Maker's Pet®



Build, Program a Self-Driving Robot

No robotics experience needed. Hardware optional.

Build, Program a Self-Driving Robot
Edition 1.0, October 2026

By Ilia O.

© 2026 REMAKE AI Statutory Trust, d/b/a Maker's Pet
makerspet.com

Maker's Pet® is a registered trademark of REMAKE AI Statutory Trust.

The latest edition, build notes and support links are at makerspet.com/blog/bld-120mm-pack.

Contents

1. Assemble the Robot 9
2. Set Up the Software on a Windows PC 31
3. Upload the Firmware and Bring Up the Robot 51
4. Map Your Place by Driving the Robot 84
5. Let the Robot Explore and Self-Drive 99
6. Practice in Simulation (No Robot Needed) 120
7. Program Your Robot 135
8. Using an Ubuntu PC Instead 145
Appendix A. Run Ubuntu in a Virtual Machine on Windows 170
Appendix B. Troubleshooting and FAQ 182
Appendix C. Quick Reference 204
Appendix D. 3D Print the Robot Parts 208

About This Book

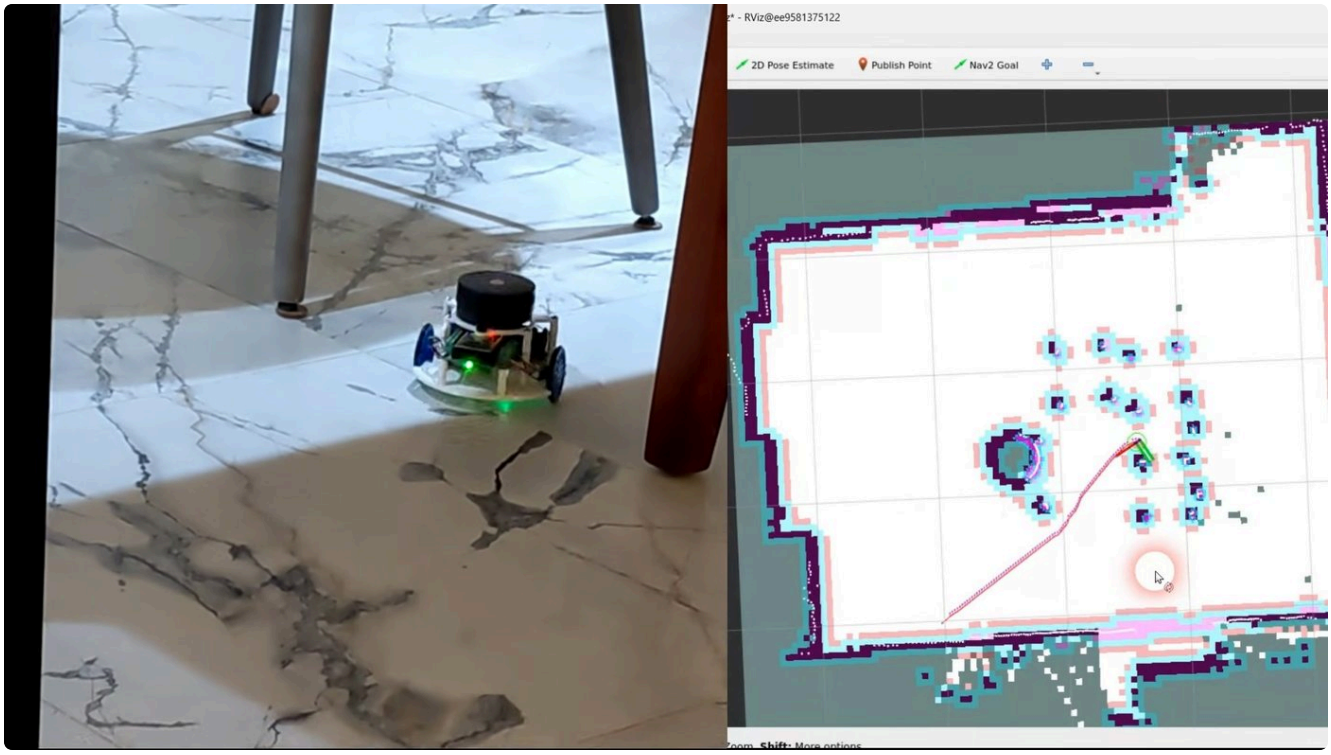
This step-by-step guide takes you from a bag of parts to your own LiDAR robot that maps your home and drives itself to any spot you click on the map. You'll assemble it, map your place, program it and let an AI assistant drive it. The book follows my Maker's Pet video course, adds the fixes and tips I collected from builders after the videos were recorded, and puts everything in one place you can read at your workbench.

What you will build

The robot is about 120 mm across. An ESP32 microcontroller, programmed with the Arduino IDE, drives two N20 gear motors with encoders and reads a spinning LiDAR distance sensor. Over WiFi, the ESP32 talks to ROS 2, the Robot Operating System, running on your PC. ROS 2 does the heavy lifting: building maps, working out where the robot is, and planning a path around obstacles.

Once it's built, your robot can:

- **Map your place completely on its own**, using SLAM and frontier exploration
- **Drive itself** to any location you pick on the map, avoiding obstacles along the way
- **Be driven manually** from your PC keyboard
- **Show its LiDAR readings** live on your screen
- **Run in a 3D simulation**, mapping and navigating a virtual world
- **Be reprogrammed** in Python or C++ to change its behavior or process its sensor data



The finished robot navigating on its own, next to the live map it builds.

What you need

- **The robot parts.** The Maker's Pet *Arduino/ROS2 Self-Driving Robot 120mm Build Pack* (BLD-120MM-PACK) contains everything, including the BDC-30P driver board, a Maker's Pet ESP32-E dev kit and an LDROBOT LD14P LiDAR. If you have a 3D printer, you can print the plastic parts yourself; the print files are listed in Appendix D.
- **Six AA alkaline batteries.**
- **A PC** running Windows 10/11 or Ubuntu 22.04. The PC runs the robot's "brain", so a faster PC gives smoother mapping and navigation.
- **A 2.4 GHz WiFi network** that both the robot and the PC can join. The ESP32 can't use 5 GHz.
- **A USB data cable** for the ESP32. Some cables only carry power and won't work.
- **A small screwdriver and a wire stripper.**

No robotics experience is necessary.

