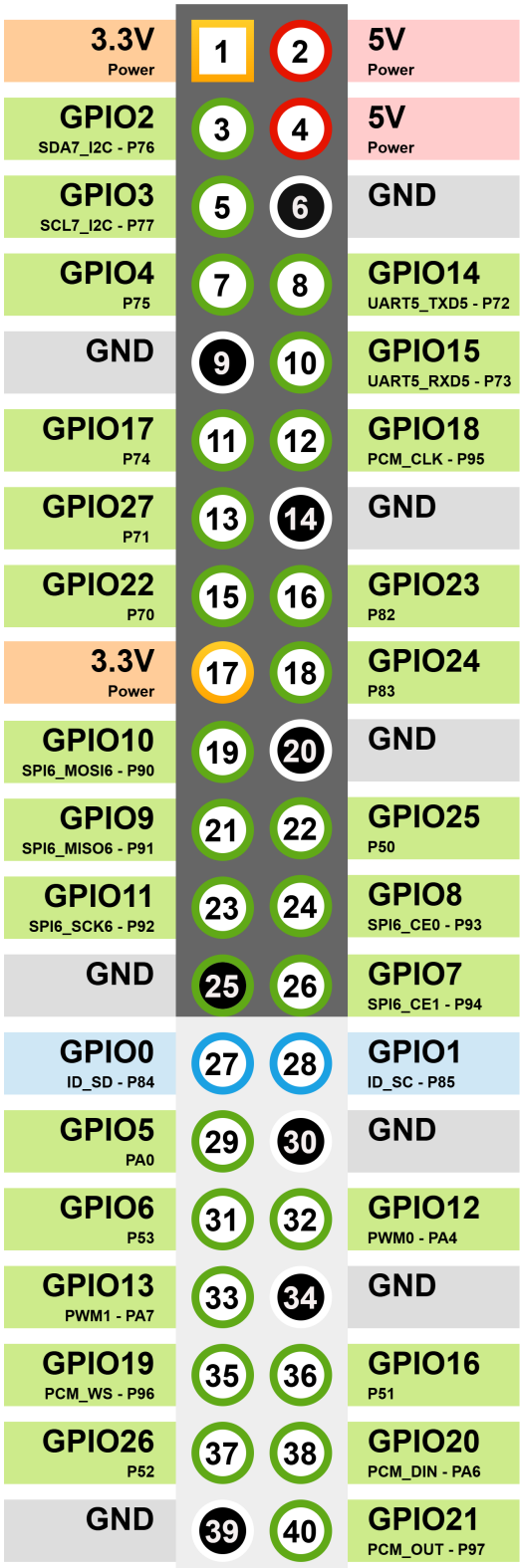

Fig. 2: RZ/V2H RDK Raspberry Pi GPIO 40-pin Header Pin Out

I2C (Inter-Integrated Circuit)

The I2C interface allows communication with multiple slave devices using just two wires: SDA (data line) and SCL (clock line).

It is commonly used for connecting sensors, displays, and other peripherals.

On the RZ/V2H RDK, the I2C pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 5: I2C7 Interface Pins

Pin Name	Function	Description
P76	SDA7	I2C7 data line - Serial Data (connected with 2.2K pull-up resistor).
P77	SCL7	I2C7 clock line - Serial Clock (connected with 2.2K pull-up resistor).

Usage example with i2c-tools

First, install the i2c-tools package if it is not already installed:

```
$ sudo apt install i2c-tools
```

List all I2C buses available on the system:

```
$ i2cdetect -l
```

Example output:

```
i2c-3   i2c             Renesas RIIC adapter                    I2C adapter
i2c-4   i2c             Renesas RIIC adapter                    I2C adapter
i2c-8   i2c             Renesas RIIC adapter                    I2C adapter
i2c-9   i2c             Renesas RSCI I2C adapter                I2C adapter
i2c-10  i2c             Renesas RSCI I2C adapter                I2C adapter
```

Find the correct bus number for I2C7:

```
root@localhost:~# ls -l /sys/class/i2c-dev/
total 0
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-10 -> ../../devices/platform/soc/
12802800.i2c/i2c-10/i2c-dev/i2c-10
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-3 -> ../../devices/platform/soc/
14401000.i2c/i2c-3/i2c-dev/i2c-3
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-4 -> ../../devices/platform/soc/
14401400.i2c/i2c-4/i2c-dev/i2c-4
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-8 -> ../../devices/platform/soc/
11c01000.i2c/i2c-8/i2c-dev/i2c-8
lrwxrwxrwx 1 root root 0 Jul 12 01:53 i2c-9 -> ../../devices/platform/soc/
12802400.i2c/i2c-9/i2c-dev/i2c-9
root@localhost:~#
```

In this example, I2C7 corresponds to bus number 10.

Hint

How to identify the correct I2C bus number for I2C7?

You can identify the correct I2C bus number by checking the device tree source (DTS) file for the RZ/V2H RDK or by referring to the system documentation.

In this case, the device tree of RZ/V2H RDK defines the IC27 interface as **12802800.i2c**, which is mapped to **I²C bus number 10**.

Scan for I2C devices on bus 10:

```
$ sudo i2cdetect -y 10
```

SPI (Serial Peripheral Interface)

The SPI interface enables high-speed communication with peripheral devices using a master-slave architecture. It uses separate lines for data in, data out, clock, and chip select.

On the RZ/V2H RDK, the SPI pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 6: SPI6 Interface Pins

Pin Name	Function	Description
P93	SS0	Slave Select 0 signal for SPI6.
P94	SS1	Slave Select 1 signal for SPI6.
P90	MOSI6	Master Out Slave In (data output from master).
P91	MISO6	Master In Slave Out (data input to master).
P92	SCK6	Serial Clock signal for SPI6.

Usage example

List SPI devices:

```
$ ls /dev/spidev*
/dev/spidev1.0
```

Test SPI communication (please connect an appropriate SPI device to test):

```
$ spi-config -d /dev/spidev1.0 -q
/dev/spidev1.0: mode=0, lsb=0, bits=8, speed=2000000, spiready=0
$ echo -n -e "1234567890" | spi-pipe -d /dev/spidev1.0 -s 10000000 | hexdump
0000000 3231 3433 3635 3837 3039
000000a
```

UART (Universal Asynchronous Receiver/Transmitter)

The UART interface provides serial communication capabilities, allowing data exchange between the RZ/V2H RDK and other devices such as micro-controllers, GPS modules, or serial consoles.

On the RZ/V2H RDK, the UART pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 7: UART5 Interface Pins

Pin Name	Function	Description
P72	TXD5	UART5 transmit data (TX) signal.
P73	RXD5	UART5 receive data (RX) signal.

Usage example with minicom

First, install the minicom package if it is not already installed:

```
$ sudo apt install minicom
```

List available serial ports:

```
$ ls /dev/ttySC*
```

Example output:

```
/dev/ttySC0  /dev/ttySC1
```

Open a serial connection using minicom (replace /dev/ttySC1 with the appropriate device):

Tip

How to identify the correct UART port for UART5?

You can identify the correct UART port by checking the device tree source (DTS) file for the RZ/V2H RDK or by referring to the system documentation.

```
root@localhost:~# ls -l /sys/class/tty/ | grep ttySC
lrwxrwxrwx 1 root root 0 Jul 12 01:53 ttySC0 -> ../../devices/
platform/soc/11c01400.serial/11c01400.serial:0/11c01400.serial:0.0/
tty/ttySC0
lrwxrwxrwx 1 root root 0 Jul 12 01:53 ttySC1 -> ../../devices/
platform/soc/12802000.serial/12802000.serial:0/12802000.serial:0.0/
tty/ttySC1
root@localhost:~#
```

In this case, the device tree of RZ/V2H RDK defines the UART5 interface as **12802000.serial**, which is mapped to **/dev/ttySC1**.

```
$ sudo minicom -D /dev/ttySC1 -b 115200
```

GPIO (General Purpose Input/Output)

The GPIO pins allow for digital input and output operations, enabling interaction with various sensors, actuators, and other electronic components.

Please refer to the RZ/V2H RDK GPIO pinout documentation for detailed information on each GPIO pin's capabilities and functions.

Usage example with gpiod:

First, install the gpiod package if it is not already installed:

```
$ sudo apt install gpiod
```

List available GPIO chips:

```
$ gpiodetect
```

List lines for a specific GPIO chip (e.g., gpiochip1):

```
$ gpioinfo gpiochip1
```

Set a GPIO line as output and change its value:

```
# Set GPIO line 92 high - turn on LED D4
$ gpioset --mode=signal gpiochip1 92=1
```

Read the value of a GPIO line:

```
$ gpioget gpiochip1 92
```

PCM (Pulse Code Modulation)

The PCM interface is used for audio data transmission, allowing the RZ/V2H RDK to connect with audio codecs and other audio peripherals.

On the RZ/V2H RDK, the PCM pins are located on the Raspberry Pi GPIO 40-pin header as follows:

Table 8: PCM Audio Interface Pins

Pin Name	Function	Description
PA6	PCM_DIN	PCM data input (from external audio device to RZ/V2H).
P97	PCM_OUT	PCM data output (from RZ/V2H to external audio device).
P96	PCM_WS	PCM word select (frame sync) signal.
P95	PCM_CLK	PCM bit clock signal.

TODO: Add usage examples for PCM interface.

Other interfaces

7. Mini-HDMI

The RZ/V2H Robotic Development Kit features a Mini-HDMI port for video output to an external display. To use this interface, connect a Mini-HDMI cable from the board to a compatible monitor.

TODO: Add more information, with Ubuntu Core, there is no display output by default.

8. USB-UART

The RZ/V2H RDK includes a USB-UART interface for serial communication and debugging purposes. This interface allows you to connect the board to a host computer via a USB cable and access the serial console.

We recommend using a terminal emulator such as *minicom* or *Tera Term* to connect to the USB-UART interface.

The configuration settings for the serial connection are as follows:

- Baud Rate: 115200
- Data Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

9. JTAG 10-pin

The RZ/V2H RDK provides a JTAG 10-pin interface for debugging and programming CM33 and two CR8 cores.

This interface is essential for low-level debugging and development tasks in Multi-Core applications.

For more information on using the JTAG interface, please refer to the RZ/V2H RDK *Multi-OS Development Section*.

3 RZ/V2H Advance Features

The RZ/V2H Advance Features section provides detailed documentation on the advanced functionalities available on the RZ/V2H platform, including hardware accelerators and multi-OS capabilities.

3.1 DRP-AI

This section provides an overview of the DRP-AI (Dynamically Reconfigurable Processor for AI) Driver available on the RZ/V2H platform, along with instructions on how to utilize its features effectively.

3.1.1 Overview

The RZ/V2H DRP-AI Driver enables the use of the DRP-AI (Dynamically Reconfigurable Processor for AI) hardware accelerator on the RZ/V2H platform.

This driver facilitates efficient execution of AI inference tasks by offloading computations to the DRP-AI, thereby improving performance and reducing CPU load and high power efficiency.

The DRP-AI device driver provides an interface to easily handle the AI inference execution function of DRP-AI. So that there is no hardware knowledge required for the user.

3.1.2 Concepts

What is Dynamically Reconfigurable Processor (DRP)?

DRP is the hardware IP (Intellectual Property) that can dynamically change its hardware (arithmetic logic circuit) configuration.

Its main advantages are the ability to reduce surface area and power consumption whilst maintaining high performance.

Dynamic Reconfiguration can change the configuration of the arithmetic circuit in execution.

This image below shows an example of dynamic reconfiguration:

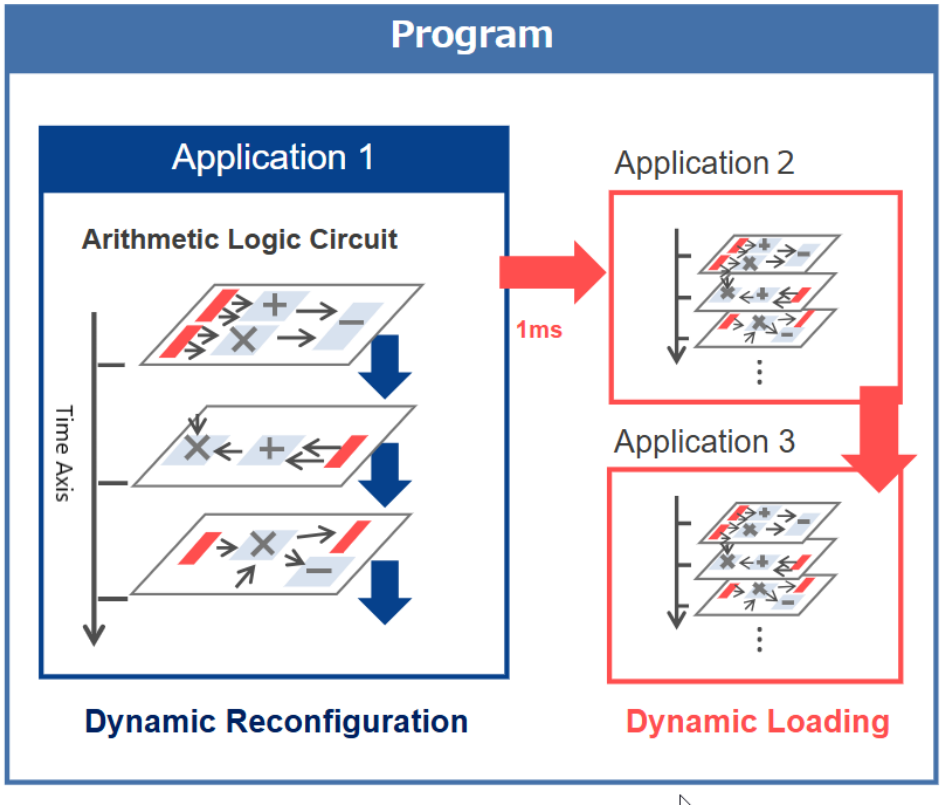

Fig. 1: DRP Dynamic Reconfiguration

What is DRP-AI?

DRP-AI is a specialized version of DRP designed specifically for AI (Artificial Intelligence) processing tasks.

By combine the DRP and AI-MAC (AI Matrix Arithmetic Circuit) to accelerate AI inference tasks efficiently.

See also

On the RZ/V2H platform, the DRP-AI3 is used as the DRP-AI hardware accelerator. For more information about the DRP-AI3, refer to the:

DRP-AI3 White Paper: Next Generation Highly Power-Efficient AI Accelerator (DRP AI3).

Software Package: AI Accelerator: DRP-AI.

DRP-AI Driver Architecture:

- Buffer to reuse input data
- Switches to avoid zero data processing
- Controller to optimize operation flow (scheduling)

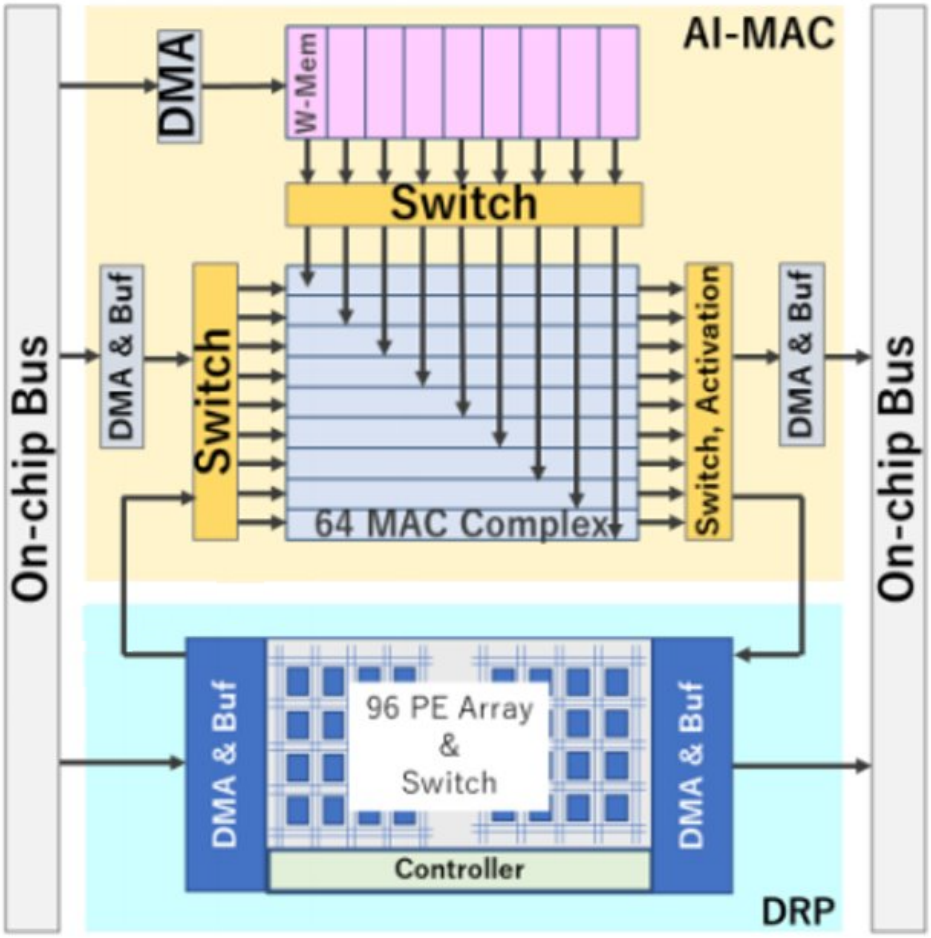

Fig. 2: DRP-AI Driver Architecture

DRP-AI Driver Execution Flow:

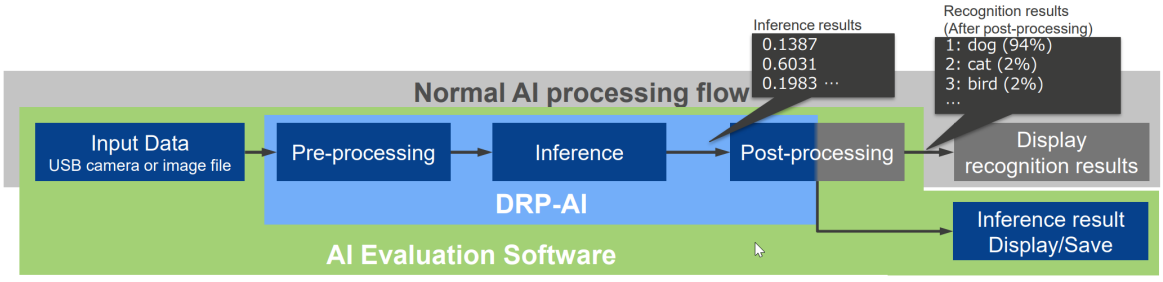

Fig. 3: DRP-AI Driver Execution Flow

The DRP-AI Driver handles the following tasks to execute AI inference on the DRP-AI:

1. Pre-processing: Prepares input data for DRP-AI processing, including format conversion, crop the image, normalization, etc...
2. Inference execution: Manages the execution of the AI model on the DRP-AI hardware.
3. Post-processing: Processes the output data from DRP-AI to obtain the final inference results.

See also

For more detail information about the DRP-AI Driver, refer to the RZ/V2H DRP-AI Driver.

It provide API functions to help you get started with DRP-AI Driver development on the RZ/V2H platform.

3.1.3 BYOM AI model support

The DRP-AI supports BYOM (Bring Your Own Model) AI models, allowing users to deploy their custom-trained AI models on the RZ/V2H platform.

Getting Started

To enable BYOM support, users need to convert their AI models into a format compatible with the DRP-AI using the:

Extension package of TVM Deep Learning Compiler for Renesas DRP-AI accelerators powered by EdgeCortix MERA™ (DRP-AI TVM).

This package provides the necessary tools and libraries to facilitate the conversion process, ensuring that the models can leverage the DRP-AI's capabilities effectively.

See also

For more information about the DRP-AI TVM extension package, refer to the:

- DRP-AI TVM Extension Package.
- DRP-AI TVM on RZ/V series.

