VX-01

Docker Infrastructure — User Manual

Hybrid Amphibious Search & Rescue Robot

ROS2 Humble · Ignition Fortress 6 · RDK X5 ARM64 · Laptop AMD64

v1.0 — github.com/Gouri-Shankar-85/VX-01

Part 1 — Setting Up (Do This Once)

This part explains what to do before you ever run anything. You do all of this once. After that you never touch it again.

1.1 What You Need on Your Laptop

You need Ubuntu 22.04, Docker Engine, and docker buildx. If you already have Docker from a previous install, check the version:

```
docker --version          # needs 24.x or later
docker compose version    # needs v2 (shows 'Docker Compose version v2.x')
docker buildx version     # needs to exist
```

If Docker is not installed:

```
curl -fsSL https://get.docker.com | sudo sh
sudo usermod -aG docker $USER
# Log out and back in after this
```

> ⚠ After adding yourself to the docker group you MUST log out and log back in. Otherwise docker commands fail with permission denied.

1.2 Put the Docker Files in Your Project

Your project folder is ~/vx-01/. Copy all the files from this kit into it so the structure looks like this:

```
~/vx-01/
├── .env
├── .dockerignore
├── build.sh
├── deploy-rdk.sh
├── docker-compose.yml
├── docker-compose.rdk.yml
├── docker/
│   ├── Dockerfile.base
│   ├── Dockerfile.sim
│   ├── entrypoint.sh
│   └── udev/
│       └── 99-vx01-devices.rules
└── vx01_ws/
    └── src/          ← your ROS2 packages
```

Now open .env and set your Docker Hub username. That is the only thing you need to fill in:

```
DOCKER_REGISTRY=yourdockerhub    # replace this with your actual username
```

1.3 Install Device Rules on Your Laptop Host

This is the step you asked about. You do this on the laptop HOST (not inside Docker). Run it once:

```
cd ~/vx-01
chmod +x build.sh
./build.sh udev-install
```

What this does: it copies 99-vx01-devices.rules to /etc/udev/rules.d/ on your laptop and reloads udev. From then on, whenever you plug in any VX-01 device it appears with a stable name (/dev/ttyTFMINI, /dev/ttyPIXHAWK, etc.) instead of /dev/ttyUSB0 which changes every time.

> ***i*** *This is the answer to your question about port names changing. After this step, the port names never change regardless of which USB socket you use.*

> ⚠ After running udev-install you need to log out and back in once, because it also adds you to the dialout and plugdev groups.

1.4 Allow Docker to Open GUI Windows (X11)

You need this every time you log into your laptop, before opening Gazebo or RViz. Run it once per session:

```
xhost +local:docker
```

To avoid doing this every session, add it to the end of your ~/.bashrc:

```
echo 'xhost +local:docker' >> ~/.bashrc
```

1.5 GitHub SSH — How It Works for Multiple Contributors

There are no usernames stored inside the Docker image or in any config file. Here is how it works:

- The docker-compose files mount your ~/.ssh folder (read-only) into the container
- They also mount your ~/.gitconfig
- The entrypoint script sets the git remote to git@github.com:Gouri-Shankar-85/VX-01.git automatically
- Each contributor uses their own laptop's SSH key — no shared credentials, no conflicts

Since you already have your SSH key on GitHub, git push will work inside the container with no extra setup. The only requirement is that your ~/.ssh/id_ed25519 or ~/.ssh/id_rsa exists on your laptop.

For a new contributor: they clone the repo, put their own SSH key on GitHub, and the container automatically uses their key. Nothing in the repo needs to change.

Part 2 — Building the Docker Images

You build images when: you first set up the project, or when you change the Dockerfile (add a new ROS package, etc.). You do NOT rebuild images just to change your ROS2 code — code changes are picked up via the mounted src/ folder.

2.1 Build the Hardware Image (Laptop AMD64)

This is the image that contains all drivers, MAVROS, RTAB-Map, Pololu library, and all sensors. It is used for hardware testing on your laptop and is the same code that runs on the RDK X5 (different architecture, same Dockerfile):

```
cd ~/vx-01
./build.sh build-base
```

First build takes 20-40 minutes. Docker caches every layer so from second build it is fast unless you change the Dockerfile.