The hardware is optional. You can learn without a physical robot. Chapter 6 runs a simulated robot in a 3D world on your PC, where you drive it, map a room and send it to goals. The programs in Chapter 7 run on the simulated robot too. All you need is the PC.

How long it takes. If you've built a kit like this before, allow about an hour each for the software setup, the assembly and the bring-up. For a first build, set aside a day.

How this book is organized

Chapter	What you do	Video
1. Assemble the Robot	Put the mechanical and electrical parts together	5 min
2. Set Up the Software on a Windows PC	Install the Arduino IDE, the firmware, WSL2, Docker and an X server	7 min
3. Upload the Firmware and Bring Up the Robot	Flash the ESP32, connect it to WiFi and test the motors and LiDAR	9 min
4. Map Your Place by Driving the Robot	Drive the robot by keyboard to create your first map	6 min
5. Let the Robot Explore and Self-Drive	Autonomous exploration and navigation to goals	6 min
6. Practice in Simulation	The same mapping and navigation in a 3D simulator, no hardware needed	10 min
7. Program Your Robot	Two Python programs (face the closest object, follow me), then let an AI assistant drive	
8. Using an Ubuntu PC Instead	Chapters 2 and 3, the Linux way	13 min
Appendix A	Running Ubuntu in a virtual machine on Windows	6 min
Appendix B	Troubleshooting and FAQ	
Appendix C	Quick reference: commands, LED codes and links	
Appendix D	3D printing the robot's plastic parts	

Pick your path:

- **Windows PC:** Chapters 1 → 2 → 3 → 4 → 5.
- **Ubuntu PC:** Chapter 1 → 8 → 4 → 5.
- **No robot yet?** Install the PC software (Chapter 2 or 8, skipping the Arduino parts), then go straight to Chapters 6 and 7.

Each chapter starts with a link to its video. The illustrations are frames from that video, so if a step isn't clear from a picture, you can watch that part of the video.

Conventions

Commands you type are shown like this:

```
docker container stop makerspet
```

Commands that start with `docker` run in a Windows PowerShell (or Ubuntu terminal) window; commands that start with `ros2` run inside the robot software's Linux shell, which Chapter 3 shows you how to open.

SINCE THE VIDEO WAS RECORDED
Boxes like this one mark corrections and additions made after the videos were published. Where the book and a video disagree, follow the book.

TIP
Tips make a step easier or save you from a common mistake.

WARNING
Warnings protect your robot, your batteries or your PC. Please don't skip them.

Software versions

This edition was written for Kaia.ai firmware 0.8.0 (ROS 2 Iron), the ESP32 Arduino board package 2.0.17 and the `kaiaai/kaiaai:iron` Docker image. A ROS 2 Jazzy image, `kaiaai/kaiaai:jazzy`, is also available; see Chapter 8.

Open source, and where to get help

Everything in this project is open source. The robot software is licensed under Apache 2.0, so feel free to fork and modify it; please keep an acknowledgement.

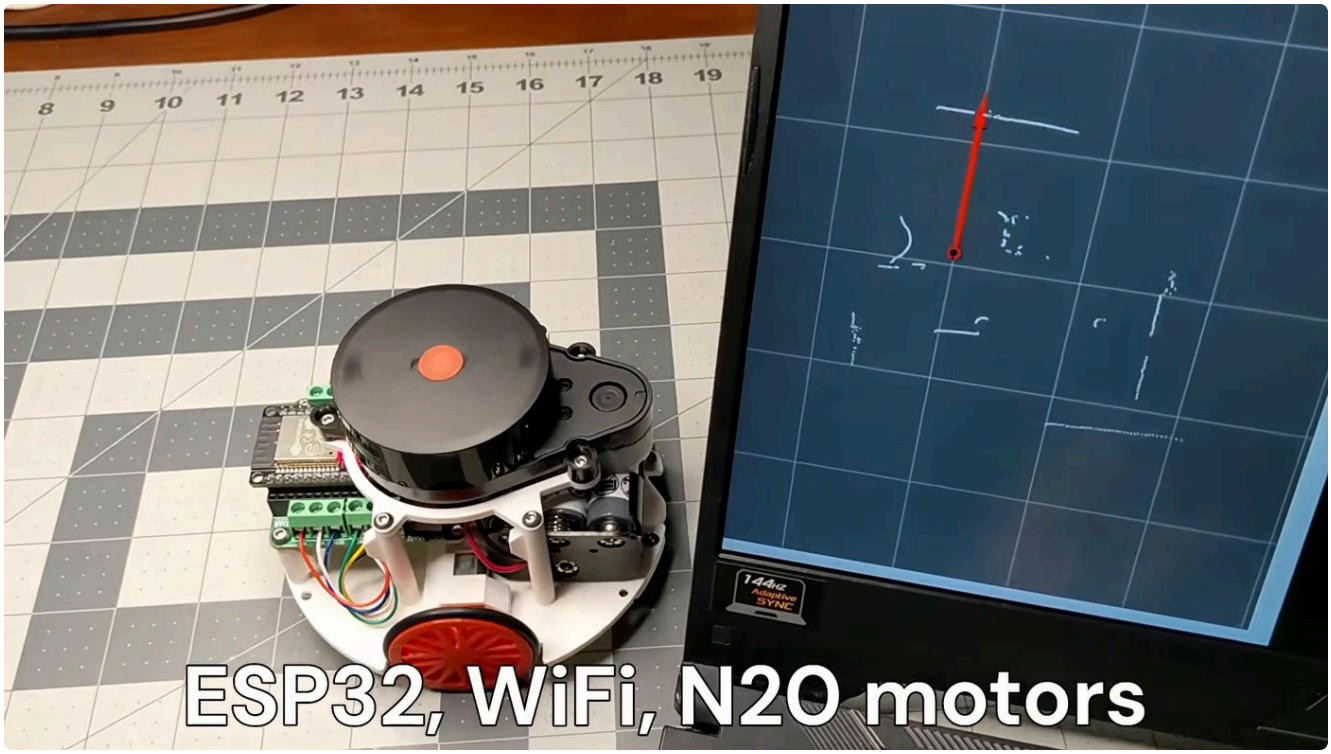
If you get stuck, first check Appendix B. If it doesn't cover your problem, ask on the support forum at github.com/makerspet/support/discussions or on the Maker's Pet Discord server. Up-to-date build notes are always at makerspet.com/blog/bld-120mm-pack.