Install the DRP-AI TVM extension package

To install the DRP-AI TVM extension package, follow the instructions provided in the package's GitHub repository: Installation.

We recommend installing DRP-AI TVM with Docker for ease of setup and to avoid dependency issues.

BYOM Development Flow

The typical development flow for deploying BYOM AI models on the RZ/V2H platform using the DRP-AI TVM extension package involves the following steps:

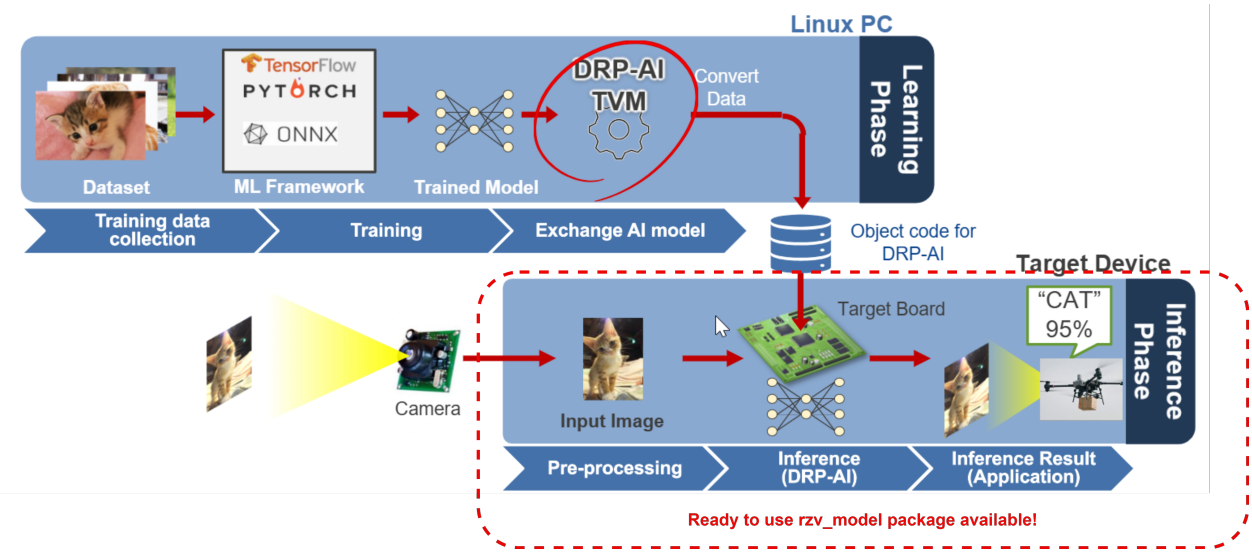

Fig. 4: BYOM Development Flow

1. Training data collection: Gather and prepare the dataset required for training the AI model.
2. Model training: Use a deep learning framework (e.g., TensorFlow, PyTorch, ONNX) to train the AI model on the collected dataset.
3. Exchange AI model: Convert the trained AI model into a format compatible with the DRP-AI using the DRP-AI TVM extension package.
4. Deployment: Deploy the converted model onto the RZ/V2H platform and integrate it with the DRP-AI Driver for inference execution.

Hint

In the deployment step, the ready to use **rzv_model** package is provided to simplify the integration of compiled models with the DRP-AI Driver.

We also provide some example of complete flow about developing some popular AI models with DRP-AI TVM extension package. Refer to the *DRP-AI with rzv_model tutorials* for more details.

Training data collection and Model training

For training data collection and model training, users can utilize popular deep learning frameworks such as TensorFlow, PyTorch, or ONNX.

See also

List of AI models that Renesas has verified for conversion with the DRP-AI TVM: Model list for RZ/V2H.

Note that, the above list is not exhaustive, and users can attempt to convert other models as well.

Exchange AI model

To convert the trained AI model into a format compatible with the DRP-AI for V2H target device, follow the instructions provided in the DRP-AI TVM extension package repository: Compile with Sample Scripts.

> **Tip**
>
> Follow the convert tips to easily convert your AI model: How to convert.

Deployment

For deploying the converted AI model onto the RZ/V2H platform, the *rzv_model* package is provided to facilitate the integration process.

The *rzv_model* package providing AI model abstractions and implementations for Renesas RZ/V MPU platforms.

This package implements various models for computer vision tasks using the DRP-AI accelerator.

Refer to the next section for more details about the *rzv_model* package and its usage.

3.1.4 The rzv_model package

Base Framework

The **rzv_model** package provides a flexible and modular framework for deploying AI models optimized on DRP-AI driver for the RZ/V2H platform. It includes the following core features:

- Abstracted model interface with hardware acceleration support.
- Common pre-processing and post-processing utilities for image-based inference.
- DRP-AI runtime integration for efficient inference execution.
- Support multiple AI model running at the same time with DRP-AI driver.
- Support for both YUV422 and RGB image formats.

Implemented Models

The following AI models are implemented, optimized and ready to use with DRP-AI acceleration.

Object Detection Models

- YOLOX Base Model
- YOLOX Hand Detection
- YOLOX Pascal VOC Detection
- Gold YOLOX Hand Detection
- YOLOv8 Base Model
- YOLOv8n Rock Paper Scissors Gesture Detection

Pose Estimation Models

- HRNetV2 Base Model
- HRNetV2 Hand Landmark Model
- RTMPose Base Model
- RTMPose Hand Model

- MediaPipe Hand Landmark Model

Package structure

The **rzv_model** package is organized into the following structure:

```
rzv_model/
  ├── CMakeLists.txt
  ├── config
  │   └── models
  ├── include
  │   └── rzv_model
  │       ├── base_model.hpp
  │       ├── model_specific.hpp
  │       └── utils.hpp
  ├── package.xml
  ├── README.md
  └── src
      ├── base_model.cpp
      ├── model_specific.cpp
      ├── platform
      │   ├── MeraDrpRuntimeWrapper.cpp
      │   ├── MeraDrpRuntimeWrapper.h
      │   ├── PreRuntime.h
      │   └── PreRuntimeV2H.cpp
      └── utils.cpp
```

- The **config/models** directory contains configuration files for each supported AI model, including the output from the DRP-AI TVM conversion step.
- The **include/rzv_model** directory contains the header files defining the base model class and utility functions.
- The **src** directory contains the implementation of the base model, platform-specific runtime wrappers, and utility functions.
- The **CMakeLists.txt** and **package.xml** files are used for building and packaging the library.

Architecture

The **rzv_model** package follows a **modular architecture** designed for extensibility, maintainability, and efficient deployment on DRP-AI.

- **Base Model:** Provides the *BaseModel* class, which implements shared functionalities such as model loading, pre-processing, inference execution, and result handling.
- **Model-Specific Implementations:** Each AI model (e.g., YOLOX, YOLOv8, HRNet, RTMPose) inherits from the base class and extends it with task-specific logic such as detection parsing or key point extraction.
- **Utility Modules:** Contain helper functions for image pre-processing, tensor conversion, normalization, and post-processing visualization.

This modular design enables developers to easily integrate new AI models and customize pre-processing or inference pipelines for various use cases on the RZ/V2H platform.

How to use the rzv_model package

Input requirements files

After complete the *Exchange AI model* step, the output should contain the compiled model files including:

```
Ouput_Directory/
  |-- deploy.json
  |-- deploy.params
  |-- deploy.so
  |-- input_0.bin
  `-- preprocess
      |-- addr_map.txt
      |-- aimac_cmd.bin
      |-- aimac_desc.bin
      |-- aimac_param_cmd.bin
      |-- aimac_param_desc.bin
      |-- drp_config.mem
      |-- drp_desc.bin
      |-- drp_param.bin
      |-- drp_param_info.txt
      `-- weight.bin
```

Also, to enable multiple AI models running simultaneously with the DRP-AI driver, a special file called `addr_map.txt` is required.

This `addr_map.txt` file is for the model in the inference phase. The purpose is to get the memory size used as multiple models scenario each model shall be allocated a memory block in advance before running.

This file is different from the one generated in the **Ouput_Directory** folder.

To obtain the `addr_map.txt` file, locate it in the **temp** folder created during the model conversion process.

For example, if you compile the model in the **/drp-ai_tvm/tutorials/** directory, a **temp** folder will be generated automatically.

You can find the `addr_map.txt` file in the following path:

`/drp-ai_tvm/tutorials/temp/<date_time>/tvmgen_default_tvmgen_default_mera_drp_main_*/drp_compilation_output/`

There might be several subdirectories representing different inference stages (executed by DRP-AI or CPU) that each contain an `addr_map.txt` file.

The correct file to use is the one with the largest memory address allocation, corresponding to the **maximum drp_desc value**, as it represents the final and complete memory size used by the AI Model with DRP-AI driver.

How to calculate the memory size used by the model?

- Open the `addr_map.txt` file with a text editor.

This file contains several lines, each representing a memory block with its start address and size. For example:

```
data_in 32ec4c0 dc00
data 32fa0c0 41040
data_out 333b100 12c00
work 334dd00 80
weight 334dd80 bd80
drp_config 3359b00 87480
aimac_param_cmd 33e0f80 140
```

(continues on next page)

(continued from previous page)

```
aimac_param_desc 33e10c0 50
aimac_cmd 33e1140 1300
aimac_desc 33e2440 170
drp_param 33e2600 350
drp_desc 33e2980 380
```

- Calculate the total memory size by summing the sizes of all `drp_desc` entries:

  In this example, the description of `drp_desc` is: start address `33e2980` and size `380`.

  Therefore, the total memory size for `drp_desc` is: `33e2980` (start address) + `380` (size) = `33e2d00`.

- Find the correct `addr_map.txt` file that contains the largest `drp_desc` value, corresponding to the total memory size used by the AI model with the DRP-AI driver.

Construct the Model configuration files

To construct the model, import the **Input requirements files** above with the following structure:

```
Model_name/
  |-- addr_map.txt
  |-- deploy.json
  |-- deploy.params
  |-- deploy.so
  `-- preprocess
      |-- addr_map.txt
      |-- aimac_cmd.bin
      |-- aimac_desc.bin
      |-- aimac_param_cmd.bin
      |-- aimac_param_desc.bin
      |-- drp_config.mem
      |-- drp_desc.bin
      |-- drp_param.bin
      |-- drp_param_info.txt
      `-- weight.bin
```

Note that, the top-level `addr_map.txt` file is required for multiple models running with DRP-AI driver.

This *Model_name* folder will place under the **config/models** directory of the **rzv_model** package.

Post-processing configuration

Each model may have different post-processing requirements based on its specific task (e.g., object detection, pose estimation).

To customize the post-processing behavior, you can modify the corresponding model-specific implementation files located in the **src/** directory of the **rzv_model** package.

The details of post-processing configuration are not covered in this section. Please refer to the example in the next section for a clearer understanding.

> **Hint**
>
> There are some sample applications from Renesas, which have the custom post-processing for the specific models.

You can refer to these applications for reference on implementing the AI model post-processing logic:

- RZ/V AI Applications Repository.
- Ignitarium Renesas - RZ/V AI Applications.

Example of using rzv_model package

Load the model and perform inference using the **rzv_model** package with the following code snippet:

```
// Example using HRNetV2 Hand Landmark model
auto model = std::make_unique<rzv_model::HRNetV2HandLandmarkModel>();
model->load(model_path);

// Prepare input
rzv_model::ModelInput input{image, roi};

// Run inference
auto result = model->run<rzv_model::KeyPointResult>(input);
```

3.1.5 DRP-AI with rzv_model tutorials

This section provides tutorials on how to use the **rzv_model** package to deploy AI models on the RZ/V2H platform with DRP-AI acceleration.

Make sure you have completed the steps in the *BYOM AI model support* section to set up the DRP-AI TVM extension package in your development machine.

Prequisites

On the development machine, which has the DRP-AI TVM environment, clone the Hand Models repository, which contains the necessary model files and conversion scripts for the tutorials.

```
$ git clone https://partnergitlab.renesas.solutions/sst1/industrial/ws078/ai/
hand_models.git
```

Then, set the hand models directory environment variable:

```
$ export HAND_MODELS_DIR=<path_to_cloned_hand_models_directory>
```

Next, change to the TVM tutorials directory and copy the compilation script:

```
$ cd $TVM_ROOT/tutorials
$ cp ${HAND_MODELS_DIR}/compilation/compile_onnx_model_quant_<model>.py .
```

Follow the specific tutorial instructions below to convert and deploy different hand detection and landmark models using the **rzv_model** package.

YOLOX Object Detection Tutorial

This tutorial describes how to use the **YOLOX Hand Detection** model for object detection with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the *Prequisites step of the tutorial* before proceeding.

For this tutorial, we will utilize the pre-trained YOLOX model.

Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the *hand_models/compilation* directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- YOLOX Hand

```
$ python3 compile_onnx_model_quant_yolox_hand_fhd_crop.py \
    $HAND_MODELS_DIR/palm_detection/yolox/yolox_hand.onnx \
    -o ../data/yolox_hand_fhd_crop \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,640,640 \
    -v 100 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_hands/
```

- Gold YOLO Nano

```
$ python3 compile_onnx_model_quant_gold_yolo_fhd_crop.py \
    $HAND_MODELS_DIR/palm_detection/gold_yolo/gold_yolo_n_hand_0303_0.
4172_1x3x480x640.onnx \
    -o ../data/gold_yolo_n_fhd_crop \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,480,640 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
    -v 100
```

Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to *the rzv_model package usage*.

YOLOv8 Object Detection Tutorial

This tutorial describes how to use the **YOLOv8n Rock Paper Scissors Gesture Detection** model for object detection with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the *Prequisites step of the tutorial* before proceeding.

Installation requirements

- Ultralytics YOLOv8
- Dataset for Rock Paper Scissors Gesture Detection

  You can use the Rock-Paper-Scissors Dataset from roboflow.

  Please download and prepare the dataset as YOLOv8 format for training.

Train the Model using Ultralytics YOLOv8 and transfer learning method

Please download the dataset above with the YOLOv8 format and locate the `data.yaml` file, replace the path in the below Python script and save as `train_model_yolov8.py`

```
from ultralytics import YOLO

# Loading pre-trained model
# Can select between: yolov8n.pt yolov8s.pt yolov8m.pt yolov8l.pt yolov8x.pt
model = YOLO('yolov8n.pt')

# Display model information
model.info()

# Train the model with 80 epochs and set the input image size is 640x640
# The number of epochs can be changed to adapt to your requirements
results = model.train(data="<path-to-the-dataset-folder>/rock-paper-scissors-14/data.yaml", epochs=80, imgsz=640)
```

Run the training process:

```
$ python3 train_model_yolov8.py
```

For faster training time, the GPU with appropriate driver should be installed.

Result will be located at: `<workspace-directory>/runs/detect/trainX/weights/best.pt` where X will increase each time you run the training process.

Export and cut the model

Since some parts of the post-processing phase of YOLOv8 model cannot be handled by the DRP-AI hardware, we need to remove certain steps to ensure it can run on the DRP-AI hardware.

Please follow this documentation to learn how to export the model to ONNX format and cut the model:

- How to convert yolov8 onnx models V2H.md

After exporting and cutting the model, you will get the ONNX model file named `yolov8n_rps_cut.onnx` (the name can be different based on your configuration).

Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the *hand_models/compilation* directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- YOLOv8n Rock Paper Scissors Gesture Detection

```
$ python3 compile_onnx_model_quant_yolov8_fhd_crop.py \
    $HAND_MODELS_DIR/palm_detection/yolov8/yolov8n_rps_cut.onnx \
    -o yolov8_rps \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,640,640 \
    -v 100 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_rps_hands
```

Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to *the rzv_model package usage*.

MediaPipe Hand Landmark Tutorial

This guide describes the process for converting **MediaPipe Hand Landmark Detection** models from **TFLite** to **ONNX** format for compatibility with **DRP-AI**.

Please complete the *Prequisites step of the tutorial* before proceeding.

Download Models

- Reference: MediaPipe Models List
- MediaPipe Hand Landmarker Models Documentation
- Hand Landmark Full Model
- Hand Landmark Lite Model
- Palm Detection Full Model
- Palm Detection Lite Model

Set Up Conversion Environment

- Reference: tensorflow-onnx GitHub Repository — Convert TensorFlow, Keras, TensorFlow.js, and TFLite models to ONNX.
- Create a dedicated conda environment with Python 3.10:

```
$ conda create --name tf2onnx python=3.10
$ conda activate tf2onnx
```

Conversion Process

- **Important:** The supported ONNX opset for the DRP-AI translator is **v12**.
- Install dependencies (downgrade NumPy to resolve compatibility issues):

```
$ pip install tensorflow-onnx
$ pip install numpy==1.26.4
```

- Convert the models to ONNX format (using NCHW layout):

```
$ python -m tf2onnx.convert --tflite hand_landmark_full.tflite --opset 12
--output hand_landmark_full.onnx --inputs-as-nchw input_1
$ python -m tf2onnx.convert --tflite palm_detection_full.tflite --opset
12 --output palm_detection_full.onnx --inputs-as-nchw input_1
```

- For the Lite version, use:

```
$ python -m tf2onnx.convert --tflite hand_landmark_lite.tflite --opset 12
--output hand_landmark_lite.onnx --inputs-as-nchw input_1
$ python -m tf2onnx.convert --tflite palm_detection_lite.tflite --opset
12 --output palm_detection_lite.onnx --inputs-as-nchw input_1
```

Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the *hand_models/compilation* directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

Best performance and accuracy:

```
$ SPARSE_ENABLE=false python3 compile_onnx_model_quant_mp_handlandmark_fhd_
crop.py \
    $HAND_MODELS_DIR/hand_landmark_estimation/mp_hand_landmark/hand_landmark_
full.onnx \
    -o ../data/mp_handlandmark_fhd_crop \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,224,224 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
    -v 100 \
    -p "--calibrate_method Entropy"
```

MediaPipe Evaluation model:

- Full model:

```
$ SPARSE_ENABLE=false python3 compile_onnx_model_quant_mediapipe.py \
    $HAND_MODELS_DIR/hand_landmark_estimation/mp_hand_landmark/hand_landmark_
full.onnx \
    -o ../data/mediapipe_entropy \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,224,224 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
    -v 100 \
    -p "--calibrate_method Entropy"
```

- Lite model:

```
$ SPARSE_ENABLE=false python3 compile_onnx_model_quant_mediapipe.py \
    $HAND_MODELS_DIR/hand_landmark_estimation/mp_hand_landmark/hand_landmark_
lite.onnx \
    -o ../data/mediapipe_lite_entropy \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,224,224 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
    -v 100 \
    -p "--calibrate_method Entropy"
```

Optional: MediaPipe Palm Detection

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false python3 compile_onnx_model_quant_
mp_palm_det.py \
    $HAND_MODELS_DIR/palm_detection/mediapipe/palm_detection_full.onnx \
    -o ../data/mp_palm_det \
    -t $SDK \
    -d $TRANSLATOR \
    -c $QUANTIZER \
    -s 1,3,192,192 \
    --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
    -v 100
```

Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to *the rzv_model package usage*.

RTMPose Hand Landmark Tutorial

This tutorial describes how to use the **RTMPose Hand Landmark** model for hand landmark estimation with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the *Prequisites step of the tutorial* before proceeding.

For this tutorial, we will utilize the pre-trained RTMPose model.

Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the *hand_models/compilation* directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- RTMPose Hand Landmark

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false python3 compile_onnx_model_quant_
rtmpose_fhd_crop.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/rtmpose/rtmpose.onnx \
      -o ../data/rtmpose_fhd_crop \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
      -v 100 \
      -p "--calibrate_method Entropy --node_to_exclude /Shape,/Slice,/Concat,/
Reshape,/mlp/mlp.0/ReduceL2,/mlp/mlp.0/Mul,/mlp/mlp.0/Clip,/mlp/mlp.0/Div,/
mlp/mlp.0/Mul_1,/mlp/mlp.1/MatMul,/gau/ln/ReduceL2,/gau/ln/Mul,/gau/ln/Clip,/
gau/ln/Div,/gau/ln/Mul_1,/gau/uv/MatMul,/gau/act_fn/Sigmoid,/gau/act_fn/Mul,/
gau/Split,/gau/Unsqueeze,/gau/Mul,/gau/Add,/gau/Split_1,/gau/Squeeze_1,/gau/
Transpose,/gau/Squeeze,/gau/MatMul,/gau/Div,/gau/Relu,/gau/Mul_1,/gau/MatMul_
1,/gau/Mul_2,/gau/o/MatMul,/gau/res_scale/Mul,/gau/Add_1,/cls_x/MatMul,/cls_y/
MatMul"
```