2.2 Build the Simulation Image (Laptop AMD64 Only)

This extends the hardware image with Ignition Fortress, ROS-GZ bridge, and RViz2. Never built for the RDK X5:

```
./build.sh build-sim
```

2.3 Build for RDK X5 (ARM64)

This cross-compiles the hardware image for ARM64 and automatically pushes it to Docker Hub. The RDK X5 will pull it from there. Make sure your .env has the correct DOCKER_REGISTRY before running this:

```
./build.sh build-arm
```

> *i This uses docker buildx with QEMU. It takes longer than a normal build because it is emulating ARM64 on your AMD64 machine. ~30-60 min first time.*

2.4 When Do You Need to Rebuild?

Situation	Rebuild needed?
You changed a .cpp or .py file in vx01_ws/src/	No — just run colcon build inside the container

Situation	Rebuild needed?
You added a new ROS2 package to src/	No — just run colcon build inside the container
You added a new apt package to a Dockerfile	Yes — rebuild that image
You changed entrypoint.sh	Yes — rebuild
You added a new sensor with a new driver	Yes — add it to Dockerfile.base, rebuild
Nothing changed, just starting work today	No — just start the container

Part 3 — Working on the Project Every Day (Laptop)

This is what you actually do each day. Short version: start container, write code, build, test, push to GitHub. No image rebuilds needed.

3.1 Open Your Dev Shell (Do This Every Day)

```
cd ~/vx-01
docker compose --profile dev up -d
docker exec -it vx01-dev bash
```

This opens a terminal inside the container. Your source code at ~/vx-01/vx01_ws/src/ is mounted live — any file you edit on your laptop appears instantly inside the container and vice versa. You can use your laptop's code editor (VS Code, etc.) normally.

3.2 Build Your ROS2 Workspace After Code Changes

Inside the container:

```
cd /vx01_ws
colcon build --symlink-install
source install/setup.bash
```

> *i --symlink-install means you do not need to re-run colcon build for Python script changes. Only C++ needs a rebuild.*

3.3 Run Simulation

From a second terminal on your laptop:

```
docker exec -it vx01-dev bash
ros2 launch vx01_sim sim.launch.py
```

Or start the dedicated sim service instead:

```
docker compose --profile sim up
```

3.4 Run Hardware (Real Sensors on Laptop)

Plug in your devices, then:

```
docker compose --profile hardware up
```

The entrypoint will print which devices it found. If a device shows as NOT connected, check the cable and that udev-install was done.

3.5 Push Code to GitHub

From inside any running container (dev, hardware, or sim):

```
cd /vx01_ws
git status
git add src/your_package/
git commit -m 'your message'
git push
```

This uses your mounted SSH key automatically. No login prompts.

3.6 Pull Latest Code from GitHub

```
cd /vx01_ws
git pull
colcon build --symlink-install
```

3.7 Stop Everything

```
# From your laptop (not inside container):
docker compose --profile dev down
docker compose --profile sim down
docker compose --profile hardware down
```

3.8 Run Hardware + Simulation Together (Hardware-in-the-Loop)

Start sim container and hardware container at the same time. Both share the same ROS2 network via network_mode: host, so topics from real sensors show up in Gazebo and vice versa:

```
docker compose --profile sim up -d
docker compose --profile hardware up -d
```

Part 4 — RDK X5 Setup and Daily Use

4.1 First-Time RDK X5 Setup (Do Once)

Yes, you need Docker on the RDK X5. The deploy-rdk.sh script installs it for you. Do this from your laptop:

```
scp deploy-rdk.sh root@<rdk-ip>:/root/
ssh root@<rdk-ip>
chmod +x deploy-rdk.sh
./deploy-rdk.sh yourdockerhub
```

The script does all of the following automatically:

- Installs Docker if not present
- Installs docker-compose plugin
- Writes udev rules for all VX-01 devices
- Adds you to dialout/plugdev/video groups
- Creates docker-compose.rdk.yml
- Pulls the ARM64 image from Docker Hub
- Starts the robot container with auto-restart enabled

> *i After deploy-rdk.sh finishes, the robot container starts automatically every time the RDK X5 boots. You do not need to SSH in each time.*

4.2 Updating the Robot After Code Changes

Step 1: On your laptop, after committing and pushing your changes, rebuild the ARM64 image and push it:

```
./build.sh build-arm
```