Happy building!

1. Assemble the Robot

Video: How to Create a Self-Driving LiDAR Robot with Arduino, ESP32, ROS2

You'll wire the motors to the BDC-30P driver board, assemble the chassis, fit the ESP32, battery and LiDAR, and connect the ESP32 to your computer. No software is needed yet. Assembly takes about an hour once you know the steps; allow more for your first build.



The finished robot: ESP32, N20 motors and LiDAR.

> **NOTE**
> The video shows the February 2025 instructions. The "Since the video was recorded" boxes add later corrections from the bring-up and troubleshooting guide (makerspet.com/blog/bld-120mm-pack/).

What's in the kit and the tools you need

Lay everything out before you start:

- **Base plate** (round, white).
- **Board posts** (4) and **LiDAR posts** (4).
- **Caster wheel mounts** (2), the **roller** and its **metal shaft**.
- **Motor clamps** (2) and two **N20 gear motors with encoders**.
- **Wheels** (2) and **tires** (2).

- **Motor cables** (2): a plug on one end, loose wires on the other.
- **BDC-30P driver board** (open source).
- **Maker's Pet ESP32-E 30-pin dev kit module**, compatible with the original 30-pin ESP32 DOIT DevKit v1.
- **LDROBOT LD14P LiDAR** and its **breakout cable**.
- **LiDAR skirt** (the white ring that holds the LiDAR).
- **Battery case** for 6 AA batteries.
- **M3 screws** of two kinds: *countersunk* (flat head) screws go on the underside of the base; *hex button* (rounded head) screws go everywhere else.

Tools:

- A Phillips screwdriver for the countersunk screws and the screw terminals (an electric one is optional).
- A hex key or hex bit for the hex button screws.
- A wire stripper.
- A USB cable for the ESP32.



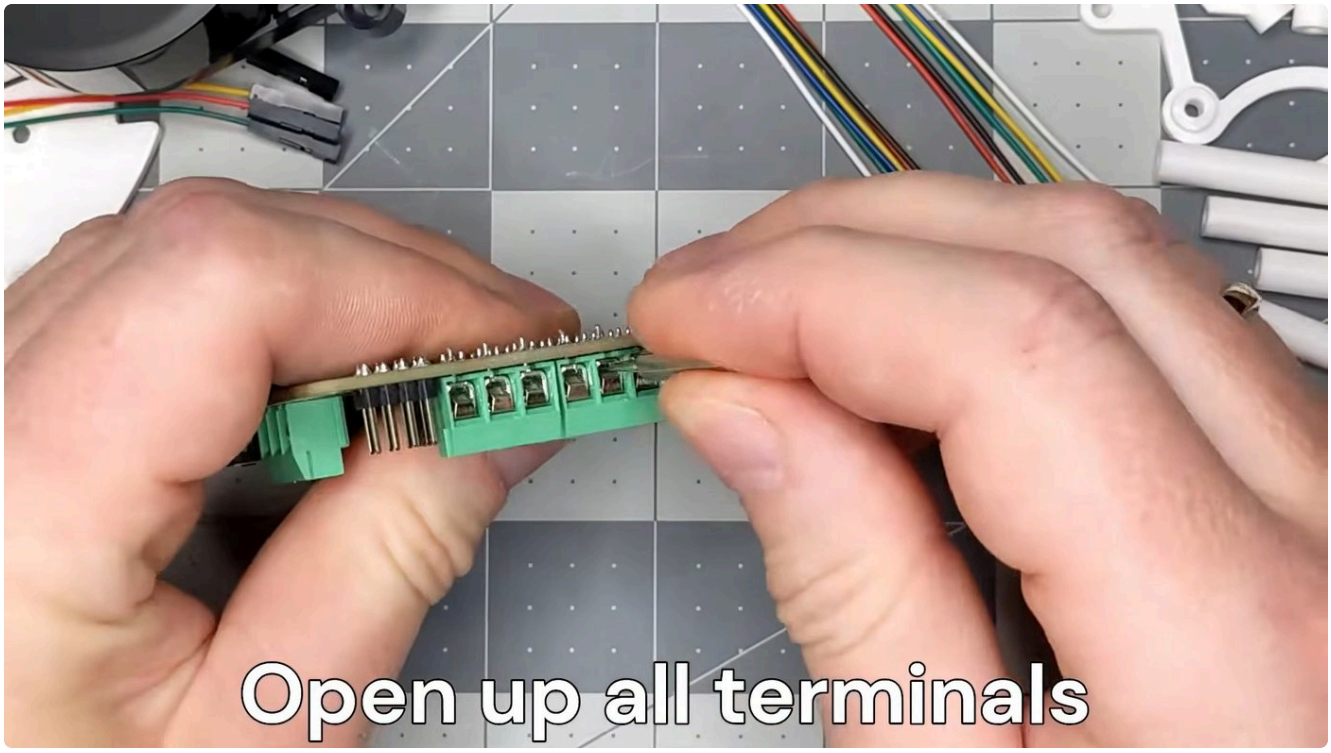
The kit laid out.

> **PRINTING THE PARTS YOURSELF**
> The plastic parts are open source. Appendix D lists the print files and the one slicer setting that matters.