- RTMPose Evaluation with basic compilation

```
$ python3 compile_onnx_model_quant_rtmpose.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/rtmpose/rtmpose.onnx \
      -o ../data/rtmpose_freihand \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
      -v 100
```

- RTMPose Evaluation with Optimized Compilation (Recommended)

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false python3 compile_onnx_model_quant_
rtmpose.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/rtmpose/rtmpose.onnx \
      -o ../data/rtmpose_optimized \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/selected_freihand/ \
      -v 100 \
      -p "--calibrate_method Entropy --node_to_exclude /Shape,/Slice,/Concat,/
Reshape,/mlp/mlp.0/ReduceL2,/mlp/mlp.0/Mul,/mlp/mlp.0/Clip,/mlp/mlp.0/Div,/
mlp/mlp.0/Mul_1,/mlp/mlp.1/MatMul,/gau/ln/ReduceL2,/gau/ln/Mul,/gau/ln/Clip,/
gau/ln/Div,/gau/ln/Mul_1,/gau/uv/MatMul,/gau/act_fn/Sigmoid,/gau/act_fn/Mul,/
gau/Split,/gau/Unsqueeze,/gau/Mul,/gau/Add,/gau/Split_1,/gau/Squeeze_1,/gau/
Transpose,/gau/Squeeze,/gau/MatMul,/gau/Div,/gau/Relu,/gau/Mul_1,/gau/MatMul_
1,/gau/Mul_2,/gau/o/MatMul,/gau/res_scale/Mul,/gau/Add_1,/cls_x/MatMul,/cls_y/
MatMul"
```

Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to *the rzv_model package usage*.

HRNetV2 Hand Landmark Tutorial

This tutorial describes how to use the **HRNetV2 Hand Landmark** model for hand landmark estimation with DRP-AI acceleration on the Renesas RZ/V2H platform.

Please complete the *Prequisites step of the tutorial* before proceeding.

For this tutorial, we will utilize the pre-trained HRNetV2 model.

Compile with DRP-AI TVM Extension Package

Run the compilation script provided in the *hand_models/compilation* directory to compile the ONNX models for DRP-AI on the RZ/V2H platform.

- HRNetv2 Hand Landmark

```
$ python3 compile_onnx_model_quant_hrnetv2_fhd_crop.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.
onnx \
      -o ../data/hrnetv2_hands_fhd_crop \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
      -v 100
```

- HRNetv2 Evaluation with Selected Hands Dataset

```
$ python3 compile_onnx_model_quant_hrnetv2.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.
onnx \
      -o ../data/hrnetv2_selected_hands \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
      -v 100
```

- HRNetv2 Evaluation with FreiHand Dataset

```
$ python3 compile_onnx_model_quant_hrnetv2.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.
onnx \
      -o ../data/hrnetv2_freihand \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/freihand_rois/ \
      -v 100
```

- HRNetv2 Evaluation with Entropy Calibration

```
$ SPARSE_ENABLE=false OPTIMIZER_ENABLE=false python3 compile_onnx_model_quant_
hrnetv2.py \
      $HAND_MODELS_DIR/hand_landmark_estimation/hrnetv2/hrnetv2_onehand10k.
onnx \
      -o ../data/hrnetv2_selected_hands_entropy \
      -t $SDK \
      -d $TRANSLATOR \
      -c $QUANTIZER \
      -s 1,3,256,256 \
      --images $HAND_MODELS_DIR/dataset/scripts/selected_hands/ \
      -v 100 \
      -p "--calibrate_method Entropy"
```

Deploy and Test on RZ/V2H RDK

After compiling the models, transfer the generated model files to the RZ/V2H RDK and run inference using the **rzv_model** package.

For more detail, please reference to *the rzv_model package usage*.

3.2 Video Codec Library

The RZ/V2H Video Codec Library provides hardware-accelerated (combined with *DRP IP*) video encoding and decoding capabilities.

This software supports the following features:

- Support for H.264 decoding and encoding
- Support for H.265 decoding and encoding

Important

In this release, the Video Codec Library is available only for the Yocto image: *Renesas Core Image Weston*.

See also

For more information on how to use the Video Codec Library, refer to the Linux Interface Specification GStreamer.

It provide GStreamer plugins for easy integration with available Video Codec functionalities.

3.3 OpenCV Accelerator

3.3.1 Overview

The RZ/V2H OpenCV Accelerator (OpenCVA) leverages the OpenCV library to provide optimized computer vision functionalities on the RZ/V2H platform by using the *DRP IP*.

Based on the OpenCV version 4.10.0, this feature enables efficient image processing and computer vision tasks by offloading some computations to the DRP, thereby enhancing performance and reducing CPU load.

See also

For more detail information about OpenCVA, refer to the RZ/V2H OpenCV Accelerator.

3.3.2 How to use OpenCVA

OpenCVA leverages the DRP's processing capability to enhance specific functions of the OpenCV library .You can use OpenCVA same as OpenCV as usual and you do not need to consider of OpenCVA architecture.

OpenCVA is automatically executed by DRP as follows if it matches the conditions under which DRP can be used.

For the DRP using conditions, see OpenCVA API specification and condition for using DRP.

OpenCVA can disable DRP, for each function. See API functions to control OpenCVA for details.

The following table lists the OpenCV functions that can be executed using DRP in the OpenCVA:

Table 1: OpenCV Functions Supported by DRP

OpenCV Function Name	Function
resize	Image resize.
cvtColor	Change color space.
cvtColorTwoPlane	Change color space.
GaussianBlur	Gaussian filter process.
dilate	Areas of bright regions grow.
erode	Areas of dark regions grow.
morphologyEX	Combination of dilate and erode.
filter2D	Image convolving.
Sobel	Extracting image edges.
adaptiveThreshold	Transforms a grayscale image to a binary image according to the formula.
matchTemplate	Compares a template against overlapped image regions.
wrapAffine	Transforms the source image using the 2x3 matrix.
wrapPerspective	Transforms the source image using the 3x3 matrix.
pyrDown	Downsampling step of the Gaussian pyramid construction.
pyrUp	Upsampling step of the Gaussian pyramid construction.
FAST	Detects corners using the FAST algorithm.
remap	Applies a generic geometrical transformation to an image.

Note

On Ubuntu OS, OpenCVA library is installed in the following path:

*/usr/lib/aarch64-linux-gnu/renesas/libopencv**

/usr/include/opencv4/renesas/opencv4

3.4 RZ/V Multi-OS

The RZ/V2H RDK equipped with the following CPUs:

- Cortex-A55 (4 cores): Runs Linux OS
- Cortex-M33 (1 core): Runs MCU based OS such as FreeRTOS, Zephyr, or BareMetal
- Cortex-R8 (2 cores): Runs MCU based OS such as FreeRTOS, Zephyr, or BareMetal

This package provides the necessary software components and examples to enable and manage multiple operating systems running concurrently on the RZ/V2H platform.

3.4.1 Overview

The RZ/V Multi-OS feature allows the RZ/V2H platform to run multiple operating systems concurrently.

This capability is particularly useful for applications that require separation of tasks, such as running a real-time operating system (RTOS) alongside a general-purpose OS like Linux

Important

On our default IPL, the following features are enabled by default:

- Remoteproc support
- CM33 and CR8 invocation from U-Boot

Attention

When using the J-Link debugger, ensure that the SW1-6 switch on the RZ/V2H RDK board is set to the "ON" position to enable JTAG debug functionality.

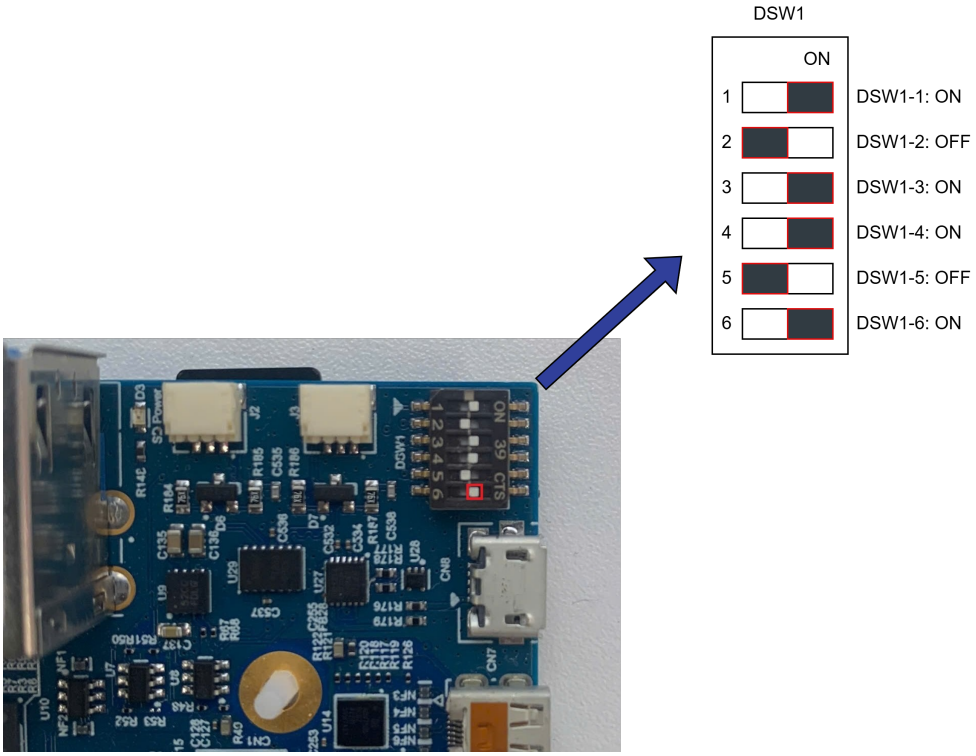

Fig. 5: RZ/V2H RDK JTAG Switch 6 ON

3.4.2 Useful References

The following documents and repositories are recommended for users who want to explore more about multi-OS integration and RZ/V FSP software development:

- RZ/V Multi-OS Package v3.2.0: Official software package enabling Linux + FreeRTOS/BareMetal multi-OS solutions on RZ/V2H.
- RZ/V2H Quick Start Guide for RZ/V Multi-OS Package: Step-by-step guide to integrate and configure Multi-OS environments.
- Getting Started with RZ/V Flexible Software Package (FSP): Instructions for developing and managing applications using RZ/V FSP.
- FSP Example Project Usage Guide: Detailed examples demonstrating peripheral control, communication, and middleware setup.

- micro_ros_renesas_demos: Demonstrations and documentation for using Micro-ROS with Renesas platforms, including agent setup and communication examples.

We recommend reviewing these resources to gain a deeper understanding of multi-OS capabilities and how to effectively utilize them in your projects.

3.4.3 Requirement

- Ubuntu or Windows machines that can install the e2Studio and Flexible Software Package (FSP) for the Renesas RZ/V MPU Series.
- Segger J-Link (**firmware version 7.96e**): JTAG debugger for flashing firmware and debugging.
- RZ/V FSP version 3.1 installed on the development machine. Follow the instructions in the RZ/V FSP Getting Started Guide to set up the FSP.

3.4.4 Note for integration

The peripherals which are NOT enabled enter Module Standby Mode after Linux kernel is booted up. That means the peripherals used on CM33 side might stop working at that time.

To avoid such a situation, Multi-OS Package incorporates the patch below:

- `0003-Set-OSTM-for-MCPU-and-RCPU.patch`

This patch prevents GTM used in RPMsg demo program from entering Module Standby Mode. If you have any other peripherals which you would like to stop entering Module Standby implicitly, please update the patch as shown below:

```
diff --git a/drivers/clk/renesas/r9a09g057-cpg.c b/drivers/clk/renesas/
r9a09g057-cpg.c
index 0468eb17305b..146d5b444e8c 100644
--- a/drivers/clk/renesas/r9a09g057-cpg.c
+++ b/drivers/clk/renesas/r9a09g057-cpg.c
@@ -302,14 +302,14 @@ static const struct rzv2h_mod_clk r9a09g057_mod_clks[] _
_initconst = {
   DEF_MOD("rcpu_cmtw1_clkm",              CLK_PLLCLN_DIV32, 4, 0, 2, 0),
   DEF_MOD("rcpu_cmtw2_clkm",              CLK_PLLCLN_DIV32, 4, 1, 2, 1),
   DEF_MOD("rcpu_cmtw3_clkm",              CLK_PLLCLN_DIV32, 4, 2, 2, 2),
-    DEF_MOD("gtm_0_pclk",                 CLK_PLLCM33_DIV16, 4, 3, 2, 3),
-    DEF_MOD("gtm_1_pclk",                 CLK_PLLCM33_DIV16, 4, 4, 2, 4),
+    DEF_MOD_CRITICAL("gtm_0_pclk",                CLK_PLLCM33_DIV16, 4, 3,
2, 3),
+    DEF_MOD_CRITICAL("gtm_1_pclk",                CLK_PLLCM33_DIV16, 4, 4,
2, 4),
   DEF_MOD("gtm_2_pclk",                   CLK_PLLCLN_DIV16, 4, 5, 2, 5),
   DEF_MOD("gtm_3_pclk",                   CLK_PLLCLN_DIV16, 4, 6, 2, 6),
-    DEF_MOD("gtm_4_pclk",                 CLK_PLLCLN_DIV16, 4, 7, 2, 7),
-    DEF_MOD("gtm_5_pclk",                 CLK_PLLCLN_DIV16, 4, 8, 2, 8),
-    DEF_MOD("gtm_6_pclk",                 CLK_PLLCLN_DIV16, 4, 9, 2, 9),
-    DEF_MOD("gtm_7_pclk",                 CLK_PLLCLN_DIV16, 4, 10, 2, 10),
+    DEF_MOD_CRITICAL("gtm_4_pclk",                CLK_PLLCLN_DIV16, 4, 7,
2, 7),
+    DEF_MOD_CRITICAL("gtm_5_pclk",                CLK_PLLCLN_DIV16, 4, 8,
2, 8),
+    DEF_MOD_CRITICAL("gtm_6_pclk",                CLK_PLLCLN_DIV16, 4, 9,
2, 9),
```

(continues on next page)

(continued from previous page)

```
+    DEF_MOD_CRITICAL("gtm_7_pclk",                    CLK_PLLCLN_DIV16, 4, 10,
2, 10),
   DEF_MOD("wdt_0_clkp",                       CLK_PLLCM33_DIV16, 4, 11, 2, 11),
   DEF_MOD("wdt_0_clk_loco",          CLK_QEXTAL, 4, 12, 2, 12),
   DEF_MOD("wdt_1_clkp",                       CLK_PLLCLN_DIV16, 4, 13, 2, 13),
--
```

Please edit the `r9a09g057-cpg.c` file accordingly and re-build the Linux Image to ensure the desired peripherals remain active during Multi-OS operation.

We recommend using the eSDK to build the Linux image to do it. For more details, please refer to the *Custom Linux Kernel and Device Tree*.

3.4.5 How to create the new project for RZ/V2H RDK

To create a new project for the RZ/V2H RDK board using the RZ/V Flexible Software Package (FSP) and e2Studio, follow these steps:

1. Open e2Studio and create a new Renesas FSP project.
2. Choose **File** > **New** > **Renesas C/C++ Project** > **Renesas RZ** > **Renesas RZ/V C/C++ FSP Project**.
3. Enter the Project name and location.
4. Select the **Custom User Board (for RZ/V2H)**, the Device **R9A09G057H44GBG**, and the target core: **CM33 or CR8**.
5. If you select the CR8 core, please select the appropriate **Preceding Project/Smart Bundle Details**.

   This setup is necessary to ensure that there are no conflicts in hardware resource usage between the CM33 and CR8 cores when both are running on the RZ/V2H platform.
6. Configure the **Build artifact type, RTOS selection and Sub-core selection** as needed.
7. Configure the **Project Templete Selection**, and click on the **Finish** button to create the project.
8. Once the project is created, you can start adding your application code and configuring the necessary peripherals using the FSP.

We also provide a sample project for RZ/V2H RDK board with Multi-OS Package.

Please follow the next section to import the sample project.

3.4.6 RZV2H RDK Multi-OS Example Packages

This section contains a collection of Multi-OS packages designed for applications on Renesas RZ/V MPU platforms, specifically targeting the RZ/V2H RDK.

These packages provide practical examples demonstrating how to operate and integrate Multi-OS environments on the RZ/V2H RDK, helping developers understand cross-core communication, system setup, and interaction between Linux and RTOS components.

Additionally, a demo showcasing Micro-ROS (uROS) running on the real-time CR8 core is supported. It demonstrates the implementation of Micro-ROS on an MCU-class core within the device.

Hardware supported

- Platform: Renesas RZ/V2H MPU
- Development Board: RZ/V2H RDK (SoC: R9A09G057H44GBG)

Software supported

- Target Operating System: Ubuntu 24.04
- RZ/V Multi-OS Package version 3.2
- RZ/V FSP version 3.1
- Micro XRCE-DDS Agent version 3.0.1
- Micro ROS Client Jazzy
- ROS2 Distribution: ROS2 Jazzy

Package Specification

Package	Target Core	Purpose / Description
Micro XRCE-DDS Agent	CA55 (Linux)	Provides the middleware agent running on the Linux core (CA55) for communication between Micro-ROS clients (running on RTOS CR8_0 core) and the ROS 2 environment on Linux via the XRCE-DDS protocol.
RZ/V2H RDK Blinky	CM33 (RTOS)	A simple LED blinking demo running on the CM33 core that verifies basic GPIO functionality and confirms that the RTOS environment is running correctly on the RZ/V2H RDK.
RZ/V2H RDK CM33 RPMsg Linux-RTOS Demo	CM33 (RTOS)	Demonstrates inter-core communication (RPMsg) between the Linux core (CA55) and the CM33 RTOS core, showing message exchange and synchronization.
RZ/V2H RDK CR8 Core0 RPMsg Linux-RTOS Demo	CR8_0 (RTOS)	Demonstrates RPMsg-based communication between the Linux core (CA55) and the CR8_0 real-time core, validating message passing and core coordination.
RZ/V2H RDK CR8 Core0 RPMsg Micro-ROS Demo	CR8_0 (RTOS)	Showcases Micro-ROS running on the CR8_0 real-time core, integrating the uROS client with the custom RPMsg transport layer for communication with Linux and ROS 2.

Installation Guide

Firmware Code for CM33/CR8

This section describes how to build and flash the firmware for the CM33/CR8 core using e² studio and the provided sample project.

1. Clone the CM33/CR8 project into your host machine.
2. Open **e² studio** and import the above project using **"Import Existing Project"**.
3. Open the configuration file.
4. Click **"Generate"** to generate configuration files.
5. Click **"Build Project"** and wait for the build process to complete.

6. Flash the firmware to the CM33/CR8 core using your preferred method (e.g., J-Link).

Important

The preceding project for all of CR8_0 packages is `RZ/V2H RDK CM33 RPMsg Linux-RTOS Demo`. Please import this CM33 project into your e² studio workspace and build it before using the CR8_0 packages.

Special Note for `RZ/V2H RDK CR8 Core0 RPMsg Micro-ROS Demo` Package

1. This demo includes a prebuilt libmicroros library. If you want to rebuild this library, use this project on the Ubuntu machine and perform the following steps:

   Go to **Project → Properties → C/C++ Build → Settings → Build Steps** tab and in **Pre-build steps**, add the command:

   ```
   cd ../micro_ros_renesas2estudio_component/library_generation && ./
   library_generation.sh "${cross_toolchain_flags}"
   ```

   Then click **Apply and Close** → **Build the project**.

2. The code includes a 30-second delay (in `main_task_entry.c` line 168) before initializing MCU tasks to prevent issues with PWM and I2C pin control on the CR8 core. By default, this delay is commented out.

   - If flashing via **J-Link**, this delay can be skipped.
   - However, when invoking the firmware from **U-Boot**, please enable this delay.

Cross Compile the Micro XRCE-DDS Agent

This section describes how to deploy and build the custom OpenAMP Micro-ROS agent application running on the CA55 core of the RZ/V2H platform.

Prerequisites: The Linux local host machine must have the Yocto SDK installed. It is recommended to use the *ROS2 cross-build Docker container*, since the environment is already fully set up.

1. Clone the `Micro-XRCE-DDS-Agent` to your local machine.
2. The custom OpenAMP agent source code is located at: `Micro-XRCE-DDS-Agent/examples/custom_agent`.
3. Navigate to the directory `Micro-XRCE-DDS-Agent/`.
4. Build the project:

   ```
   mkdir build && cd build