Step 2: On the RDK X5, pull the new image and restart:

```
ssh root@<rdk-ip>
docker pull yourdockerhub/vx01-base:humble-arm64
docker compose -f /root/vx01/docker-compose.rdk.yml down
docker compose -f /root/vx01/docker-compose.rdk.yml up -d
```

4.3 Everyday Commands on RDK X5

What you want to do	Command
Check robot is running	docker ps

What you want to do	Command
See live logs	docker logs -f vx01-robot
Open a shell inside the robot container	docker exec -it vx01-robot bash
Check which devices are connected	docker exec vx01-robot ls -la /dev/tty*
Restart just the MAVROS service	docker restart vx01-mavros
Stop everything	docker compose -f /root/vx01/docker-compose.rdk.yml down
Start everything	docker compose -f /root/vx01/docker-compose.rdk.yml up -d
See CPU/RAM usage	docker stats

Part 5 — Devices and Stable Port Names

5.1 Why Port Names Change (and How We Fix It)

Without udev rules, Linux assigns port names based on the order devices are plugged in. TFmini plugged first gets /dev/ttyUSB0. Plug something else first and it becomes /dev/ttyUSB1. The udev rules in this project create permanent symlinks so the name is always the same regardless of plug order or USB port:

Device	Stable Symlink	Old Unstable Name	Protocol
Radilink PIX6 (ArduPilot)	/dev/ttyPIXHAWK	/dev/ttyACM0 (changes)	USB ACM 921600
Pololu Maestro 18ch	/dev/ttyMAESTRO	/dev/ttyACM1 (changes)	USB HID + Serial
Benewake TFmini-S	/dev/ttyTFMINI	/dev/ttyUSB0 (changes)	UART 115200 baud
Hiwonder IM10A IMU	/dev/ttyIMU	/dev/ttyUSB1 (changes)	UART 9600 baud
YDLIDAR HP60C	/dev/video0	/dev/video0 (UVC, stable)	USB 2.0 UVC

> *i The udev rules match devices by their USB vendor/product ID, not by port. So even if you replug everything or use a hub, the symlinks are created correctly.*

To find the vendor ID of an unknown device:

```
lsusb
# Shows: Bus 001 Device 004: ID 10c4:ea60 Silicon Labs CP210x
# The ID is vendor:product — put those in the udev rules
```

5.2 YDLIDAR HP60C Depth Camera

Parameter	Value
Technology	Structured light (speckle pattern), 940nm IR
Range	0.2 – 4m (best accuracy ≤2m), plane accuracy <2mm@1m
Depth image	640×480 @ 20fps, FOV H:73.8° V:58.8°
RGB image	1920×1080 @ 20fps, FOV H:80.9° V:51.7°

Parameter	Value
Interface	USB 2.0 Type-C (standard UVC — no proprietary SDK needed)
Power	5V USB, <1.5W average
ROS2 driver	libuvc + openni2 (pre-installed in image)
SLAM input topic	/camera/depth/image_raw → RTAB-Map

5.3 Benewake TFmini-S

Parameter	Value
Range	0.1–12m (90% reflectivity), 0.1–7m (10% reflectivity)
Accuracy	±6cm at 0.1–6m, ±1% at 6–12m
FOV	2° (narrow beam — good for obstacle/cliff detection)
Frame rate	1–1000Hz (default 100Hz)
Interface	UART 115200 baud (default) or I2C
Power	5V, ≤140mA
ROS2 topic	/tfmini/range (sensor_msgs/Range)

5.4 Hiwonder IM10A IMU

Parameter	Value
Sensors	3-axis gyro + 3-axis accel + magnetometer + barometer
Output rate	10Hz default, up to 200Hz
Gyro range / accuracy	±2000°/s, zero drift ±0.5–1°/s
Accel range	±16g, RMS noise 0.75mg
Yaw accuracy	0.1° (9-axis with mag), 0.5° (6-axis)
Interface	UART (9600 default, up to 921600) or I2C (up to 400kHz)
ROS2 support	Native — listed on datasheet
ROS2 topic	/imu/data (sensor_msgs/Imu)

5.5 Radilink PIX6 + ArduPilot + MAVROS

MAVROS is pre-installed. GeographicLib datasets are downloaded during the Docker build so you never have to do it manually. MAVROS connects on /dev/ttyPIXHAWK at 921600 baud.

```
# Verify connection inside container:
ros2 topic echo /mavros/state
# connected: True means it is working
```