Connect the motor wires to the board

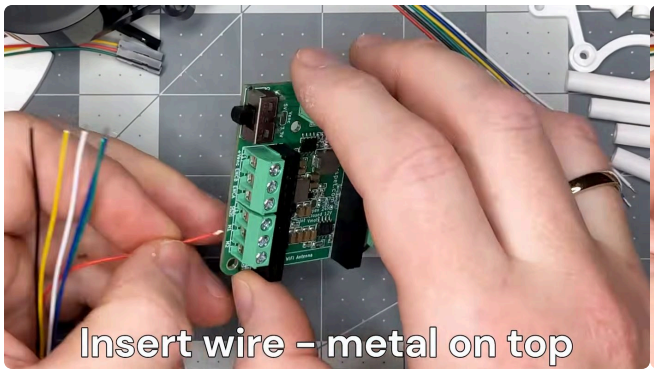
Wire the screw terminals first, while the board is loose in your hand.

1. Unscrew all the board's wire terminals fully.

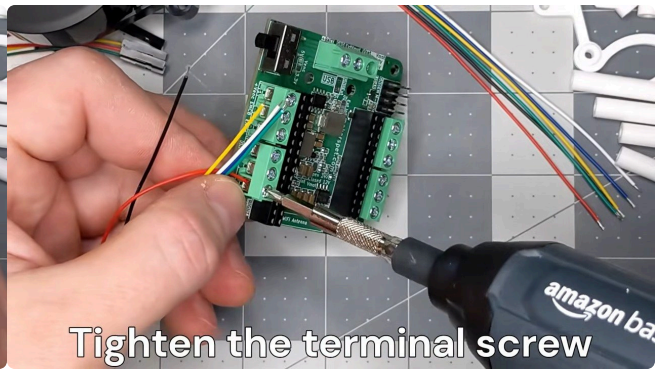


Open up all the terminals fully.

2. Strip 8–10 mm of insulation off a motor wire.
3. Bend the bare strands back over the insulation. The thicker end grips better.
4. Insert the wire with the bare strands facing **up** and tighten the screw.



Insert the wire with the metal on top.



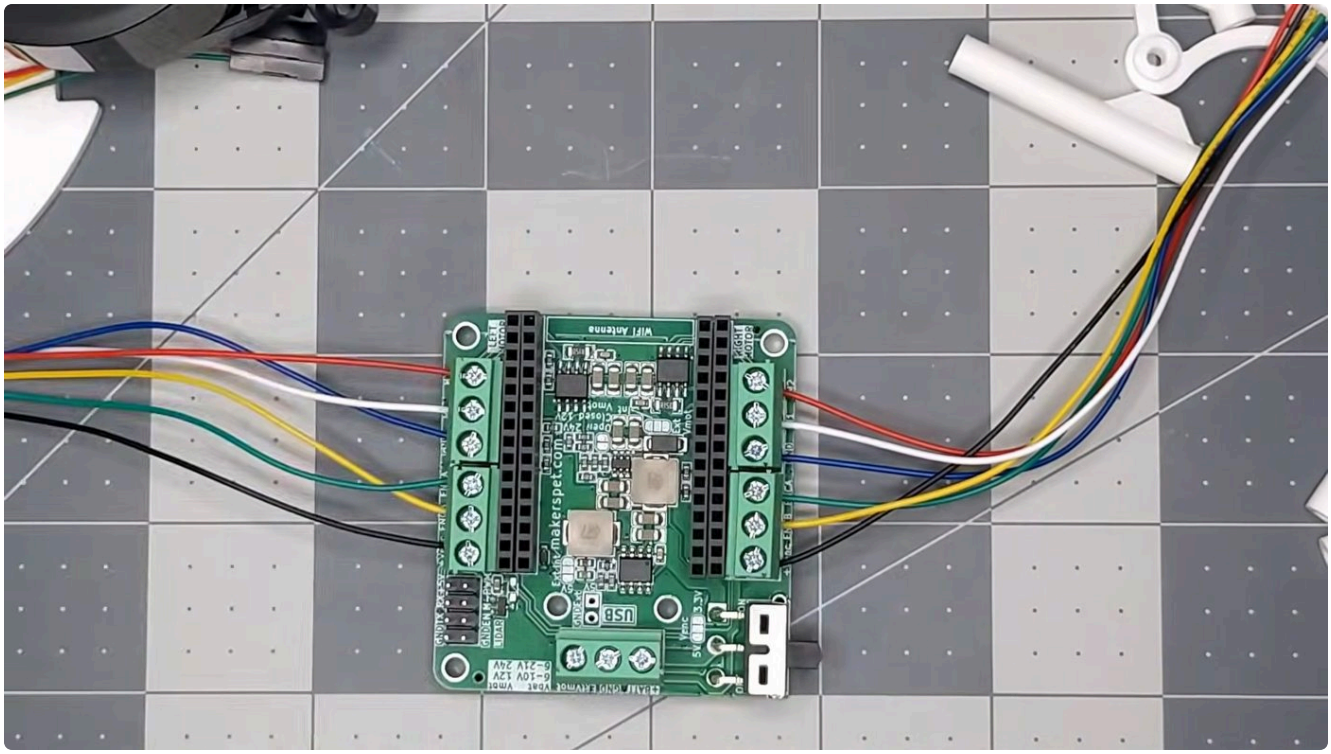
Tighten the terminal screw.

5. Connect the rest the same way, one motor cable per motor terminal block. Tug each wire gently to check it holds.

Both motor terminal blocks use the same order:

Terminal	M2	M1	GND	ENCA	ENCB	+Venc
Wire color	red	white	blue	green	yellow	black

> **NOTE**
> These are the colors of the cables shipping now. Restocked cables may differ; if so, go by the pin order on the plug and the terminal labels on the board.



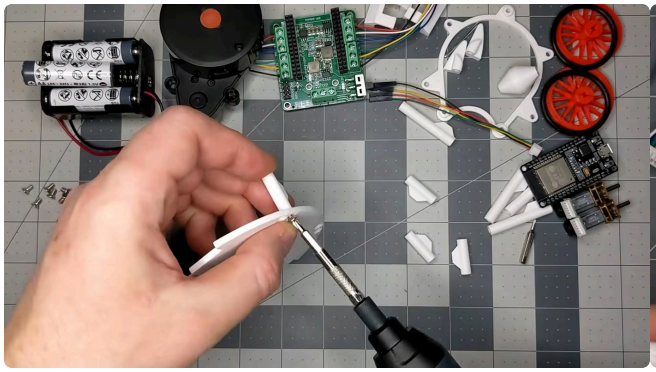
Both motor cables wired in.