   # We have to disable the UAGENT_LOGGER_PROFILE because the current SDK
   cannot compile spdlog.
   # For native builds, we can remove the DUAGENT_LOGGER_PROFILE flag.
   cmake .. -DCMAKE_TOOLCHAIN_FILE=../coretexa55-toolchain.cmake \
            -DUAGENT_BUILD_USAGE_EXAMPLES=ON \
            -DUAGENT_LOGGER_PROFILE=OFF \
            -DCMAKE_BUILD_TYPE=Release \
            -DCMAKE_INSTALL_PREFIX=./arm64-install

   make -j$(nproc)
   make install
   ```

5. Wait until the build process completes.
6. Deploy the output to the target board by copying the output artifact:
   - On the **Host machine**:

```
# Copy the built CustomXRCEAgent binary to the arm64-install folder
for deployment
$ cp ./examples/custom_agent/CustomXRCEAgent arm64-install/bin

# Compress the arm64-install folder
$ tar -cf libdds_agent.tar.bz2 -C arm64-install .
```

   - Then copy `libdds_agent.tar.bz2` to the target board using **scp** or another file transfer method.
   - On the **Target machine**:

```
# Extract the archive
$ mkdir tmp-install
$ sudo tar -xf libdds_agent.tar.bz2 -C tmp-install

# Install libdds_agent to the system
$ cd tmp-install
$ sudo cp -r * /usr/local/
$ sudo ldconfig
```

Usage Guide

RPMsg Linux-RTOS Demo

This demo behaves identically to the version released in the **RZ/V Multi-OS Package**. For more details, refer to the RZ/V2H Quick Start Guide for the RZ/V Multi-OS Package.

1. Flash the `RPMsg Linux-RTOS Demo` firmware to the target board.
2. On the board's terminal, run the `rpmsg_sample_client` with sudo privilege:

```
root@localhost:~# rpmsg_sample_client
```

Example output:

```
root@localhost:~# rpmsg_sample_client
[694] proc_id:0 rsc_id:0 mbx_id:1
metal: warning:   metal_linux_irq_handling: Failed to set scheduler: -1.
metal: info:      metal_uio_dev_open: No IRQ for device 10480000.mbox-uio.
[694] Successfully probed IPI device
...
```

3. Based on the firmware you have flashed, select the corresponding option below and press **Enter** when prompted.

Input Option	Target Core / Firmware	Description
1	CM33 (Linux <=> RTOS RPMsg Demo)	Select this if you have flashed the `RZ/V2H RDK CM33 RPMsg Linux-RTOS Demo` firmware.
4	CR8_0 (Linux <=> RTOS RPMsg Demo)	Select this if you have flashed the `RZ/V2H RDK CR8 Core0 RPMsg Linux-RTOS Demo` firmware.

> **Note**
>
> Ensure that the firmware on your target board matches the selected option to avoid communication errors.

4. Example output:
   - If the Input Option is `1`:

```
[CM33]  received payload number 469 of size 486
[CM33] sending payload number 470 of size 487
[828] cond signal 1 sync:0
...
```

   - If the Input Option is `4`:

```
[CR8_0 ]  received payload number 469 of size 486
[CR8_0 ] sending payload number 470 of size 487
[790] cond signal 2 sync:0
...
```

5. By typing `e`, the sample program should terminate with the message shown below:

```
please input
> e
[xxx] 42f00000.rsctbl closed
[xxx] 43000000.vring-ctl0 closed
...
```

uROS and Custom Micro XRCE-DDS Agent

This section describes how to run the Micro-ROS Client demo and the custom XRCE-DDS RPMsg Agent.

1. (Optional) Connect the UART-to-TTL cable to **GPIO pin 40** on the RDK board to view log output from the CR8 core over UART channel 5 (P72–TXD5 / P73–RXD5).
2. Flash the `RZ/V2H RDK CR8 Core0 RPMsg Micro-ROS Demo` firmware to the target board.
3. On the board's terminal, run the CustomXRCEAgent with sudo privilege:

```
root@localhost:~# CustomXRCEAgent
```

Example output:

```
root@localhost:~# CustomXRCEAgent
[787] proc_id:0 rsc_id:0 mbx_id:1
```

(continues on next page)

(continued from previous page)

```
metal: warning:   metal_linux_irq_handling: Failed to set scheduler: -1.
...
```

4. On another terminal, use the following ROS 2 commands to verify communication:

```
source /opt/ros/jazzy/setup.bash
ros2 topic list
ros2 topic echo /cr8/heartbeat
```

Behavior:

- The CR8 firmware creates the topic `/cr8/heartbeat` and continuously publishes data to it.
- The custom Micro XRCE-DDS Agent makes this topic available in the ROS 2 environment running on the CA55 core.
- From the CA55 core, you can subscribe to and retrieve data from the `/cr8/heartbeat` topic.

Example output:

```
rz@localhost:~$ source /opt/ros/jazzy/setup.bash
rz@localhost:~$ ros2 topic list
/cr8/heartbeat
/parameter_events
/rosout
rz@localhost:~$ ros2 topic echo /cr8/heartbeat
data: 328
---
data: 329
---
data: 330
...
```

Troubleshooting

1. **Can't open the configuration.xml of CR8 e² studio project?**

   Confirm the RZ/V FSP version is 3.1 and import the **CM33 project** into the workspace and build it first, then try opening the CR8 project again.

2. **The behavior of the RPMsg demo is strange?**

   Restart the **RDK board** to reset the RPMsg endpoint.

3. **Unknown status of the micro-ROS demo?**

   Use a **USB-to-TTL** module to read logs from the UART interface of the RDK board (baud rate: **115200**).

   You should see output similar to the following:

```
[CR8] Start main_task_entry
[CR8] RPMsg endpoint ready
[CR8] Heartbeat publisher ready on /cr8/heartbeat
[CR8] Heartbeat #50 (uptime=16061 ms)
```

   You should run the `CustomXRCEAgent` only after the message `[CR8] RPMsg endpoint ready` appears on the UART log.

4. **Can't flash the firmware over J-Link?**

   Make sure you are using the correct **J-Link firmware version** and that **DIP switch SW1-6** is turned **ON**.

4 ROS2 Application Development with RZ/V2H RDK

This section covers the development of ROS2 applications using the Renesas RZ/V2H RDK board.

Target ROS2 Jazzy running on Ubuntu 24.04 LTS and RZ/V2H RDK.

When developing ROS2 applications for the Renesas RZ/V2H RDK platform, it is essential to set up the development environment correctly.

We provide a guide to help you get started with ROS2 development, including setting up the necessary tools, **cross-building applications**, and deploying them to the RZ/V2H RDK board.

The following topics are covered in this section:

- **Sample Applications:** Examples of ROS2 applications that can be built and executed on the RZ/V2H RDK.
- **Repositories and Packages:** Information about available RZ/V ROS2 packages.
- **ROS2 Development:** Instructions on the development workflow for ROS2 applications, including cross-build and deployment.
- **Other Concepts:** Additional concepts and advanced topics related to ROS2 development.
- **Appendix:** Supplementary information.

4.1 Sample Applications

This section introduces sample ROS2 applications developed for the Renesas RZ/V2H RDK platform, demonstrating various functionalities and use cases.

4.1.1 Vision Based Robotic Arm Teleoperation

Note

Available for *FoxGlove* and *MuJoCo Visualization* simulation environment without real robotic hardware!

Fig. 1: Arm Teleoperation Demo

Key features

The RZ/V Demo Arm Teleoperation package enables:

- Detect hand landmarks via camera input and control the arm and gripper for grasping tasks.
- Support for running two AI models simultaneously on the DRP-AI IP.
- Mapping of hand landmarks to robotic arm and hand joint commands.
- Control of AgileX Piper Arm (6 DOFs) with dexterous robotic hands (Inspire RH56).
- Simultaneous control of virtual and physical AgileX Piper Arm.
- Visualization through Foxglove Studio and MuJoCo.

RZ/V ROS2 Packages Used

- agilex_piper_arm_bringup
- agilex_piper_controller
- agilex_piper_utils
- agilex_piper_arm_description
- agilex_piper_ros2_control
- arm_hand_control
- cartesian_controllers
- rzv_model
- rzv_pose_estimation
- foxglove_keypoint_publisher

- rzv_playground

Optional: With Inspire RH56 hand support

- inspire_rh56_description
- inspire_rh56_dexhand
- inspire_rh56_hand_bringup
- inspire_rh56_hand_utils
- inspire_rh56_hand_ros2_control
- piper_arm_inspire_hand_bringup

RZ/V ROS2 Host PC Packages Used

Optional: Those packages are required on the **host PC** if you want to use MuJoCo simulation:

- agilex_piper_arm_description
- agilex_piper_mujoco
- cartesian_controllers
- mujoco
- mujoco_ros2_control
- mujoco_sim_ros2

Please install the ROS2 Jazzy on the host PC as per ROS2 Jazzy installation guide.

Quick Setup Instructions

1. Prepare the cross compiled ROS2 workspace with the required packages mentioned above.
   - Setup the RZ/V2H RDK board as per *RZ/V2H RDK board setup*.
   - Setup the host machine for cross-compilation as per *Common docker environment setup*.
   - Collect all *required packages* in the `ros2_ws/src/` directory inside the cross-compile docker container.
   - Cross-compile the ROS2 workspace using *cross-build the ROS2 Application using Yocto SDK*.
   - Deploy the `install` directory to the RZ/V2H RDK board using *Deploying the ROS2 Application* or using the `scp` command.
2. Install the required dependencies on the RZ/V2H RDK board.

```
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

3. **Optional**: Connect the AgileX Piper Arm and Inspire RH56 hand to the RZ/V2H RDK board if you want to control the real arm and hand.
4. Connect a compatible USB camera to the RZ/V2H RDK board for hand detection and landmark estimation.
   - The common setup is that the camera is fixed in one position and faces upward.
   - The USB camera's field of view should capture the user's hand, and the user's hand must remain within the camera's frame.
5. Launch the Vision Based Robotic Arm Teleoperation application.

- Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path/to>/install/setup.bash
```

- For real Agilex Piper Arm and Inspire RH56 Hand control, use:

```
$ ros2 launch rzv_playground hand_palm_pose_teleop_inspire_hand.launch.py use_
mock_hardware:=false
```

- For real Agilex Piper Arm with compatible Gripper, use:

```
$ ros2 launch rzv_playground hand_palm_pose_teleop_piper_gripper.launch.py
use_mock_hardware:=false
```

- For virtual hand control with FoxGlove (without real arm), use:

```
$ ros2 launch rzv_playground hand_palm_pose_teleop_inspire_hand.launch.py use_
mock_hardware:=true
```

- For virtual hand control with MuJoCo (without real arm), use:

```
$ ros2 launch rzv_playground hand_palm_pose_teleop_piper_gripper.launch.py \
   bringup_launch_file:=agilex_piper_mujoco_cartesian_control.launch.py
```

Make sure to check the correct CAN interface and serial port parameters in the launch files before running the above commands.

6. Visualize the robotic arm and hand movements by following the instructions below:

- Move your hand **up or down**, the Piper arm will move **up or down** accordingly.
- Move your hand **forward or backward**, the Piper arm will move **forward or backward**.
- Move your hand **left or right**, the Piper arm will move **left or right**.
- **Close your thumb**, the robotic hand or gripper will switch to the **grasping position**.
- If the system **cannot detect your hand** after a certain period, the Piper arm will **reset to its initial position**.

7. For simulation using Foxglove Studio, refer to the *FoxGlove Visualization* section for setup instructions.

The input layout file for FoxGlove Studio is located at: `rzv_playground/config/foxglove/*.json` inside the ROS2 workspace.

For MuJoCo simulation, refer to the *MuJoCo Visualization* section for setup instructions.

After setting up the MuJoCo environment, you can visualize the robotic arm and hand movements in the MuJoCo simulator on your host PC:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path/to>/install/setup.bash
$ ros2 launch agilex_piper_mujoco bringup_mujoco_cartesian_motion_controller.
launch.py
```

Make sure to set up the MuJoCo environment on your host PC as described in the *MuJoCo Visualization* section before running the above command.

Application Details

For more details about the Vision Based Robotic Arm Teleoperation application, refer to the README.md in rzv_playground package section.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the Vision Based Robotic Arm Teleoperation sample application.

4.1.2 Vision Based Dexterous Hand

> **Note**
>
> Available for *FoxGlove* simulation environment without real robotic hardware!

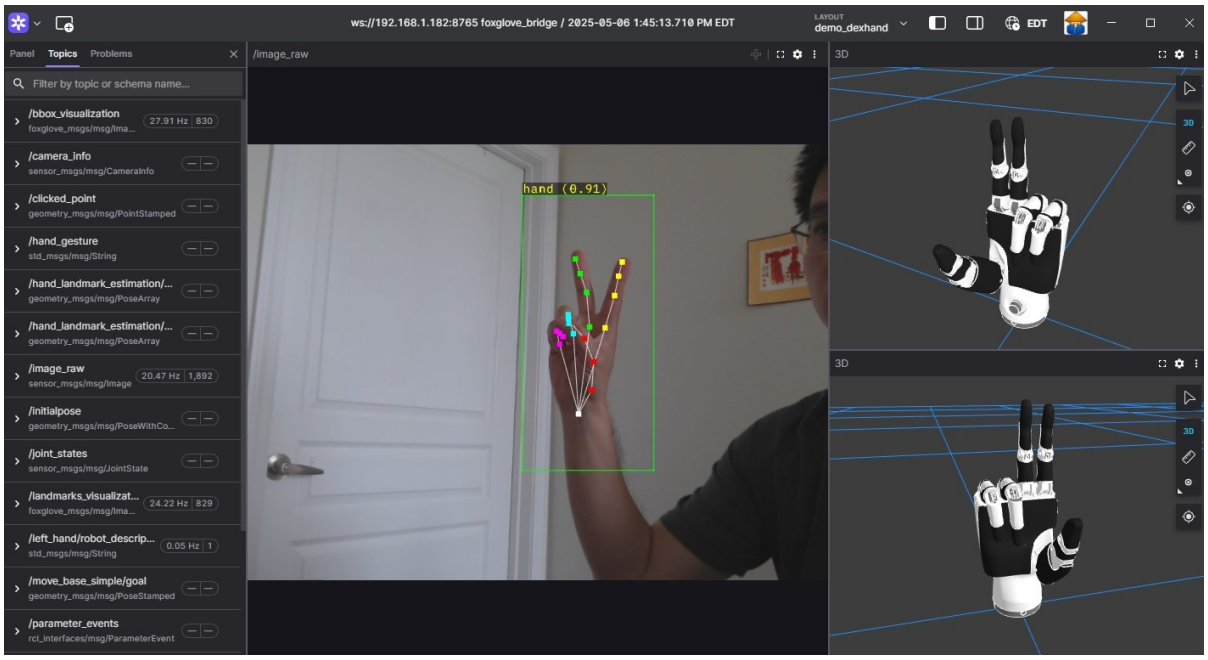

Fig. 2: Dexterous Hand Demo

Key features

The RZ/V Demo DexHand package enables:

- Hand landmark estimation and interpretation.
- Simultaneous control of virtual and physical dexterous hands.
- Visualization through Foxglove Studio.
- Support for multiple dexterous hand models.
- Support for running two AI models simultaneously on the DRP-AI IP: one for hand detection and another for hand landmark estimation.
- Support for multiple AI models for both hand detection and hand landmark estimation.

RZ/V ROS2 Packages Used

Base package

- arm_hand_control
- foxglove_keypoint_publisher
- rzv_demo_dexhand
- rzv_pose_estimation
- rzv_model

Option 1: Using Insprire RH6 hand

- inspire_rh56_urdf
- inspire_rh56_dexhand

Option 2: Using Ruiyan RH2 hand

- ruiyan_rh2_controller
- ruiyan_rh2_urdf
- ruiyan_rh2_dexhand

Quick Setup Instructions

1. Prepare the cross compiled ROS2 workspace with the required packages mentioned above.

- Setup the RZ/V2H RDK board as per *RZ/V2H RDK board setup*.
- Setup the host machine for cross-compilation as per *Common docker environment setup*.
- Collect all *required packages* in the `ros2_ws/src/` directory inside the cross-compile docker container.
- Cross-compile the ROS2 workspace using *cross-build the ROS2 Application using Yocto SDK*.
- Deploy the `install` directory to the RZ/V2H RDK board using *Deploying the ROS2 Application* or using the `scp` command.

Note

For cross-compilation by using Yocto SDK, please install `ruiyan_rh2_controller/rh6_ctrl/lib/libRyhandArm64.so` to the SDK sysroot using the following command:

```
$ sudo cp ruiyan_rh2_controller/rh6_ctrl/lib/libRyhandArm64.so <path_to_sdk_sysroot>/usr/lib/
```

2. Install the required dependencies on the RZ/V2H RDK board.

```
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

3. **Optional**: Connect the dexterous hand to the RZ/V2H RDK board if you want to control the real hand.

> **Note**
>
> Before using the RuiYan RH2 Dexhand, ensure that the hand is properly initialized using the provided setup script located in the `ruiyan_rh2_dexhand/setup/ruiyan_rh2_init.sh` or in the `install/ruiyan_rh2_dexhand/share/ruiyan_rh2_dexhand/setup/ruiyan_rh2_init.sh` after installation.

4. Connect a compatible USB camera to the RZ/V2H RDK board for hand detection and landmark estimation.
5. Launch the Vision Based Dexterous Hand application.

- Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path/to>/install/setup.bash
```

- For real dexterous hand control, use:

```
# For Inspire RH56 hand
$ ros2 launch rzv_demo_dexhand demo_physical_inspire_rh56_hand.launch.py

# For Ruiyan RH2 hand
$ ros2 launch rzv_demo_dexhand demo_physical_ruiyan_rh2_hand.launch.py
```

- For virtual hand control (without real dexterous hand), use:

```
# For Inspire RH56 hand
$ ros2 launch rzv_demo_dexhand demo_virtual_inspire_rh56_hands.launch.py

# For Ruiyan RH2 hand
$ ros2 launch rzv_demo_dexhand demo_virtual_ruiyan_rh2_hands.launch.py
```

6. Based on your hand gesture shown in front of the camera, the dexterous hand will mimic your hand movements.

> **Note**
>
> The common setup uses a fixed USB camera placed in front of the user and pointing **upward toward the hand**. The camera captures the palm from below, so that the **hand appears from bottom to top** in the image, the **wrist is at the bottom**, and the **fingers point upward**.
>
> When your hand is positioned correctly within the camera view, the **robot hand will mimic your gestures accurately**. The robot hand only interprets motion along the **vertical (bottom-to-top) direction**.
>
> Refer to the top image for the correct orientation between the camera and the user's hand.