5.6 Pololu Mini Maestro 18-Channel

The libpololu-usb C library is built from source inside the Docker image from Pololu's open-source SDK at github.com/pololu/pololu-usb-sdk. This works on both AMD64 and ARM64. The Maestro appears as /dev/ttyMAESTRO (virtual serial) and also as a USB HID device (for the hidapi Python binding).

```
# Verify Maestro is detected inside container:
ls -la /dev/ttyMAESTRO
lsusb | grep '1ffb'
```

5.7 SLAM Without a 2D LiDAR

VX-01 has no spinning 2D LiDAR. RTAB-Map handles this correctly using the HP60C depth camera alone. It builds a 3D point cloud map, does loop closure detection, and publishes a /rtabmap/map topic that Nav2 can use for navigation.

SLAM option	Needs 2D LiDAR	Works with HP60C	Installed
RTAB-Map	No	**✓ RGB-D mode, primary choice**	**✓ Yes**
ORB-SLAM3	No	**✓ RGB-D mode, heavier**	**✓ Yes**
slam_toolbox	Yes	✗ needs LaserScan	No
cartographer	Yes	✗ needs LaserScan	No

```
# Launch RGB-D SLAM:
ros2 launch rtabmap_ros rtabmap.launch.py \
  rgb_topic:=/camera/color/image_raw \
  depth_topic:=/camera/depth/image_raw \
  camera_info_topic:=/camera/color/camera_info \
  approx_sync:=true
```

Part 6 — Troubleshooting

Problem	Most likely cause	Fix
Gazebo or RViz window does not open	X11 not allowed	Run on laptop: xhost +local:docker then restart container
/dev/ttyTFMINI not found (symlink missing)	udev rules not installed on host	Run: ./build.sh udev-install on the HOST, then replug the device
Permission denied on /dev/ttyUSB0	Not in dialout group yet	sudo usermod -aG dialout $USER then log out and back in
MAVROS shows connected: False	Wrong device path or baud	Check ls /dev/ttyPIXHAWK exists. Verify ArduPilot baud is 921600
docker buildx build fails	No builder created	docker buildx create --name vx01-builder --use then retry
colcon build fails with missing package	rosdep not run	Inside container: rosdep install --from-paths src --ignore-src -r -y
git push asks for password	SSH key not mounted or wrong remote URL	Check ~/.ssh/id_* exists. Entrypoint sets remote to SSH automatically
Container exits immediately on RDK X5	launch file not found (workspace not built)	docker exec -it vx01-robot bash then: cd /vx01_ws && colcon build
HP60C not seen as /dev/video0	USB hub bandwidth issue	Plug directly into RDK X5 USB port, not through a hub. Check: v4l2-ctl --list-devices
TFmini gives wrong distances	Baud rate mismatch	Default is 115200. Match your ROS2 driver config to the sensor config
Two CP2102 adapters, TFmini and IMU on same port	udev rule collision	Check ATTRS{serial} to distinguish them: udevadm info -a -n /dev/ttyUSB0

Part 7 — Quick Reference

One-Time Setup (Laptop)

```
./build.sh udev-install           # install device rules on host
echo 'xhost +local:docker' >> ~/.bashrc && source ~/.bashrc
./build.sh build-base             # build hardware image
./build.sh build-sim              # build sim image
./build.sh build-arm              # cross-compile + push to Docker Hub
```

Every Day (Laptop)

```
docker compose --profile dev up -d          # start dev container
docker exec -it vx01-dev bash               # open shell
  cd /vx01_ws && colcon build --symlink-install   # inside container
  git pull / git push                               # inside container
docker compose --profile sim up             # run Gazebo
docker compose --profile hardware up        # run real hardware
docker compose --profile dev down           # stop when done
```

After Changing a Dockerfile

```
./build.sh build-base             # if Dockerfile.base changed
./build.sh build-sim              # if Dockerfile.sim changed
./build.sh build-arm              # to update RDK X5
```

RDK X5 First Time

```
scp deploy-rdk.sh root@<ip>:/root/
ssh root@<ip> 'chmod +x deploy-rdk.sh && ./deploy-rdk.sh yourdockerhub'
```

RDK X5 After Image Update

```
docker pull yourdockerhub/vx01-base:humble-arm64
docker compose -f /root/vx01/docker-compose.rdk.yml down
docker compose -f /root/vx01/docker-compose.rdk.yml up -d
```