> **WARNING**
> A loose screw terminal is the most common cause of a motor that "doesn't work", often because the stripped end is too short. If in doubt, strip longer than 8–10 mm (see Troubleshooting).

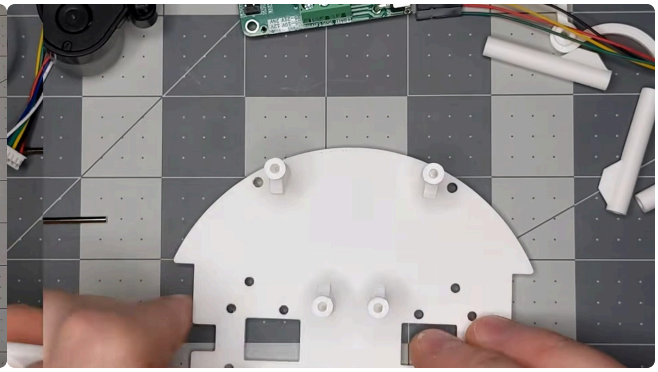
Build the chassis

Posts

1. Attach the four **board posts**, the shorter posts, to the base plate with **countersunk M3 screws**. Orient each post's tab as shown: grab the post and its tab by hand and twist it carefully into place.



Attach the board posts.

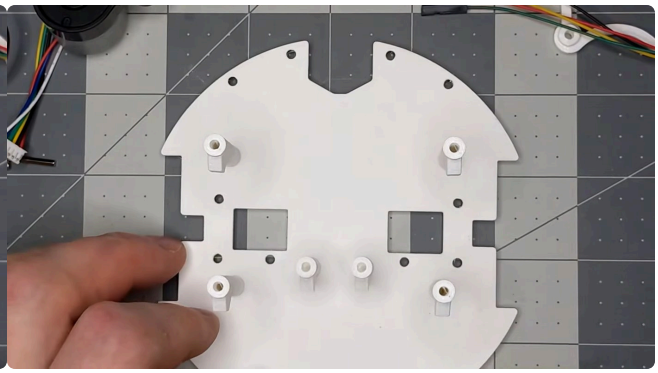


All four board posts in place.

2. Attach the four longer **LiDAR posts** the same way, oriented as shown.



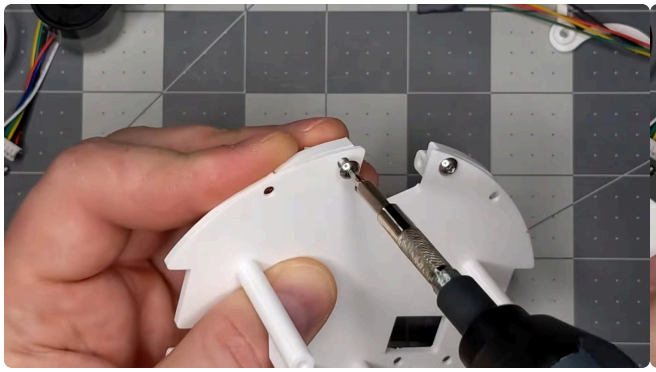
Attach the LiDAR posts.



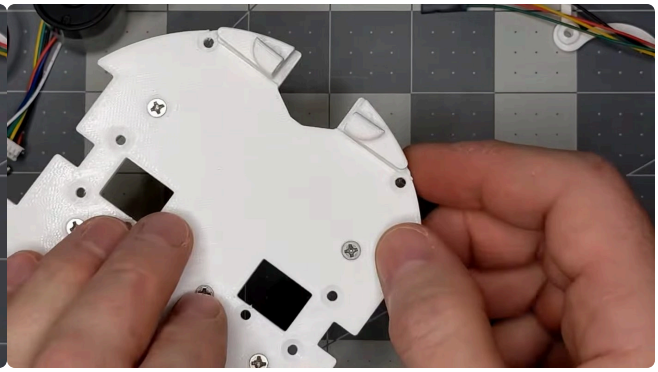
Board and LiDAR posts in place, oriented as shown.

Caster roller

1. Attach the two **caster wheel mounts** to the back of the base with **hex button M3 screws**, left slightly loose.



Attach the caster mounts.



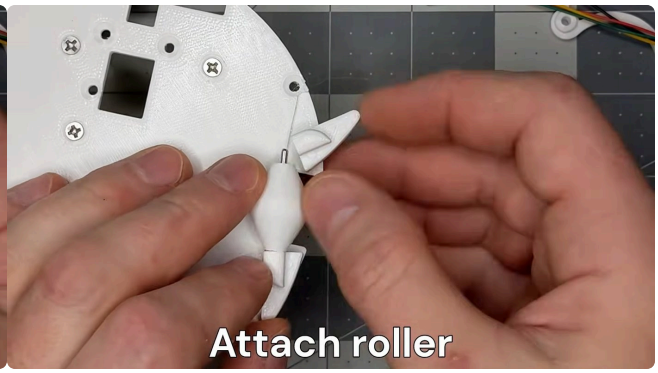
Caster mounts in place.

2. Push the metal shaft through the roller.
3. Place the shaft ends into the two caster mounts.
4. Tighten the caster mount screws.

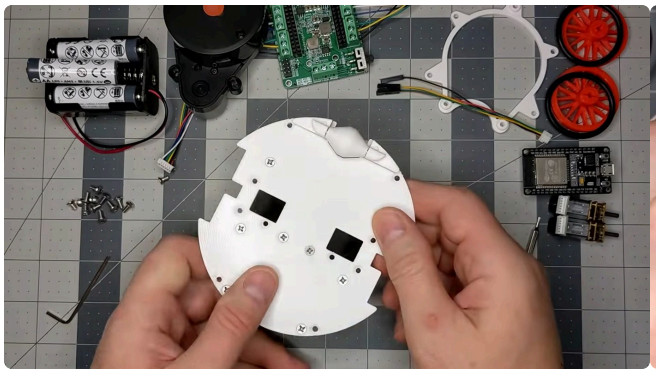
5. Press the roller against the table and roll the base back and forth until it spins freely.