7. For simulation using Foxglove Studio, refer to the *FoxGlove Visualization* section for setup instructions.

The input layout file for FoxGlove Studio is located at: `rzv_demo_dexhand/config/foxglove/demo_dexhand.json` inside the ROS2 workspace.

Application Details

For more details about the Vision Based Dexterous Hand application, refer to the README.md in rzv_demo_dexhand package section.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the Vision Based Dexterous Hand sample application.

4.1.3 Rock Paper Scissors

> **Note**
>
> Available for *FoxGlove* simulation environment without real robotic hardware!

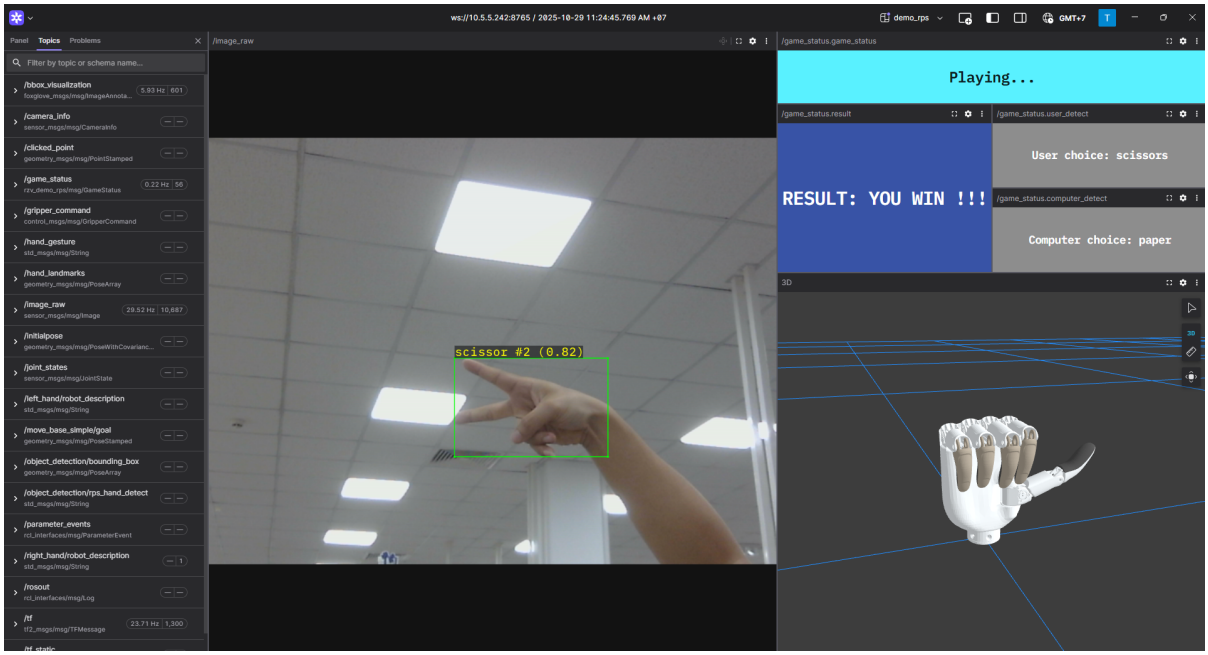

Fig. 3: Rock Paper Scissors Demo

Key features

The RZ/V Demo RPS (Rock Paper Scissors) package enables:

- Rock-Paper-Scissors Controller: Detects Rock - Paper - Scissors gestures in real time, executes the game logic, and sends commands to control the robotic hand accordingly.
- Compatible with the Inspire RH56 Dexhand and Ruiyan RH2 robotic hands.
- RPS Object detection and interpretation
- Simultaneous control of virtual and physical dexterous hands
- Visualization through Foxglove Studio

RZ/V ROS2 Packages Used

Base package

- arm_hand_control
- foxglove_keypoint_publisher

- rzv_object_detection
- rzv_demo_rps
- rzv_model

Option 1: Using Insprire RH6 hand

- inspire_rh56_urdf
- inspire_rh56_dexhand

Option 2: Using Ruiyan RH2 hand

- ruiyan_rh2_controller
- ruiyan_rh2_urdf
- ruiyan_rh2_dexhand

Quick Setup Instructions

1. Prepare the cross compiled ROS2 workspace with the required packages mentioned above.

- Setup the RZ/V2H RDK board as per *RZ/V2H RDK board setup*.
- Setup the host machine for cross-compilation as per *Common docker environment setup*.
- Collect all *required packages* in the `ros2_ws/src/` directory inside the cross-compile docker container.
- Cross-compile the ROS2 workspace using *cross-build the ROS2 Application using Yocto SDK*.
- Deploy the `install` directory to the RZ/V2H RDK board using *Deploying the ROS2 Application* or using the `scp` command.

Note

For cross-compilation by using Yocto SDK, please install `ruiyan_rh2_controller/rh6_ctrl/lib/libRyhandArm64.so` to the SDK sysroot using the following command:

```
$ sudo cp ruiyan_rh2_controller/rh6_ctrl/lib/libRyhandArm64.so <path_to_sdk_sysroot>/usr/lib/
```

2. Install the required dependencies on the RZ/V2H RDK board.

```
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

3. **Optional**: Connect the dexterous hand to the RZ/V2H RDK board if you want to control the real hand.

Note

Before using the RuiYan RH2 Dexhand, ensure that the hand is properly initialized using the provided setup script located in the `ruiyan_rh2_dexhand/setup/ruiyan_rh2_init.sh` or in the `install/ruiyan_rh2_dexhand/share/ruiyan_rh2_dexhand/setup/ruiyan_rh2_init.sh` after installation.

4. Connect a compatible USB camera to the RZ/V2H RDK board for Rock Paper Scissors hand gestures detection.
5. Rules of the application:

- Similar to the traditional game.
- The user initiates a game by showing the "HI" pose (scissors gesture) in front of the camera.
- The robotic hand performs a 1-2-3 countdown to signal the start of the round.
- When the countdown is finished, the player must show their chosen gesture (rock, paper, or scissors) within 2 seconds. If no gesture is detected in this time, the game is aborted.
- In case players give the choice, the robotic hand randomly selects and displays rock, paper, or scissors.
- After that, the game result is displayed by the robotic hand using the following gestures: OK - Draw, Thumbs Down - You lose, Victory - You win.
- Wait 2 seconds after the result is shown to start a new game.

6. Launch the Rock Paper Scissors application.

- Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path/to>/install/setup.bash
```

- For real dexterous hand control, use:

```
# For Inspire RH56 hand
$ ros2 launch rzv_demo_rps demo_physical_inspire_rh56_hand_rps.launch.py

# For Ruiyan RH2 hand
$ ros2 launch rzv_demo_rps demo_physical_ruiyan_rh2_hand_rps.launch.py
```

- For virtual hand control (without real dexterous hand), use:

```
# For Inspire RH56 hand
$ ros2 launch rzv_demo_rps demo_virtual_inspire_rh56_hand.launch.py

# For Ruiyan RH2 hand
$ ros2 launch rzv_demo_rps demo_virtual_ruiyan_rh2_hand.launch.py
```

7. For simulation using Foxglove Studio, refer to the *FoxGlove Visualization* section for setup instructions.

The input layout file for FoxGlove Studio is located at: `rzv_demo_rps/config/foxglove/demo_rps.json` inside the ROS2 workspace.

Application Details

For more details about the Rock Paper Scissors application, refer to the README.md in rzv_demo_rps package section.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the Rock Paper Scissors sample application.

4.1.4 Static Object Detection

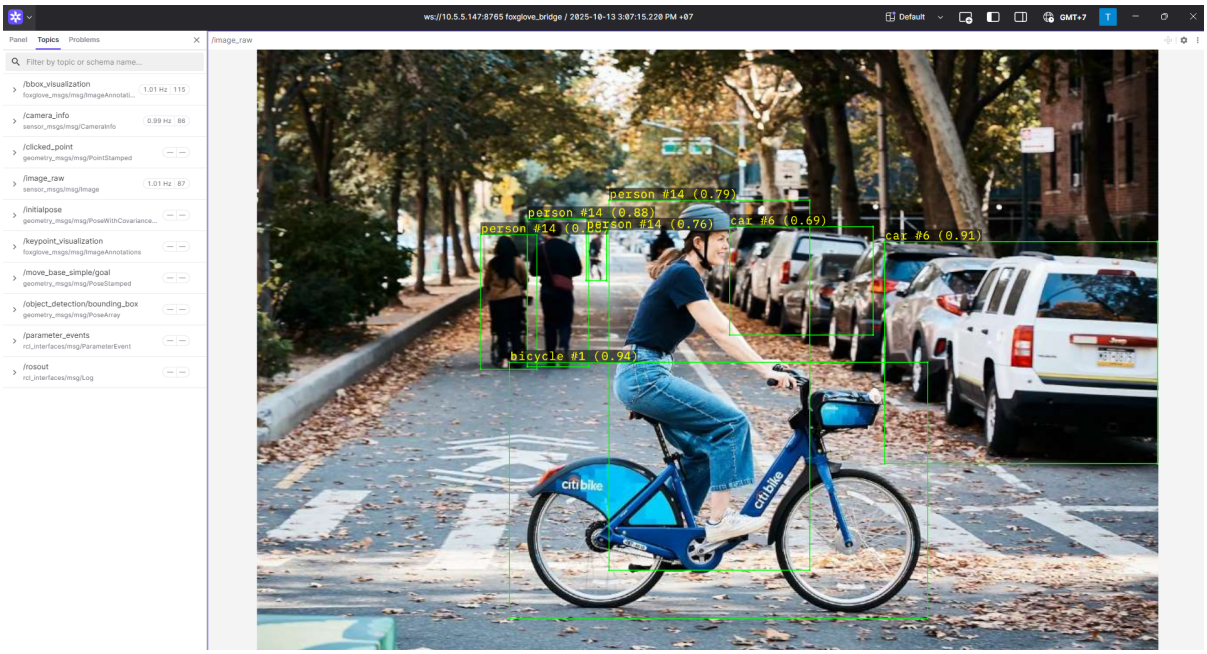

Fig. 4: Static Object Detection Demo

Key features

The RZ/V Static Object Detection package enables:

- Demonstrates the usage of AI library (DRP-AI) that is wrapped in ROS2 node.
- Support YOLOX Pascal VOC model (20 classes) for static object detection.
- Multi-threaded processing support.
- Visualization through Foxglove Studio.

RZ/V ROS2 Packages Used

- rzv_model
- rzv_object_detection
- foxglove_keypoint_publisher

Quick Setup Instructions

1. Prepare the cross compiled ROS2 workspace with the required packages mentioned above.

- Setup the RZ/V2H RDK board as per *RZ/V2H RDK board setup*.
- Setup the host machine for cross-compilation as per *Common docker environment setup*.
- Collect all *required packages* in the `ros2_ws/src/` directory inside the cross-compile docker container.
- Cross-compile the ROS2 workspace using *cross-build the ROS2 Application using Yocto SDK*.
- Deploy the `install` directory to the RZ/V2H RDK board using *Deploying the ROS2 Application* or using the `scp` command.

2. Install the required dependencies on the RZ/V2H RDK board.

```
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

3. Launch the Object Detection application.

- Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path/to>/install/setup.bash
```

- For static object detection, use:

```
# Pascal VOC object detection on static image
$ ros2 launch rzv_object_detection static_object_detection_yolox.launch.py

# Hand detection on static image using YOLOX
$ ros2 launch rzv_object_detection static_hand_detection_yolox.launch.py

# Hand detection on static image using Gold YOLOX
$ ros2 launch rzv_object_detection static_hand_detection_gold_yolox.launch.py
```

4. For visualization using Foxglove Studio, refer to the *FoxGlove Visualization* section for setup instructions.

The input layout file for FoxGlove Studio is located at: `rzv_object_detection/config/foxglove/objects_detection.json` inside the ROS2 workspace.

Application Details

For more details about the Static Object Detection application, refer to the README.md in rzv_object_detection package section.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the Static Object Detection sample application.

4.1.5 Static / Camera-based Hand Landmark Estimation

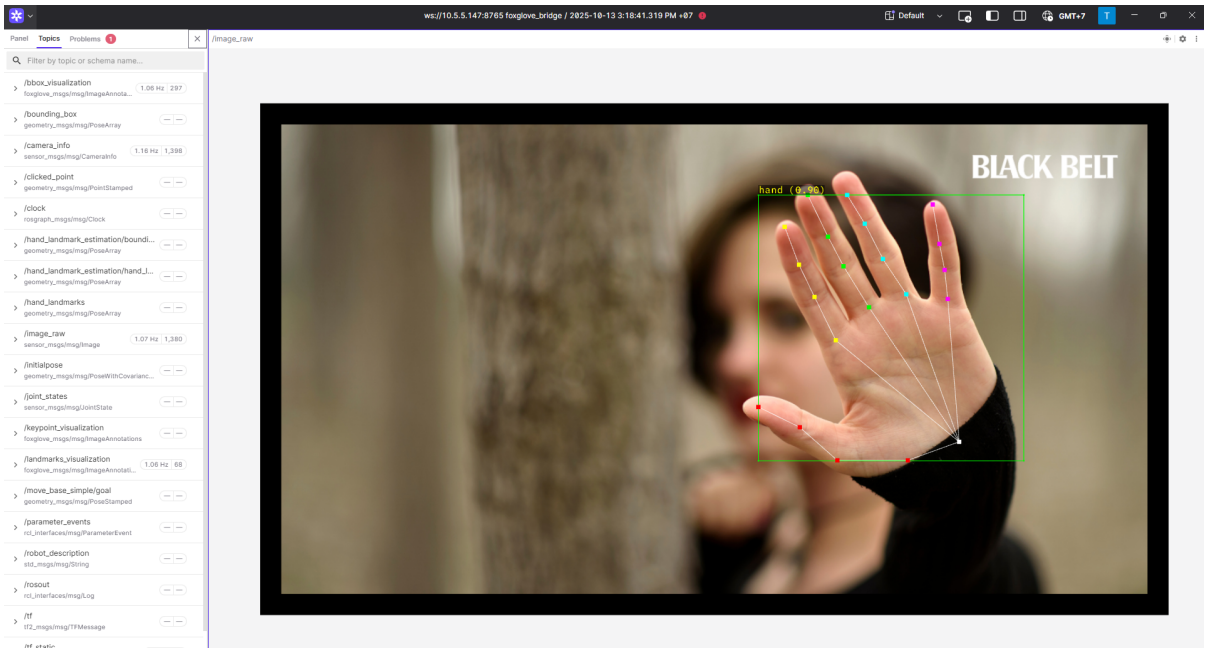

Fig. 5: Hand Landmark Demo

Key features

The RZ/V Pose Estimation package enables:

- Demonstrates the usage of AI library (DRP-AI) that is wrapped in ROS2 node.
- Hand detection and landmark estimation
- Real-time camera-based estimation
- Static image-based estimation
- Smooth landmark tracking
- Two-stage pipeline: Hand detection using YOLOX models -> Landmark estimation using various models
- Support for multiple landmark models:
    1. MediaPipe Hand Landmark model
    2. HRNetV2 Hand Landmark model
    3. RTMPose Hand model
- EMA-based landmark smoothing
- Integration with Foxglove Studio for visualization
- Multi-threaded processing support

RZ/V ROS2 Packages Used

- rzv_pose_estimation
- rzv_model
- foxglove_keypoint_publisher

Quick Setup Instructions

1. Prepare the cross compiled ROS2 workspace with the required packages mentioned above.

- Setup the RZ/V2H RDK board as per *RZ/V2H RDK board setup*.
- Setup the host machine for cross-compilation as per *Common docker environment setup*.
- Collect all *required packages* in the `ros2_ws/src/` directory inside the cross-compile docker container.
- Cross-compile the ROS2 workspace using *cross-build the ROS2 Application using Yocto SDK*.
- Deploy the `install` directory to the RZ/V2H RDK board using *Deploying the ROS2 Application* or using the `scp` command.

2. Install the required dependencies on the RZ/V2H RDK board.

```
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

3. **Optional:** Connect a compatible USB camera to the RZ/V2H RDK board for hand detection and landmark estimation.
4. Launch the Static / Camera-based Hand Landmark Estimation application.

- Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path/to>/install/setup.bash
```

- For hand landmark estimation on static image, use:

```
$ ros2 launch rzv_pose_estimation static_hand_landmark_estimation.launch.py
```

- For hand landmark estimation using camera input, use:

```
$ ros2 launch rzv_pose_estimation camera_hand_landmark_estimation.launch.py
```

5. For visualization using Foxglove Studio, refer to the *FoxGlove Visualization* section for setup instructions.

The input layout file for FoxGlove Studio is located at: `rzv_pose_estimation/config/foxglove/landmark_estimation.json` inside the ROS2 workspace.

Application Details

For more details about the Static / Camera-based Hand Landmark Estimation application, refer to the README.md in rzv_pose_estimation package section.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the Static / Camera-based Hand Landmark Estimation sample application.

4.2 Repositories and Packages

This section provides information about the repositories and packages used for ROS2 development on the Renesas RZ/V2H RDK platform.

4.2.1 Arm Hand Control

A ROS 2 package for controlling robotic hands through gesture recognition and landmark tracking.

Overview

This package provides nodes for controlling robotic hands, particularly the Inspire RH56 Dexhand. It supports:

- Hand gesture interpretation (predefined gestures)
- Hand landmark interpretation (from hand tracking algorithms)
- Direct control of the Inspire RH56 Dexhand hardware
- Pick-and-place operations via action server

For more details about the arm_hand_control package, refer to the README.md in the arm_hand_control package.

RZ/V ROS2 Package Dependencies

This package depends on the following RZ/V ROS2 packages:

- inspire_rh56_dexhand

For other dependencies, refer to the package.xml file.

License

This package is licensed under the MIT License.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the arm_hand_control package.

4.2.2 RZ/V Demo Dexhand

This package provides launch files and configurations for demonstrating dexterous hand control on Renesas RZ/V platforms.

It integrates vision-based pose estimation with both virtual and physical hand control.

Overview

This package provide launch files to run the dexterous hand demo using hand landmark estimation from camera input.

It supports controlling both virtual and physical dexterous hands, with visualization through Foxglove Studio.

For more details about the rzv_demo_dexhand package, refer to the README.md in the rzv_demo_dexhand package.

RZ/V ROS2 Package Dependencies

This package depends on the following RZ/V ROS2 packages:

- arm_hand_control
- foxglove_keypoint_publisher
- inspire_rh56_urdf
- rzv_pose_estimation
- ruiyan_rh2_urdf

For other dependencies, refer to the package.xml file.

License

This package is licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the rzv_demo_dexhand package.

4.2.3 RZ/V Demo RPS

This ROS 2 package enables hand using rock-paper-scissors gesture recognition.

It captures hand gestures through a vision-based recognition system and translates them into control commands to interact with games.

Beside that, also providing launch files and configurations for demonstrating Rock-Paper-Scissor on Renesas RZ/V platforms.

Overview

This package provides node for controlling robotic hands. It supports:

- Rock-Paper-Scissors Controller: Detects Rock Paper Scissors gestures in real time, executes the game logic, and sends commands to control the robotic hand accordingly.
- Compatible with the Inspire RH56 Dexhand and Ruiyan RH2 robotic hands.
- RPS Object detection and interpretation
- Simultaneous control of virtual and physical dexterous hands
- Visualization through Foxglove Studio

For more details about the rzv_demo_rps package, refer to the README.md in the rzv_demo_rps package.

RZ/V ROS2 Package Dependencies

This package depends on the following RZ/V ROS2 packages:

- arm_hand_control
- rzv_object_detection
- foxglove_keypoint_publisher
- inspire_rh56_urdf
- ruiyan_rh2_urdf

For other dependencies, refer to the package.xml file.

License

This package is licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the rzv_demo_rps package.

4.2.4 RZ/V Object Detection

A ROS2 package for performing object detection on Renesas RZ/V2H platform.

Implements various detection models and provides nodes for both static image and camera-based detection.

Overview

This package provides ROS2 nodes for:

- General object detection using YOLOX Pascal VOC model
- Hand detection using YOLOX and Gold YOLOX models
- Rock-Paper-Scissors hand detection using YOLOv8 models
- Real-time camera-based detection
- Static image-based detection

For more details about the rzv_object_detection package, refer to the README.md in the rzv_object_detection package.

RZ/V ROS2 Package Dependencies

This package depends on the following RZ/V ROS2 packages:

- foxglove_keypoint_publisher

For other dependencies, refer to the package.xml file.

License

This package is licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the rzv_object_detection package.

4.2.5 RZ/V Play Ground

Collection of demonstration and teleoperation launch files for robotic arm and hand systems on the RZ/V platform.

Overview

This package provides comprehensive launch files for various robotic systems, featuring:

- Hand pose teleoperation: Vision-based control using MediaPipe hand tracking
- Teleoperation twist control: Keyboard/game-pad control for both cartesian and joint space

- Multi-robot support: Agilex Piper Arm, SO ARM101, and various hand configurations
- Visualization: Foxglove Studio integration for real-time monitoring
- Safety features: Mock hardware mode for testing without physical robots

For more details about the rzv_playground package, refer to the README.md in the rzv_playground package.

RZ/V ROS2 Package Dependencies

This package depends on the following RZ/V ROS2 packages:

- rzv_object_detection
- rzv_pose_estimation
- foxglove_keypoint_publisher
- arm_hand_control
- agilex_piper_arm_bringup
- piper_arm_inspire_hand_bringup
- piper_arm_ruiyan_hand_bringup
- so_arm101_bringup

For other dependencies, refer to the package.xml file.

License

This package is licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the rzv_playground package.

4.2.6 RZ/V Pose Estimation

A ROS2 package for performing hand landmark estimation on Renesas RZ/V2H platform.

This package combines hand detection with landmark estimation to provide detailed hand pose analysis.

Overview

This package provides ROS2 nodes for:

- Hand detection and landmark estimation
- Support for multiple landmark estimation models
- Real-time camera-based estimation
- Static image-based estimation
- Smooth landmark tracking

For more details about the rzv_pose_estimation package, refer to the README.md in the rzv_pose_estimation package.

RZ/V ROS2 Package Dependencies

This package depends on the following RZ/V ROS2 packages:

- foxglove_keypoint_publisher

For other dependencies, refer to the package.xml file.

License

This package is licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the rzv_pose_estimation package.

4.2.7 Foxglove Keypoint Publisher

A ROS2 package that publishes keypoint poses (landmarks or bounding boxes) as Foxglove image annotations for visualization purposes.

Overview

This package provides a ROS2 node that:

- Subscribes to pose array messages containing key-points (like hand landmarks or bounding boxes)
- Converts these poses to Foxglove image annotations for visualization
- Supports configurable visualization parameters through YAML files
- Includes test configurations for:
  - Hand landmarks
  - Bounding boxes
  - Body poses

For more details about the foxglove_keypoint_publisher package, refer to the README.md in the foxglove_keypoint_publisher package.

RZ/V ROS2 Package Dependencies

There is no direct dependency on other RZ/V ROS2 packages.

For other dependencies, refer to the package.xml file.

License

This package is licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the foxglove_keypoint_publisher package.

4.2.8 Inspire RH56 Hand Packages

A collection of ROS 2 packages for controlling and interfacing with the Inspire RH56 robotic hand, including:

- inspire_rh56_dexhand: This package provides the hardware interface for controlling the Inspire RH56 Dexhand robotic hand.
- inspire_rh56_hand_bringup: ROS 2 package that provides launch files, controller configurations, robot descriptions, and test scripts for the Inspire RH56 6-DOF dexterous hand.
- inspire_rh56_hand_description: ROS2 package containing URDF description files for the Inspire RH56 dexterous hand.
- inspire_rh56_hand_ros2_control: ROS 2 package that provides a ros2_control hardware interface for the Inspire RH56 6-DOF dexterous hand.
- inspire_rh56_hand_utils: Utility nodes for Inspire RH56 Hand gripper action adapter with configurable joint mapping.
- inspire_rh56_urdf: URDF package for the Inspire RH56 dexterous hand.

Overview

This collection of packages provides comprehensive support for the Inspire RH56 robotic hand, including hardware interfaces, controllers, and robot descriptions. It enables users to easily set up and control the Inspire RH56 hand in ROS 2 environments.

For more details about each package, refer to their respective `README.md files` in the Inspire RH56 Hand packages.

License

Those packages are licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the collection of Inspire RH56 Hand packages.

4.2.9 Ruiyan RH2 Hand Packages

A collection of ROS 2 packages for controlling and interfacing with the Ruiyan RH2 robotic hand, including:

- ruiyan_rh2_controller: ROS2 package that provides the adaption of the controller for the RuiYan RH2 robotic hand for ARM64 target systems.
- ruiyan_rh2_dexhand: A ROS2 package that converts the `sensor_msgs/JointState` to `rh6_cmd/Rh6Cmd` as this message is required by the `rh6_ctrl` node for controlling the RuiYan RH2 Dexterous Hand.
- ruiyan_rh2_urdf: URDF package for the Ruiyan RH2 dexterous hand.

Note

For cross-compilation by using Yocto SDK, please install `ruiyan_rh2_controller/rh6_ctrl/lib/libRyhandArm64.so` to the SDK sysroot using the following command:

```
$ sudo cp ruiyan_rh2_controller/rh6_ctrl/lib/libRyhandArm64.so <path_to_
sdk_sysroot>/usr/lib/
```

Before using the RuiYan RH2 Dexhand, ensure that the hand is properly initialized using the provided setup script located in the `ruiyan_rh2_dexhand/setup/ruiyan_rh2_init.sh` or in the `install/ruiyan_rh2_dexhand/share/ruiyan_rh2_dexhand/setup/ruiyan_rh2_init.sh` after installation.

Overview

This collection of packages provides comprehensive support for the Ruiyan RH2 robotic hand, including hardware interfaces, controllers, and robot descriptions. It enables users to easily set up and control the Ruiyan RH2 hand in ROS 2 environments.

For more details about each package, refer to their respective `README.md files` in the Ruiyan RH2 Hand packages.

License

Those packages are licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the collection of Ruiyan RH2 Hand packages.

4.2.10 AgileX Piper Arm Packages

A collection of ROS 2 packages for controlling and interfacing with the AgileX Piper robotic arm, including:

- agilex_piper_arm_bringup: ROS 2 package that provides launch files, controller configurations, robot descriptions, and test scripts for the AgileX Piper arm and gripper.
- agilex_piper_arm_description: ROS2 package containing URDF description files for the Agilex Piper 6-DOF robotic arm.
- agilex_piper_controller: This package provides a C++ implementation of the controller for the Agilex Piper robotic arm.
- agilex_piper_ros2_control: ROS 2 package that provides a ros2_control hardware interface for the AgileX Piper arm and gripper.
- agilex_piper_utils: Utility nodes for Agilex Piper arm message conversion and pose transformations.
- agilex_piper_mujoco: MuJoCo simulation package for the Agilex Piper robotic arm with ros2_control integration.r5

Overview

This collection of packages provides comprehensive support for the AgileX Piper robotic arm, including hardware interfaces, controllers, and robot descriptions. It enables users to easily set up and control the Piper arm in ROS 2 environments.

For more details about each package, refer to their respective `README.md files` in the AgileX Piper Arm packages.

License

Those packages are licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the collection of AgileX Piper Arm packages.

4.2.11 SO-ARM101 Arm Packages

A collection of ROS 2 packages for controlling and interfacing with the SO-ARM101 robotic arm, including:

- so_arm101_bringup: ROS 2 package that provides launch files, controller configurations, robot descriptions, and test scripts for the SO ARM101 robotic arm.
- so_arm101_description: ROS2 package containing URDF description files for the SO-ARM101 robotic arm.
- so_arm101_ros2_control: This package provides ROS 2 control hardware interface for the SO ARM101 6-DOF robotic arm using STS3215 servo motors.
- so_arm101_utils: Utility nodes for SO-ARM101 arm message conversion and pose transformations.

Overview

This collection of packages provides comprehensive support for the SO-ARM101 Arm robotic hand, including hardware interfaces, controllers, and robot descriptions. It enables users to easily set up and control the SO-ARM101 Arm hand in ROS 2 environments.

For more details about each package, refer to their respective `README.md files` in the SO-ARM101 Arm Hand packages.

License

Those packages are licensed under the Apache License 2.0.

CHANGELOG

- v1.0.0 (2025-31-10): Initial release of the collection of SO-ARM101 Arm Hand packages.

4.3 ROS2 Application Development

This section covers the development of ROS2 applications on the Renesas RZ/V2H RDK platform, including setup, cross-build, and deploy the application.

4.3.1 Cross-build the ROS2 Application

This section provides instructions on how to cross-build ROS2 applications for the Renesas RZ/V2H RDK platform.

What is Cross-building?

Cross-building is the process of compiling software on a host system to run on a different target system.

The advantages of cross-building is that it allows developers to build applications for embedded systems without needing to compile directly on the target device, which have limited resources causing longer build times.

In this case, we will be cross-building ROS2 applications on a development machine (host) to run on the Renesas RZ/V2H RDK board (target).

We provide two methods for cross-building ROS2 applications for the RZ/V2H RDK platform:

- **Cross-build with Yocto SDK**: Uses the Yocto-provided SDK for cross-compiling ROS 2 applications targeting the RZ/V2H platform.

- **Cross-build with QEMU Docker**: Utilizes a Docker container with QEMU emulation to create a cross-compilation environment for building ROS2 applications.

Table 1: Comparison Between Build Methods

Method	Advantages	Disadvantages
Yocto SDK	Very fast compilation.	Possible **ABI version mismatch error**, if package versions differ between Yocto SDK and Ubuntu.
QEMU Docker	No **ABI mismatch**, as the container matches the target runtime.	**Slower build time** due to QEMU emulation.
Native build	Simplest setup, directly compiles on the RZ/V2H target.	**Very slow build time** and requires sufficient storage and memory on the board.

Follow the respective sections for detailed instructions on each cross-build method.

Tip

For most use cases, the **Cross-build with Yocto SDK** method is recommended for its speed and efficiency.

But if you encounter *ABI mismatch errors*, consider using the **Cross-build with QEMU Docker** method.

Cross-build with Yocto SDK

Caution

Be aware of potential **ABI mismatch errors** when using the Yocto SDK method.

Refer to the *ABI mismatch appendix* for more details.

To cross-build ROS2 applications for the RZ/V2H RDK platform, we will use the Yocto SDK provided with the RZ/V2H RDK Linux image.

Prerequisites

Complete the *Common docker environment setup* step as described in the **System Configuration** section.

After this step, the Docker container is expected to be fully set up with the Yocto SDK installed.

Access the Docker container terminal:

```
$ docker exec -it [name_of_docker_container] bash
```

Inside the container, go to your ROS2 workspace:

```
$ cd <path-to-your-ROS2-workspace>
```

Tip

`$ROS2_WS` is the default workspace directory set by the Dockerfile.

```
$ cd $ROS2_WS
```

Environment variables set in the Docker container:

- `$ROS2_WS`: Default ROS2 workspace directory.
- `$TOOLCHAINS_WS`: Directory for cross-compilation toolchain files.

Copy cross.cmake file to your workspace:

```
$ cp $TOOLCHAINS_WS/cross.cmake $ROS2_WS/
```

Note

On the first time create the container, the default toolchain files are already installed to the `$ROS2_WS`

Requirements

Before cross-building ROS 2 applications, ensure that the ROS 2 workspace is fully set up with all necessary dependency packages.

Refer to the *Common ROS2 Workspace structure* to gain some tips for setting up the ROS2 workspace for your development.

Make sure to include all required ROS2 packages in the `src/` directory of your workspace.

The workspace structure:

```
ros2_ws/
├── src/                    # Source code for ROS 2 packages
│   ├── package_1/
│   ├── package_2/
│   └── ...
├── build/                  # Build output directory (generated by cross-colcon-
build command)
├── install/                # Installation directory (generated by cross-colcon-
build command)
├── cross.cmake             # CMake toolchain file for cross-compilation
└── log/                    # Log files (generated by cross-colcon-build
command)
```

Cross-build Steps

Using `cross-colcon-build`

Use the `cross-colcon-build` command to build ROS 2 packages for the RZ/V2H RDK.

The `cross-colcon-build` script is a wrapper around the standard `colcon build` command that performs the following steps:

- Checks if a `cross.cmake` toolchain file exists in the current working directory.
- Unsets the `LD_LIBRARY_PATH` variable to avoid conflicts with the cross-compilation environment.
- Loads the cross-compilation environment by sourcing the appropriate SDK setup script.

- Runs `colcon build` with the necessary CMake arguments for cross-compilation, along with any user-provided arguments.

The `cross-colcon-build` command functions similarly to the standard `colcon build` command, but with key differences tailored for cross-compilation.

Instead of building binaries for your current host architecture, it targets the **RZ/V2H (arm64)** platform using the configured **Yocto SDK** and **toolchain**.

Usage Guide

Simply replace `colcon build` with `cross-colcon-build` in your workflow.

You can pass any additional arguments supported by `colcon build` directly to `cross-colcon-build`.

Example:

```
$ cd <path-to-your-ROS2-workspace>
$ cross-colcon-build --packages-select rzv_demo_dexhand
```

This command ensures that all necessary environment variables and toolchain settings are correctly applied for cross-compiling your ROS 2 packages for the **RZ/V2H** platform.

> **Note**
>
> For more details, refer to the `env.conf` file to understand exactly what `cross-colcon-build` is doing under the hood.
>
> Make sure your `ros2_ws` contain the `cross.cmake` file. To obtain the default `cross.cmake` file, copy it from the `$TOOLCHAIN_WS` directory.

Deployment

After successfully cross-building the ROS2 applications, deploy the built packages to the RZ/V2H RDK target device.

Copy the contents of the `install/` directory to the target device using the `scp` command or using the **ROS2: Deploy to Target** feature in VS Code workspace.

Refer to the *ROS2 Deployment* section for detailed instructions on deploying the applications to the target device.

After deployment, please install any additional dependencies on the target device using:

```
$ source /opt/ros/jazzy/setup.bash
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

Know issue

During cross-build, you will see:

```
WARNING:colcon.colcon_ros.prefix_path.ament:The path '/opt/poky/5.1.4/
sysroots/cortexa55-poky-linux/opt/ros/jazzy' in the environment variable
AMENT_PREFIX_PATH doesn't contain any 'local_setup.*' files.
```

The warning message during cross-build is expected and does not indicate a problem with your setup. It results from intentionally removing ROS2 setup files from the SDK sysroot to prevent hard code paths that would break execution on the target board.

As long as your application runs correctly on the RZ/V2H device, you can safely ignore this warning.

Without this, this error will be occur on the target board:

```
rz@localhost:/home/rz# source install/setup.bash
not found: "/opt/poky/5.1.4/sysroots/cortexa55-poky-linux/opt/ros/jazzy/local_
setup.bash"
rz@localhost:/home/rz#
```

Cross-build with Docker and QEMU

This method currently supports cross-compiling ROS2 applications for the RZ/V2H platform using Docker Buildx.

The provided Dockerfile and scripts have been tested with the RZ/V2H AI SDK and are intended for development and evaluation purposes.

Note

Supported: ROS2 Jazzy cross-compilation for arm64 (RZ/V2H) using the SDK and Docker Buildx.

Limitations: Only RZ/V2H (arm64) builds are supported. Additional ROS2 packages or custom dependencies may need to be installed manually via:

```
$ sudo apt install ros-<distro>-<package-name>
```

Prerequisites

1. Install Docker. Refer to the Docker Official Installation Guide.

   Ensure that Docker is installed on your host machine with Buildx and QEMU support for cross-platform builds.

2. Clone the *X Compilation QEMU Docker* repository to your Host PC.

   ```
   $ git clone https://partnergitlab.renesas.solutions/sst1/industrial/
   ws078/utility/x_compilation_qemu_docker.git
   ```

3. RZ/V2H RDK Yocto SDK installer.

   Obtain the installer:

   ```
   poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.
   1.4.sh
   ```

4. Copy the Yocto SDK installer to the Docker build context directory. Please replace the paths below with your actual file locations.

   ```
   $ cp poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-
   toolchain-5.1.4.sh ~/x_compilation_qemu_docker/
   ```

After completing the prerequisites, the project structure should be:

```
x_compilation_qemu_docker/
├── Dockerfile
```

(continues on next page)

(continued from previous page)

```
├── setup_docker_builx.sh
├── poky-glibc-x86_64-renesas-core-image-weston-cortexa55-rz-cmn-toolchain-5.
1.4.sh
└── README.md
```

Build the Docker Image

Run `./setup_docker_builx.sh <path_to_ros2_workspace> [name_of_docker_container]` to set up the environment:

```
$ cd x_compilation_qemu_docker/
$ ./setup_docker_builx.sh <path_to_ros2_workspace> [name_of_docker_container]
```

- <path_to_ros2_workspace>: Path to your ROS 2 workspace on your machine. It will be mounted inside the Docker container.
- [name_of_docker_container]: (Optional) Name for the Docker container. Default is *rzv2h_ros_sdk*.

To start working inside the Docker container, use:

```
$ docker exec -it [name_of_docker_container] bash
```

Requirements

Before cross-building ROS 2 applications, ensure that the ROS 2 workspace is fully set up with all necessary dependency packages.

Refer to the *Common ROS2 Workspace structure* to gain some tips for setting up the ROS2 workspace for your development.

Make sure to include all required ROS2 packages in the `src/` directory of your workspace.

The workspace structure:

```
ros2_ws/
├── src/                  # Source code for ROS 2 packages
│   ├── package_1/
│   ├── package_2/
│   └── ...
├── build/                # Build output directory (generated by colcon build
command)
├── install/              # Installation directory (generated by colcon build
command)
└── log/                  # Log files (generated by colcon build command)
```

Cross-build Steps

Inside the Docker container, navigate to your ROS2 workspace:

```
$ cd <path-to-your-ROS2-workspace>
```

Build the ROS2 packages using the `colcon build` command:

```
$ colcon build
```

See also

For more details on colcon to build the ROS2 applications, refer to the Using colcon to build packages.

Deployment

After successfully cross-building the ROS2 applications, deploy the built packages to the RZ/V2H RDK target device by copying the contents of the `install/` directory to the target.

```
$ scp -r <path/to>/ros2_ws/install/* rz@<RZ/V2H-RDK-IP-address>:/path/to/
deployment/directory/
```

Install any additional dependencies on the target device using:

```
$ souce /opt/ros/jazzy/setup.bash
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

4.3.2 Useful tool for development of ROS2 applications

This section provides instructions on deploying, debugging ROS2 applications to the Renesas RZ/V2H RDK platform.

Attention

This method support the cross-build method using Yocto SDK only.

Prerequisites

1. Complete the *Common docker environment setup* step as described in the **System Configuration** section.
2. Also, complete the *Cross-build the ROS2 Application using Yocto SDK* step as described in the **ROS2 Development** section.
3. The following structure is expected after cross-building the ROS2 application:

```
ros2_ws/
├── build
├── .clang-format   # Clang format configuration file
├── cross.cmake
├── install         # Installation directory (generated by cross-colcon-build
command)
├── log
├── src
└── .vscode         # VS Code workspace configuration files
```

In case your workspace does not contain the `.vscode/` directory, you can copy it manually from the `$TOOLCHAIN_WS` directory.

4. Ensure your development machine has the following:
    - Visual Studio Code (VS Code)

- Remote Development extension for VS Code
- C/C++ extension installed inside the development container

Attention

- Your **development machine** can establish an SSH connection to the target board.
- Your **target board** has both gdbserver and rsync installed. If not, you can install it using:

```
$ sudo apt install gdbserver rsync
```

VS Code Workspace Configuration

With the VS Code workspace configuration, you can easily deploy and remote debug your ROS2 applications on the RZ/V2H RDK platform.