Push the shaft through the roller.



Place the shaft ends into the caster mounts.



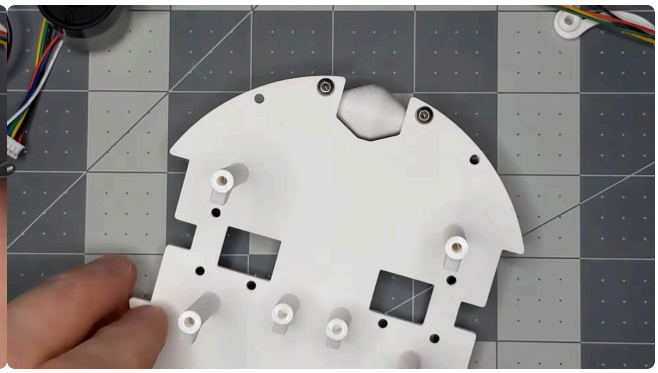
Roller in place, bottom view.



Tighten the caster mount screws.



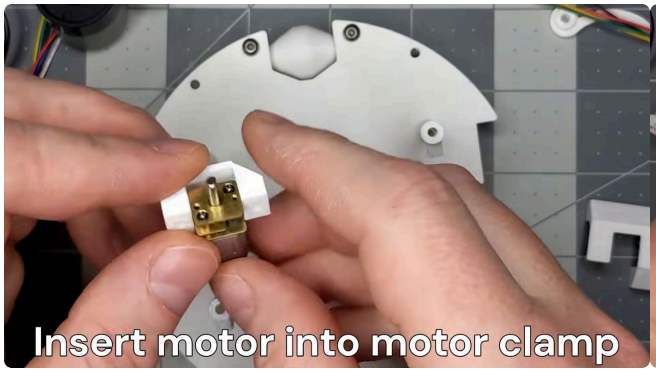
Roll the base to loosen the roller.



Posts and roller in place, top view.

Motors

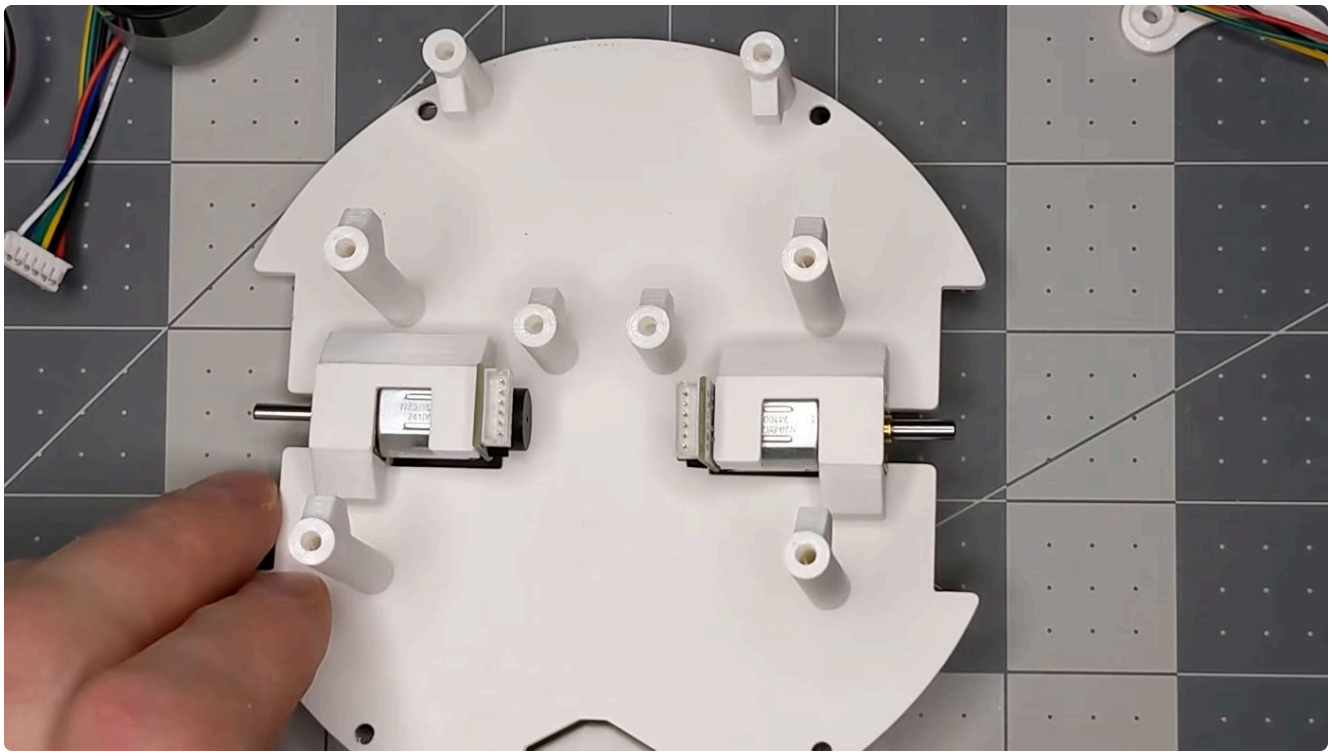
1. Insert each motor into its clamp, front face flush with the clamp.
2. Attach each motor to the base with **three countersunk M3 screws**, shafts pointing outward.
3. Don't overtighten; it can bend the plastic base.



Insert the motor into the motor clamp.