VS Code configuration files are located in the `.vscode/` directory of your ROS2 workspace:

```
.vscode/
├── c_cpp_properties.json          # C/C++ extension configuration file
├── deploy.sh                      # Deployment script to copy files to
target
├── launch.json                    # Debugger launch configuration file
├── run_program.sh                 # Script to run the program on target
├── settings.json                  # VS Code workspace settings file
├── settings.linux.json            # Linux-specific settings overrides
├── start_target_gdbserver.sh      # Script to start gdbserver on target
└── tasks.json                     # Task runner configuration file
```

By default, the configuration files are set up by the Dockerfile during the Docker container creation.

Supported Features

The VS Code workspace configuration supports the following features:

Target for ROS2 Jazzy Remote Run/Debug for arm64 (RZ/V2H) with cross-build using Yocto SDK method.

VS Code Tasks

- **ROS2: SSH to Target** - Start an SSH session to the target board.
- **ROS2: Debug Run (GDB)** - Start remote debugging a ROS2 node using the `ros2 run` command with the GDB server.
- **ROS2: Debug Launch (GDB)** - Start remote debugging a ROS2 node within a launch file using the `ros2 launch` command with the GDB server.
- **ROS2: Run Package Executable** - Run a ROS2 node using the `ros2 run` command.
- **ROS2: Launch Package LaunchFile** - Run ROS2 nodes using the `ros2 launch` command.
- **ROS2: Deploy to Target** - Deploy the `install` folder to the target board.
- **ROS2: Build Debug** - Build the workspace with the *Debug* configuration.
- **ROS2: Build Release** - Build the workspace with the *Release* configuration.
- **ROS2: Clean All** - Clean all build artifacts (`build`, `install`, `log` folders).

VS Code Launch Configurations

- **GDB for ROS2 Run** - Configure debugging in VS Code compatible with ROS2 *Run* debugging using GDB (`ros2 run`).
- **GDB for ROS2 Launch** - Configure debugging in VS Code compatible with ROS2 *Launch* debugging using GDB (`ros2 launch`).

> **Note**
>
> All tasks support running with custom arguments.
>
> A prompt will be displayed for argument input on each execution.

> **Tip**
>
> You can use the hot key `Ctrl+Shift+P` in VS Code and search for **Tasks: Run Tasks** to execute the tasks listed above.

Limitations:

- Only RZ/V2H (arm64) builds are supported.
- Only support debug one ROS node at a time (both ros2 run and ros2 launch).
- The speed can be slow if the install folder is large or the network connection is poor.

Prepare for Deployment

1. Use the `Remote Development` plugin for VS Code to start working inside the development Docker container.
2. Prepare the workspace following the *VS Code Workspace Configuration*.
3. Modify the contents of `c_cpp_properties.json` in your workspace. For example, modify the `includePath` or related configuration in `settings.json` to match your workspace setup. This file is provided as a reference example only.
4. Open the `settings.json` file, then edit the following variables to match your development environment:

> **Attention**
>
> The variables listed below are essential for the remote debugging and deployment workflow.

Table 2: Debug Configuration Variables

Variable Name	Required	Description	`ros2 run` required	`ros2 launch` required
`HOST_GDB_PATH`	Optional	Path to the GDB executable on the host machine.		
`TARGET_LOCAL_SYSROOT`	Optional	Path to the sysroot of the target device SDK (used for debugging).		
`TARGET_IP`	**Yes**	IP address of the target device.	Yes	Yes
`TARGET_GDB_PORT`	Optional	GDB server port used on the target device.		
`TARGET_USER`	Optional	Username used to connect to the target device.		
`TARGET_PASSWORD`	Optional	Password used to authenticate the target connection.		
`TARGET_ROS2_WS`	Optional	Target ROS2 workspace path (used for deployment).		
`NODE_PACKAGE_NAME`	**Yes**	The ROS 2 package name that contains the executable.	Yes	For debug only
`NODE_EXECUTABLE_NAME`	**Yes**	The name of the executable to be launched.	Yes	For debug only
`LAUNCH_PACKAGE_NAME`	**Yes**	The ROS 2 package name that contains the launch file.	No	Yes
`LAUNCH_FILE_NAME`	**Yes**	The launch file used to start the ROS 2 application.	No	Yes

Note

- Variables marked as *Optional* should be modified only if your development environment differs from the default one generated by the `Dockerfile`.
- Variables marked as **Required** must be set correctly each time you run a remote debug session; otherwise, the workflow will not operate properly.

When using this configuration, it will mimic the following commands on the target board:

```
ros2 run <NODE_PACKAGE_NAME> <NODE_EXECUTABLE_NAME>

ros2 launch <LAUNCH_PACKAGE_NAME> <LAUNCH_FILE_NAME>
```

Start Deployment

Run the **ROS2: Deploy to Target** task in VS Code to deploy the `install` folder to the target board.

After deployment, please install any additional dependencies on the target device using:

```
$ source /opt/ros/jazzy/setup.bash
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Please replace `<path/to>install/` with the actual path to the `install/` directory on your RZ/V2H RDK board.

Start Running / Remote Debugging

1. Update the variables `NODE_PACKAGE_NAME`, `NODE_EXECUTABLE_NAME`, `LAUNCH_PACKAGE_NAME` and `LAUNCH_FILE_NAME` in the `settings.json` file to match the node you want to debug.
2. Build your workspace using the *Debug* build type (required for debugging):

```
cross-colcon-build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

3. Use **Run Task** or **Launch** to start your workflow.
4. Enter any custom arguments if prompted.
5. Begin debugging or running the application.

Example: Workflow

Debugging `foxglove_keypoint_publisher_node` in `demo_virtual_inspire_rh56_hands.launch.py` from the RZ/V ROS Package

- In `settings.json`, set the following variables:

```
{
  "NODE_PACKAGE_NAME": "foxglove_keypoint_publisher",
  "NODE_EXECUTABLE_NAME": "foxglove_keypoint_publisher_node",
  "LAUNCH_PACKAGE_NAME": "rzv_demo_dexhand",
  "LAUNCH_FILE_NAME": "demo_virtual_inspire_rh56_hands.launch.py"
}
```

- Build the workspace with *Debug* configuration using the VS Code task **ROS2: Build Debug** or the command:

```
cross-colcon-build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

- In VS Code, press `Ctrl+Shift+D` and select **GDB for ROS2 Launch**.
- Enter the custom arguments when prompted:

```
video_device:=/dev/video0
```

- Press `Enter` to start the debugging session. The following command will be executed on the target board:

```
$ ros2 launch --launch-prefix 'gdbserver localhost:2345' --launch-prefix-
filter 'foxglove_keypoint_publisher_node' rzv_demo_dexhand demo_virtual_
inspire_rh56_hands.launch.py
```

Note

When debugging in `ros2 launch` mode:

1. Set `NODE_PACKAGE_NAME` to the package containing the target executable.
2. Set `NODE_EXECUTABLE_NAME` to the name of the executable to be debugged.

Troubleshooting

1. Can't deploy / remote run / remote debug:
   - Check the target IP, username, and password in the `settings.json` file.
   - Ensure the target board is powered on and connected to the network.
   - Verify that `gdbserver` and `rsync` are installed on the target board.
   - Establish the first SSH connection to the target board using the `ssh command` to accept the host key.
   - Delete the known hosts entry for the target IP in the `~/.ssh/known_hosts` file if the target board's host key has changed.
2. Remote debug fails to start or disconnects immediately:
   - Ensure the workspace is built with the *Debug* configuration.
   - Verify that the correct package and executable names are set in the `settings.json` file.
   - Check network stability between the host and target board.
   - Make sure the GDB server port is not blocked by a firewall or not used by other process.

4.4 Other Concepts

High-level overview of important concepts relevant to RZ/V2H RDK ROS2 development.

4.4.1 Foxglove Visualization

Foxglove Studio is a popular framework to visualize ROS running applications similar to Rviz.

It provides a web-based visualization client and a desktop app on both Window and Ubuntu machine.

Foxglove Setup

Use the following steps to visualize a ROS application in Foxglove.

1. Install Foxglove Studio on your Host PC

   Download the Foxglove Studio package to install Foxglove Studio on your Host PC.
2. Start Foxglove Studio on your Host PC.
3. Create a new connection by clicking on the "Open connection".

   Select "FoxGlove WebSocket" as the connection type.
4. In the WebSocket URL field, enter the following URL to connect to a ROS2 application running on the RZ/V2H RDK board:

   ```
   ws://<board-IP>:8765
   ```

   Replace <board-IP> with the actual IP address of your RZ/V2H RDK board.

   Make sure that the RZ/V2H RDK board is running a ROS2 application with the Foxglove WebSocket server node active, and your machine can reach the board over the network.

   > **Tip**
   >
   > By default, the Foxglove Bridge runs on the RZ/V2H RDK.

It can also be deployed to any ROS2 device on the same network. For example a laptop or another embedded device.

Keep in mind that running the Foxglove Bridge consumes some CPU resources on the device, so it is recommended to monitor system load when enabling it alongside other ROS2 nodes.

5. Click on the "Open" button to establish the connection.
6. Once connected, you can add various visualization panels to display data from your ROS2 application.
7. To add a panel, click on the "+" button in the top-right corner of the Foxglove Studio window and select the desired panel type (e.g., 3D View, Image, Plot, etc.).

Hint

Some tutorials also provide a pre-configured Foxglove layout in the form of a JSON file, which is typically located under `package_name/config/foxglove`.

To use those make sure you have the repository containing the tutorial cloned on your visualization machine.

Import the layout file by clicking on the **Import from file...** in the layout drop-down menu and choosing the layout file from the correct folder.

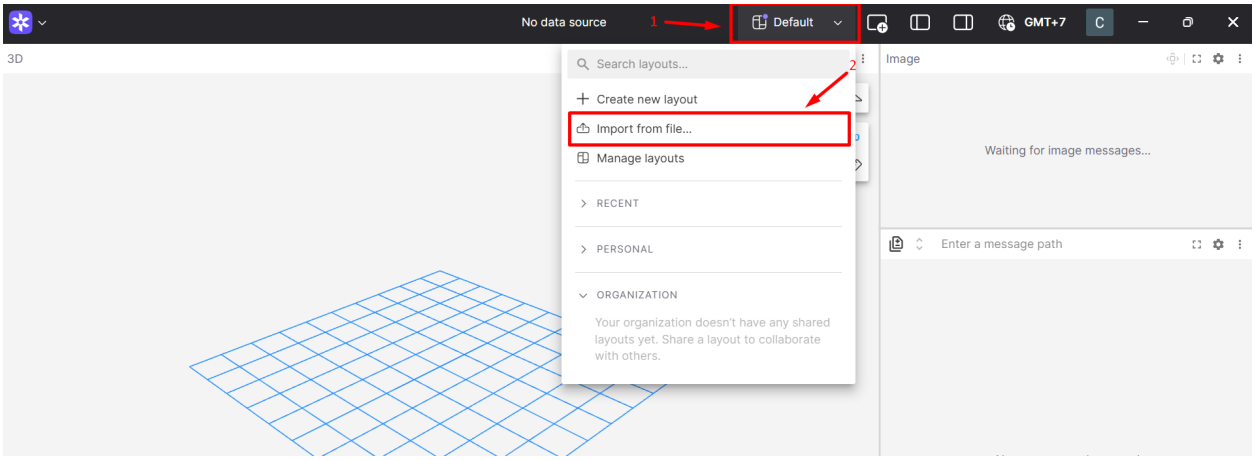

Fig. 6: Foxglove Import Layout

4.4.2 MuJoCo Visualization

MuJoCo Visualization is a simulation platform designed for robotics development, particularly in the context of ROS2.

It provides a virtual environment where developers can test and validate their robotic applications without the need for physical hardware.

MuJoCo Setup

Use the following steps to visualize a ROS application in MuJoCo.

1. Install the ROS2 Jazzy on the host PC as per ROS2 Jazzy installation guide.
2. Install MuJoCo and the necessary ROS2 MuJoCo packages on your Host PC.

The required packages are, please install them into your **Host PC's ROS2 Jazzy workspace**:

- mujoco_sim_ros2
- mujoco_ros2_control
- mujoco

Additional, some other ROS2 packages are also required on the **Host PC's ROS2 Jazzy workspace** in order to run the MuJoCo simulation for the RZ/V Demo Arm Teleoperation application:

- agilex_piper_arm_description
- agilex_piper_mujoco
- cartesian_controllers

For other applications, please refer to the respective application documentation for any additional packages that may be required.

Those packages can be cloned into the `src/` directory of your *ROS2 Jazzy workspace* on the Host PC.

3. Build the ROS2 workspace on the Host PC.

```
$ cd <path-to-your-ROS2-Jazzy-workspace>
$ rosdep update
$ rosdep install --from-paths src --ignore-src -r -y
$ colcon build --symlink-install
```

4. Load the workspace environment:

```
$ source /opt/ros/jazzy/setup.bash
$ source <path-to-your-ROS2-Jazzy-workspace>/install/setup.bash
```

5. Launch the MuJoCo simulation, for example the RZ/V Demo Arm Teleoperation application.

```
$ ros2 launch agilex_piper_mujoco bringup_mujoco_cartesian_motion_controller.
launch.py
```

The MuJoCo simulator window should open, displaying the robotic arm like below:

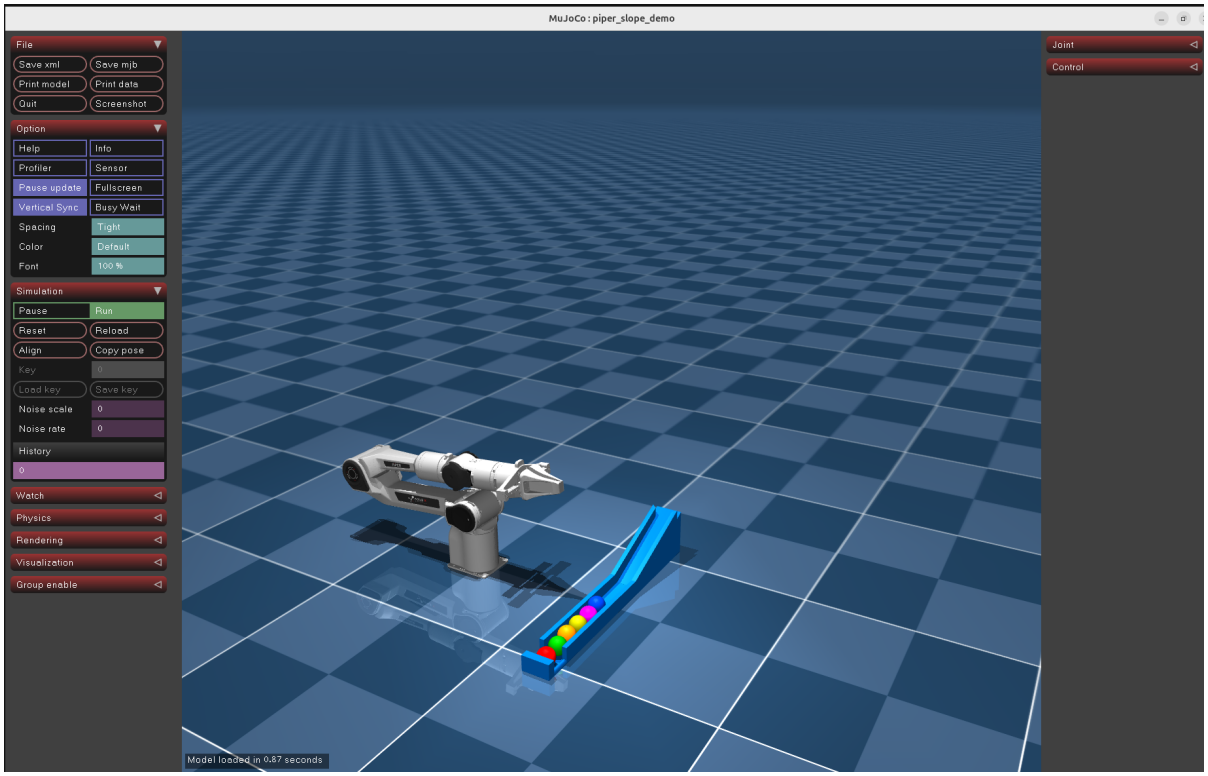

Fig. 7: MuJoCo Simulation of RZ/V Demo Arm Teleoperation Application

4.4.3 Intel RealSense Camera Installation

This section provides instructions for installing and configuring Intel RealSense cameras on the Renesas RZ/V2H RDK board for ROS2 applications.

Why should we use cross-build instead of native-build?

While we can build *librealsense2* and *realsense-ros* native, building *librealsense2* this way takes a significant amount of time. By using cross-build, we can reduce both the build time and the deployment time to the target machine.

However, cross-build has limitations when it comes to installing or uninstalling related software. Please follow this guide carefully to learn how to use it correctly.

For more detailed information, please refer to the official RealSense ROS and librealsense documentation.

For more information about building native, please refer to the Install librealsense2 documentation

Start from the **Install librealsense2** step and skip **3. Build and apply patched kernel modules for**, as we have already applied these patches in the kernel image by do the next step.

Prerequisites

Before proceeding:

- Ensure that you have a working cross-compilation environment set up as described in the *Common docker environment setup* section.
- Ensure that you have set up the eSDK environment as described in the *eSDK Setup* section.
- Installing the required dependencies for building *librealsense2* inside the cross-compilation Docker container:

```
$ sudo apt-get install -y ccache
```

Patch the Kernel for Intel RealSense Support

To enable support for Intel RealSense cameras, you need to patch the kernel with the necessary drivers. Follow these steps on the host machine where the Yocto eSDK is set up:

1. Download the following RealSense kernel patch files from **Common Utils** repository:
   - 0001-media-uvcvideo-Add-support-for-Realsense-camera-form.patch
   - intel_realsense.cfg
2. Complete the **Step 1 and 2: Applying Patches for linux-yocto** in the *Custom Linux Kernel and Device Tree* section.
3. Copy the `intel_realsense.cfg` and `0001-media-uvcvideo-Add-support-for-Realsense-camera-form.patch` files to the *~/poky_sdk/workspace/appends/linux-yocto/* directory.
4. Edit the *~/poky_sdk/workspace/appends/linux-yocto_6.10.bbappend* file to include the new configuration fragment at the end of file:

   ```
   renesas@docker-pc:~$ vi ~/poky_sdk/workspace/appends/linux-yocto_6.10.
   bbappend
   ```

   - Add the following line to the bbappend file:

   ```
   SRC_URI:append = "file://intel_realsense.cfg \
              file://0001-media-uvcvideo-Add-support-for-Realsense-camera-
   form.patch"
   ```

5. **Skip step 3 and 4**, continue build the kernel and deploy it by following **Step 5: Build the Modified Kernel** in the *Custom Linux Kernel and Device Tree* section.

Cross-compile and Install librealsense2

Build the *librealsense2* library inside the cross-compilation Docker container:

1. Start the cross-compilation Docker container as described in the *Common docker environment setup* section.
2. Inside the Docker container, clone the *librealsense* repository:

   ```
   $ git clone https://partnergitlab.renesas.solutions/sst1/industrial/
   ws078/rzv_ros_package/intel-realsense-camera/librealsense.git --depth 1 -
   -single-branch -b rzv_ros2
   ```

3. Navigate to the *librealsense* directory:

   ```
   $ cd librealsense
   ```

4. Create a build directory and navigate into it:

   ```
   $ mkdir build && cd build
   ```

5. Configure the build using CMake with the appropriate toolchain file:

   ```
   $ cmake .. -DCMAKE_TOOLCHAIN_FILE=~/toolchains/cross.cmake \
              -DBUILD_EXAMPLES=OFF \
              -DBUILD_GRAPHICAL_EXAMPLES=OFF \
   ```

(continues on next page)

(continued from previous page)

```
            -DBUILD_UNIT_TESTS=OFF \
            -DCMAKE_BUILD_TYPE=Release \
            -DCMAKE_MAKE_PROGRAM=/usr/bin/make \
            -DENABLE_PRECOMPILED_HEADERS=OFF \
            -DCMAKE_INSTALL_PREFIX=./tmp-install
```

6. Build and install the library:

```
$ make -j$(nproc)
$ sudo make install
```

7. Copy the installed files to the Yocto SDK sysroot:

```
$ sudo cp -r ./tmp-install/* $ROS2_SDK_SYSROOT/usr/
```

8. Deploy the *librealsense2* library to the RZ/V2H RDK board:

   Compress the *tmp-install* directory:

```
$ sudo tar -cf librealsense.tar.bz2 -C ./tmp-install/ .
```

   Copy the `librealsense.tar.bz2` file to the RZ/V2H RDK board using *scp* or any other file transfer method.

   On the RZ/V2H RDK board, extract the files to the root directory:

```
$ mkdir -p ~/librealsense_install
$ tar -xvf librealsense.tar.bz2 -C ~/librealsense_install/
$ sudo cp -r ~/librealsense_install/* /usr/
```

Setup udev rules for Intel RealSense Cameras

Install rules so the device is accessible without root:

```
$ git clone https://partnergitlab.renesas.solutions/sst1/industrial/ws078/rzv_
ros_package/intel-realsense-camera/librealsense.git --depth 1 --single-branch
-b rzv_ros2

$ cd librealsense

# Running the udev rules setup script
$ sudo ./scripts/setup_udev_rules.sh
```

Verify device on the target board

Connect the Intel RealSense camera to the RZ/V2H RDK board via USB before running the following commands.

```
# Check connected RealSense devices
$ rs-enumerate-devices
# or
$ rs-fw-update -l   # list devices & firmware info
```

If nothing appears, recheck cables/ports (prefer USB 3.x), udev rules, and power.

Cross-compile and Install realsense-ros

Build the *realsense-ros* package inside the cross-compilation Docker container:

1. Start the cross-compilation Docker container as described in the *Common docker environment setup* section.
2. Inside the Docker container, clone the *realsense-ros* repository into your ROS2 workspace *src* directory:

```
$ cd ~/ros2_ws/src
$ git clone https://partnergitlab.renesas.solutions/sst1/industrial/ws078/rzv_
ros_package/intel-realsense-camera/realsense_ros.git --depth 1 --single-branch
-b rzv_ros2
```

3. Navigate to your ROS2 workspace directory and cross-compile the workspace using the Yocto SDK as described in the *cross-build the ROS2 Application using Yocto SDK* section.
4. Deploy the *install* directory to the RZ/V2H RDK board using *Deploying the ROS2 Application* or using the *scp* command.
5. Install the required dependencies on the RZ/V2H RDK board:

```
$ rosdep install --from-paths <path/to>install/*/share -y -r --ignore-src
```

Warning (safe and can ignore):

The following warning will be displayed when running the above *cross-colcon-build* command. This occurs because, during cross-compilation, the build process tries to detect the camera using the *rs-enumerate-devices* command.

We have already verified this command in a previous step and confirmed that it works.

```
--- stderr: realsense2_camera
bash: line 1: rs-enumerate-devices: command not found
---
```

(Optional) Compressed Image Transport

If you want to publish or subscribe to compressed image topics (e.g., for bandwidth optimization), please install the following packages before running the RealSense node.

On the target board:

```
sudo apt install ros-$ROS_DISTRO-image-transport
sudo apt install ros-$ROS_DISTRO-compressed-image-transport
```

Running the Camera Node

```
# Setup environment
source install/setup.bash
```

Launch Directly

```
# Default launch
ros2 run realsense2_camera realsense2_camera_node

# Launch with pointcloud enabled
ros2 run realsense2_camera realsense2_camera_node --ros-args -p pointcloud__
neon_.enable:=true
```

Launch via `rs_launch.py`

```
# Default launch
ros2 launch realsense2_camera rs_launch.py

# Pointcloud
ros2 launch realsense2_camera rs_launch.py pointcloud__neon_.enable:=true

# With YAML
ros2 launch realsense2_camera rs_launch.py config_file:=realsense_config.yaml
```

Example `realsense_config.yaml`

```
# Enable synchronization between multiple streams (e.g., depth + color)
enable_sync: true

# Enable depth-to-color alignment (depth image pixels align with the color
image)
align_depth:
  enable: true

# Perform an initial hardware reset when starting the camera node
initial_reset: true

depth_module:
  # Depth stream resolution and frame rate: width x height x fps
  depth_profile: 640x480x30
  # Laser emitter power level (range depends on camera model)
  laser_power: 360.0

rgb_camera:
  # Color stream resolution and frame rate: width x height x fps
  color_profile: 640x480x30

pointcloud__neon_:
  # Enable NEON (ARM SIMD) optimized point cloud processing
  enable: false

  # Output point cloud in ordered (2D grid) format
  # true  -> Keep pixel-to-point mapping (width × height)
  # false -> Unordered, removing invalid points to reduce data size
  ordered_pc: false

  # Allow points without texture (color) to remain in the point cloud
  # true  -> Keep all valid depth points, even without color
  # false -> Remove points without texture mapping
  allow_no_texture_points: false

# Enable specific filters (comma-separated if multiple)
filters: colorizer

colorizer:
```

(continues on next page)

(continued from previous page)

```
  # Preset for colorizer filter (affects contrast/brightness of depth
coloring)
  visual_preset: 2
  # Color scheme for depth visualization (e.g., jet, white-to-black, etc.)
  color_scheme: 2

# Time (in seconds) to wait before attempting to reconnect after device loss
reconnect_timeout: 6.0
```

Note

Setting the FPS or image size too high when pointcloud is enabled can cause performance issues. Please reduce the FPS or image size to suit your use case.

Update Firmware

```
# Dry run / info
rs-fw-update -l

# Update (example; replace with your FW file)
sudo rs-fw-update -f /path/to/Intel_RealSense_D4XX_FW.bin
```

Note

- Use the firmware recommended for your specific model and compatible version with `librealsense`.
- Keep the camera on a stable USB 3.x port during updates. Do **not** disconnect power.

4.5 Appendix

4.5.1 What is ABI mismatch error?

ABI (Application Binary Interface) mismatch occurs when the compiled application expects certain library versions or system interfaces that differ from those available on the target system.

It can lead to runtime errors or crashes when the application is executed on the target device.

In the context of cross-building ROS2 applications for the Renesas RZ/V2H RDK platform by using the Yocto SDK, an ABI mismatch error may arise if there are discrepancies between the versions of ROS2 packages and libraries used during the cross-compilation process **(installed in the SDK sysroot)** and those present on the target RZ/V2H RDK board **(installed over apt repository)**.

Example output indicating ABI mismatch:

```
[ros2_control_node-3] Stack trace (most recent call last):
[ros2_control_node-3] #17   Object "/usr/lib/aarch64-linux-gnu/ld-
linux-aarch64.so.1", at 0xffffffffffffffff, in
[ros2_control_node-3] #16   Object "/opt/ros/jazzy/lib/controller_
manager/ros2_control_node", at 0xaaaad5ad5d6f, in _start
[spawner-5] [INFO] [1756346496.296861080] [spawner_joint_state_
```

(continues on next page)

(continued from previous page)

```
broadcaster]: waiting for service /controller_manager/list_
controllers to become available...
[ros2_control_node-3] #15   Source "../csu/libc-start.c", line 360,
in __libc_start_main_impl [0xffff9b068597]
[ros2_control_node-3] #14   Source "../sysdeps/nptl/libc_start_call_
main.h", line 58, in __aarch64_ldadd4_relax [0xffff9b0684c3]
[ros2_control_node-3] #13   Object "/opt/ros/jazzy/lib/controller_
manager/ros2_control_node", at 0xaaaad5ad4e07, in main
[ros2_control_node-3] #12   Object "/opt/ros/jazzy/lib/librclcpp.so",
at 0xffff9b6f342b, in
rclcpp::executors::MultiThreadedExecutor::spin()
[ros2_control_node-3] #11   Object "/opt/ros/jazzy/lib/librclcpp.so",
at 0xffff9b6f30b3, in
rclcpp::executors::MultiThreadedExecutor::run(unsigned long)
[ros2_control_node-3] #10   Object "/opt/ros/jazzy/lib/librclcpp.so",
at 0xffff9b6dd8a3, in rclcpp::Executor::execute_any_
executable(rclcpp::AnyExecutable&)
.............................
[ros2_control_node-3] #0    Object "/opt/ros/jazzy/lib/libhardware_
interface.so", at 0xffff9af7e26c, in
[ros2_control_node-3] Bus error (Invalid address alignment [0x9])
```

4.5.2 How to avoid ABI mismatch error?

Using the check_cross_build_versions.sh script to check for version mismatches between SDK and target.

> **Note**
>
> Make sure to update the `PROJECT_ROOT` variable in the script to point to your ROS2 workspace `src` directory before running it.

If the script reports any mismatches, you may need to take one of the following actions:

- Try to run the application on the target system and see if it works despite the version mismatches.
- Rebuild the Yocto SDK with the correct versions of ROS2 packages that match those on the target system.
- Manual update the ROS2 packages on the SDK sysroot to match the target system versions. Refer to the next section for detailed steps.

4.5.3 How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?

The procedure below outlines the steps to update or add ROS2 packages in the Yocto SDK sysroot to resolve ABI mismatch issues or missing packages.

If you plan to update the version of an existing ROS2 package, please follow the steps under the **Optional** section. Otherwise, you can skip to the main steps.

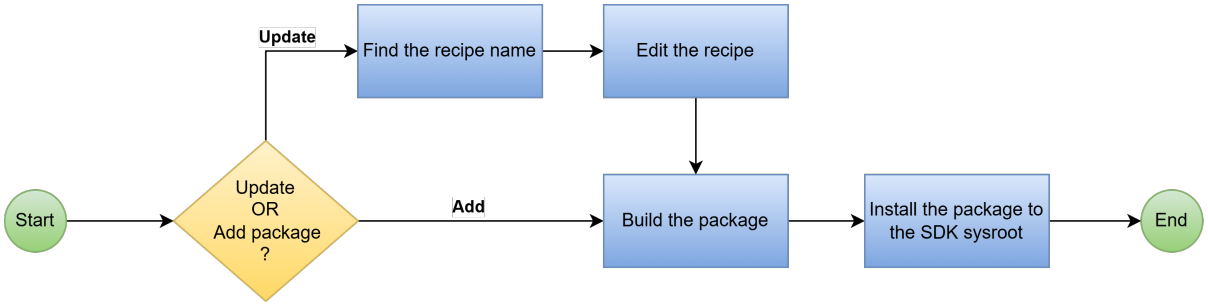

Fig. 8: Updating/Adding ROS2 packages in Yocto SDK sysroot

Note

This section assumes that you have already set up the Yocto eSDK for RZ/V2H RDK as per the instructions in the *RZ/V2H eSDK Setup* section.

- Open **a new terminal** and source the Yocto eSDK environment:

```
$ source ~/poky_sdk/environment-setup-cortexa55-poky-linux
```

Please replace the path with the actual path to your eSDK working folder.

- **Optional:** If you want to update version of an existing ROS2 package in the SDK sysroot, please edit the correct recipe file in the `~/poky_sdk/layers/meta-ros/meta-ros2-jazzy`.

For example, to update the `pinocchio` package:

1. Find the corresponding recipe file located at: `~/poky_sdk/layers/meta-ros/meta-ros2-jazzy`

```
 ubuntu@ros-xbuild:~/poky_sdk/layers/meta-ros/meta-ros2-jazzy$ find . -
name "*pinocchio*"
 ./generated-recipes/pinocchio
 ./generated-recipes/pinocchio/pinocchio_3.6.0-1.bb
 ./generated-recipes/kinematics-interface-pinocchio
 ./generated-recipes/kinematics-interface-pinocchio/kinematics-interface-
pinocchio_0.0.1-1.bb
 ./recipes-bbappends/pinocchio
 ./recipes-bbappends/pinocchio/pinocchio_%.bbappend
 ubuntu@ros-xbuild:~/poky_sdk/layers/meta-ros/meta-ros2-jazzy$
```

2. Edit the recipe file.

For example, to update to version `3.8.0`, modify the relevant fields:

- Find the correct new SRCREV for newer version available at the package's official repository.

Tip

You can find the correct SRCREV by visiting the package's GitHub repository which is SRC_URI combine with ROS_BRANCH. Open the `pinocchio_3.6.0-1.bb` file and check the following fields:

```
# matches with: https://github.com/ros2-gbp/pinocchio-release/
archive/release/jazzy/pinocchio/3.6.0-1.tar.gz
```

```
ROS_BRANCH ?= "branch=release/jazzy/pinocchio"
SRC_URI = "git://github.com/ros2-gbp/pinocchio-release;${ROS_
BRANCH};protocol=https"
SRCREV = "38cb5592495c829a7e41c9b796ff9e617c797b27"
S = "${WORKDIR}/git"
```

The SRCREV is usually the commit hash or tag corresponding to the desired version.

For example, for `pinocchio`, you can check its GitHub releases for ROS2 Jazzy: Pinocchio Releases Jazzy.

And the correct SRCREV for version `3.8.0` is `89e88cb7ac670c41dd86f48e5b4ac4c697579811` which is the commit hash of the release tag: release/jazzy/pinocchio/3.8.0-1.

- Update the SRCREV field in the recipe file - `pinocchio_3.6.0-1.bb` accordingly.
- Rename the recipe file to reflect the new version, e.g., `pinocchio_3.8.0-1.bb`. Also, update the name of bbappend files if necessary.

- Build the package.

```
$ devtool modify pinocchio
$ devtool build pinocchio
```

- Install the package into the SDK sysroot:

```
$ cd ~/poky_sdk/workspace/sources/pinocchio/oe-workdir/image/opt/ros/jazzy
$ sudo cp -r * /opt/poky/5.1.4/sysroots/cortexa55-poky-linux/opt/ros/jazzy/
```

4.5.4 Common ROS 2 topics

1. A typical ROS 2 workspace has the following structure:

```
ros2_ws/
├── src/                # Source code for ROS 2 packages
│   ├── package_1/
│   ├── package_2/
│   └── ...
├── build/              # Build output directory (generated)
├── install/            # Installation directory (generated)
└── log/                # Log files (generated)
```

2. Code style should follow the official documentation: Code style and language versions.
3. Organizing Files and Folders Inside a ROS 2 Package: Package Organization For a ROS Stack [Best Practices].

5 Frequency Asked Questions

1. Does the RZ/V2H RDK support multiple operating systems?

Yes, the RZ/V2H RDK supports multiple operating systems running concurrently on its different CPU cores.

For more details, refer to the *RZ/V Multi-OS* section.

2. How can I set up the USB-UART interface on the RZ/V2H RDK?

Refer to the *Other interfaces - USB-UART* section for detailed instructions on setting up the USB-UART interface for serial communication and debugging.

3. How do I configure a static IP address for high-speed interfaces on the RZ/V2H RDK?

Refer to the *Set Static IP* section for step-by-step instructions on configuring a static IP address.

4. I build the Ruiyan related ROS2 packages, but encounter the following error:

```
--- stderr: rh6_ctrl
  CMake Error at CMakeLists.txt:47 (message):
  library Path: /home/ubuntu/ros2_ws/src/ruiyan_rh2_controller/rh6_
ctrl/lib
  not found!


  ---
  Failed   <<< rh6_ctrl [0.97s, exited with code 1]
```

Please make sure to copy the `libRyhandArm64.so` file to the SDK sysroot before cross-compilation as mentioned in the *Dexterous Hand Control Application* and *Rock Paper Scissors Application* sections.

5. Does the RZ/V2H RDK support Docker?

Yes, the RZ/V2H RDK supports Docker. You can install Docker on the RZ/V2H RDK by following steps:

```
$ sudo apt update

# Make sure you can access the internet
$ ping bing.com

# Install docker
$ curl -fsSL https://get.docker.com | sudo sh
```

(continues on next page)

(continued from previous page)

```
# Add user to docker group
$ sudo usermod -aG docker $USER

# Logout and Login back to make it effects

# Simple check
$ docker --version
$ docker run hello-world
```

6. How to avoid ABI mismatch error when cross-compiling ROS2 applications?

Please refer to the *How to avoid ABI mismatch error?* section for detailed instructions.

7. How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?

Please refer to the *How to update/add the Yocto SDK with the correct ROS2 package versions with eSDK?* section for detailed instructions.

8. What USB-WIFI adapters are compatible with the RZ/V2H RDK?

The following USB-WIFI adapters have been tested and are compatible with the RZ/V2H RDK:

- Ralink Technology, Corp. MT7601U Wireless Adapter

  Linux firmware download link: mt7601u.bin

  Firmware path: `/lib/firmware/mt7601u.bin`

- AC1300 Tp-Link T3U Nano

  Linux firmware download link: rtw8822b_fw.bin

  Firmware path: `/lib/firmware/rtw88/rtw8822b_fw.bin`

> **Important**
>
> By default, no firmware is installed. Please install the correct **Linux firmware file** into the appropriate **firmware path** on the **RZ/V2H RDK root file system**, and then reboot the device to ensure proper driver support for these adapters.

If you have other USB-WIFI adapters, please refer to the manufacturer's documentation for compatibility and driver installation instructions.

You can take the following steps as general guidance to check and install the required drivers and firmware for other USB-WIFI adapters:

- Enable the corresponding driver in the Linux kernel configuration (refer to *Custom Linux Kernel and Device Tree*).

  Recommend enable the driver as a module (m type) and copy the built module to the RZ/V2H RDK.

  For example, for the two adapters listed above, the following kernel configurations have already been enabled:

```
CONFIG_MT7601U
CONFIG_MT76x0U
CONFIG_RTL8XXXU
```

(continues on next page)

(continued from previous page)

```
CONFIG_RTW88
CONFIG_RTW88_8822BU
```

- Build and install the driver module on the RZ/V2H RDK.
- Find the correct **firmware files** at: Linux fimrware repository and place them in the appropriate **directory path** within the **RZ/V2H RDK root file system** to ensure proper device operation. Typically, firmware files are located in the `/lib/firmware/` directory.
- Reboot the RZ/V2H RDK and check if the USB-WIFI adapter is recognized and functioning properly.

> **Tip**
>
> All firmware file can be found in the Linux firmware repository.
>
> Please searching the repository with the USB-WIFI adapter model name or chip-set can help to locate the correct firmware file.

How to know which driver and firmware is needed for your USB-WIFI adapter?

- Connect the USB-WIFI adapter to the RZ/V2H RDK.
- Open a terminal and run the command: `lsusb` to list all connected USB devices.
- Identify your USB-WIFI adapter from the list and note its Vendor ID and Product ID.
- Use the Vendor ID and Product ID to search online for the specific driver and firmware required for your USB-WIFI adapter model.
- Refer to the manufacturer's website or Linux community forums for additional information on driver and firmware installation.
- Using `dmesg` command after plugging in the USB-WIFI adapter can also provide useful information about the required driver and firmware.

  For example, run the command: `dmesg | grep -i firmware` to check for any firmware-related messages.

```
rz@localhost:~$ dmesg | grep firmware
[   17.946635] rtw_8822bu 3-1:1.0: Direct firmware load for rtw88/
rtw8822b_fw.bin failed with error -2
```

  This log indicates that the `rtw8822b_fw.bin` firmware file is required at `/lib/firmware/rtw88/rtw8822b_fw.bin` for the adapter to function properly.

- Once you have identified the required driver and firmware, follow the steps mentioned above to install them on the RZ/V2H RDK.

9. Failed opening device /dev/video0 or /dev/tty: Permission denied

Please complete the First Time Boot Setup as described in the *First Time Boot Setup* section to add your user to the necessary groups (e.g., *video*, *dialout*) to gain access to these device files.

10. No space left on device error

Please complete the First Time Boot Setup as described in the *First Time Boot Setup* section to expand the root filesystem to utilize the full storage capacity of the RZ/V2H RDK.