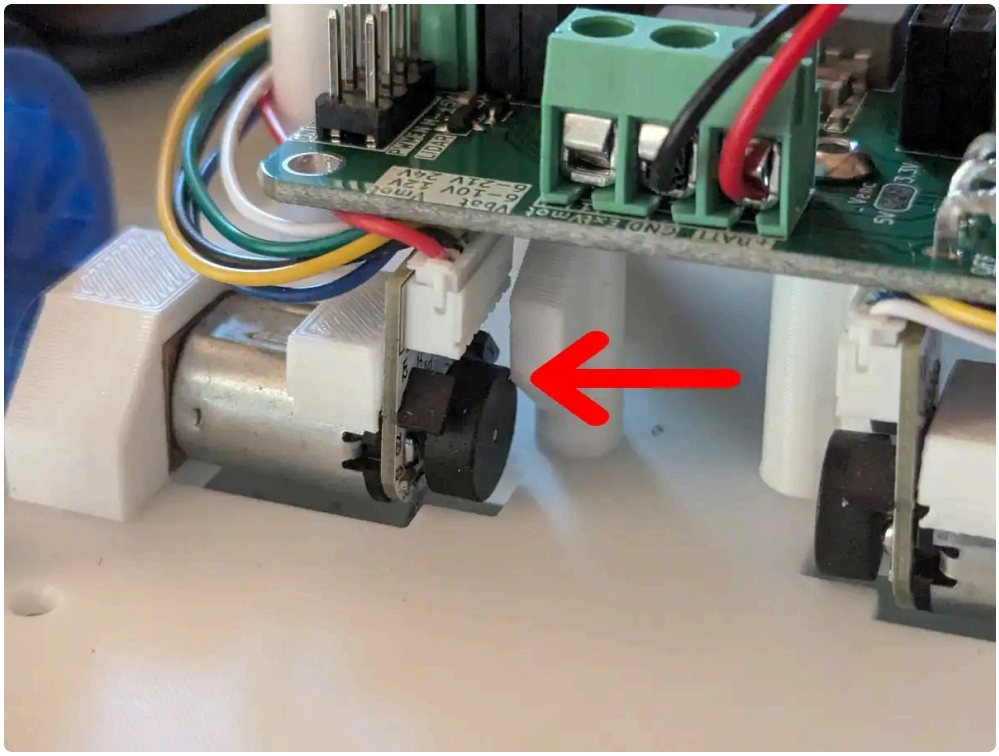
Screw the motor to the base; don't overtighten.



Both motors mounted, shafts pointing out.

SINCE THE VIDEO WAS RECORDED

Check that no board post tab touches a motor. A post pressing on a motor can block it, and a blocked motor can burn out when powered. If one does, twist the post to turn its tab away.



Make sure the board post tab doesn't touch the motor.

SINCE THE VIDEO WAS RECORDED

Bent plastic, including the base, can rub against the motor gears or encoder disk, stall the motor and burn it out. Tighten the motor screws just until snug.

Wheels

1. Make sure each wheel is free of debris.
2. Press a tire into each wheel's groove all the way around.

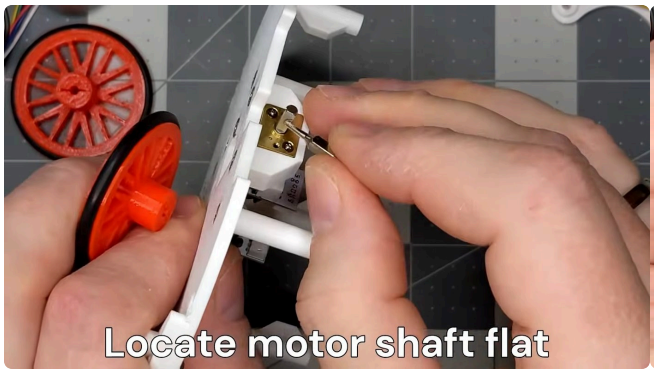


Put the tire on the wheel.



Press the tire into the groove with your fingers.

3. Find the flat on the motor shaft and the matching flat in the wheel's hub.

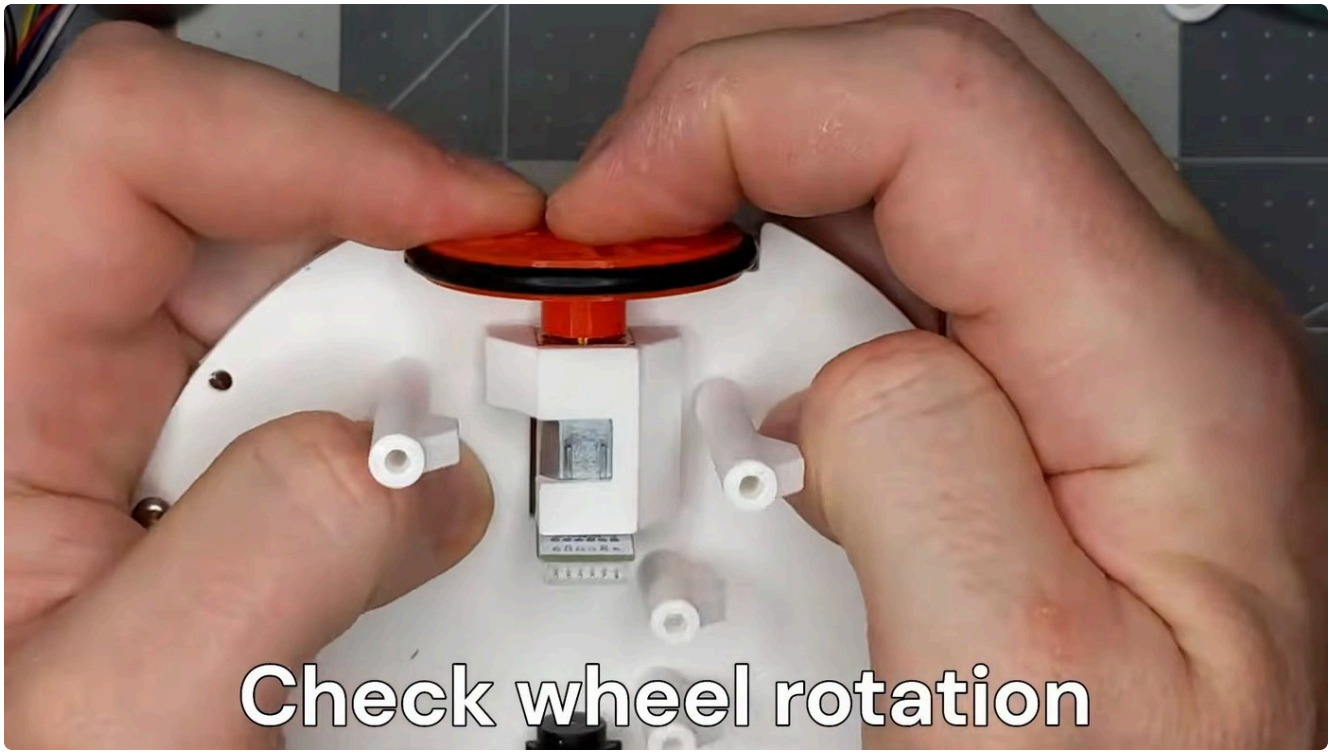


The flat on the motor shaft.



The flat in the wheel hub.

4. Line up the flats and press the wheel fully onto the shaft with both hands.
5. Spin each wheel by hand to check it doesn't rub.

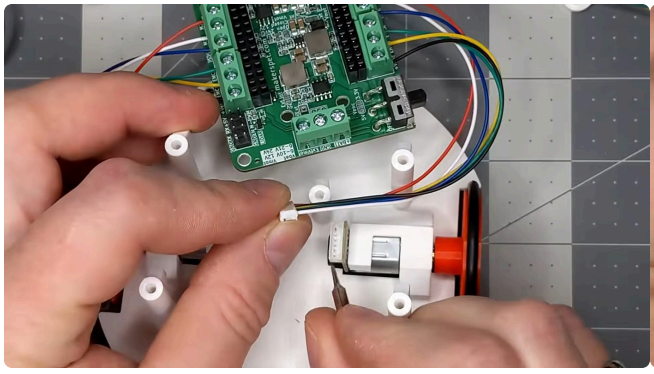


Press the wheel fully onto the shaft with both hands. Check wheel rotation.

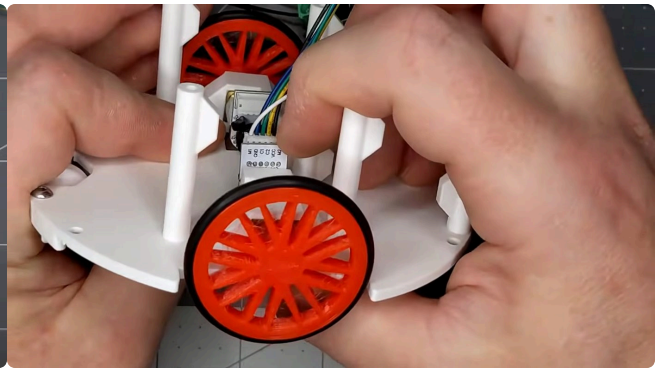
> **SINCE THE VIDEO WAS RECORDED**
> Grit or plastic shavings in the gearbox can jam the gears and burn out the motor. That's why you clean the wheels and hubs first.

Mount the board

1. Line up each motor cable's plug with the connector on the back of the motor and press it in gently. If it won't go in, check the alignment rather than forcing it.

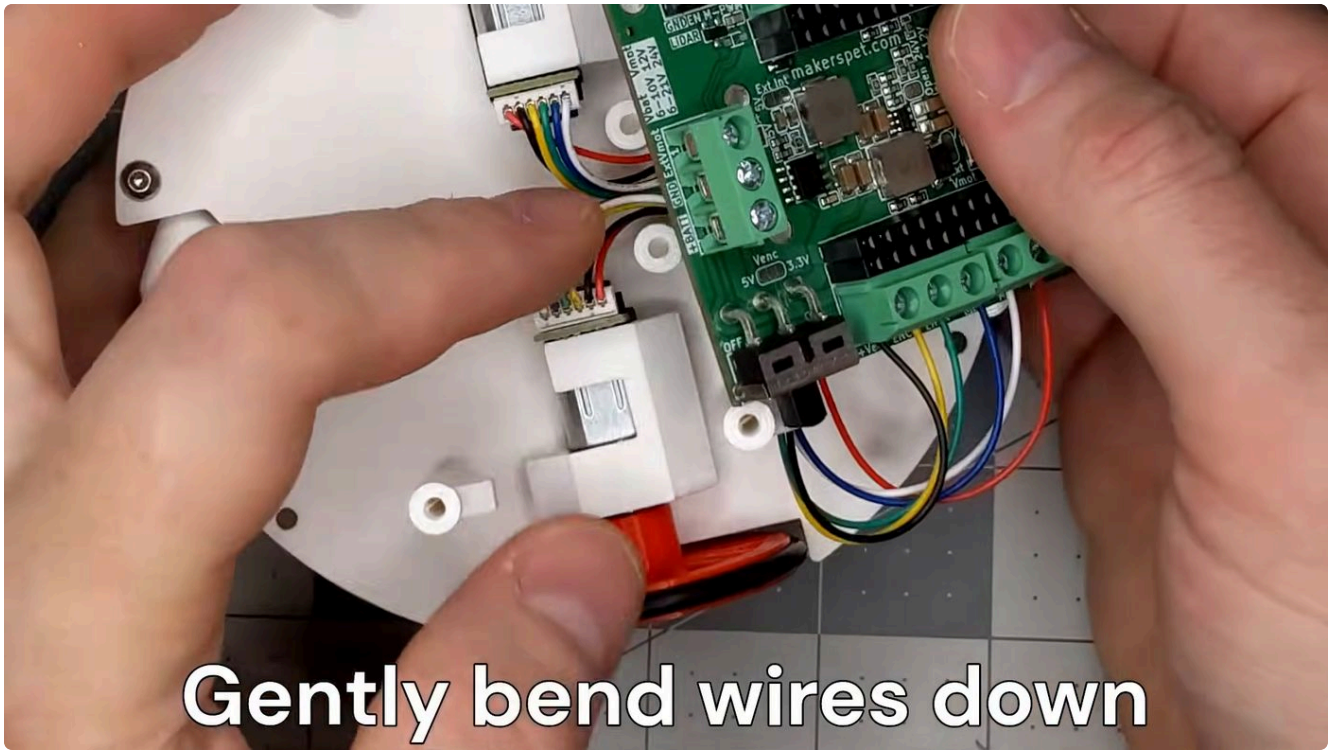


Align the plug with the motor's receptacle.



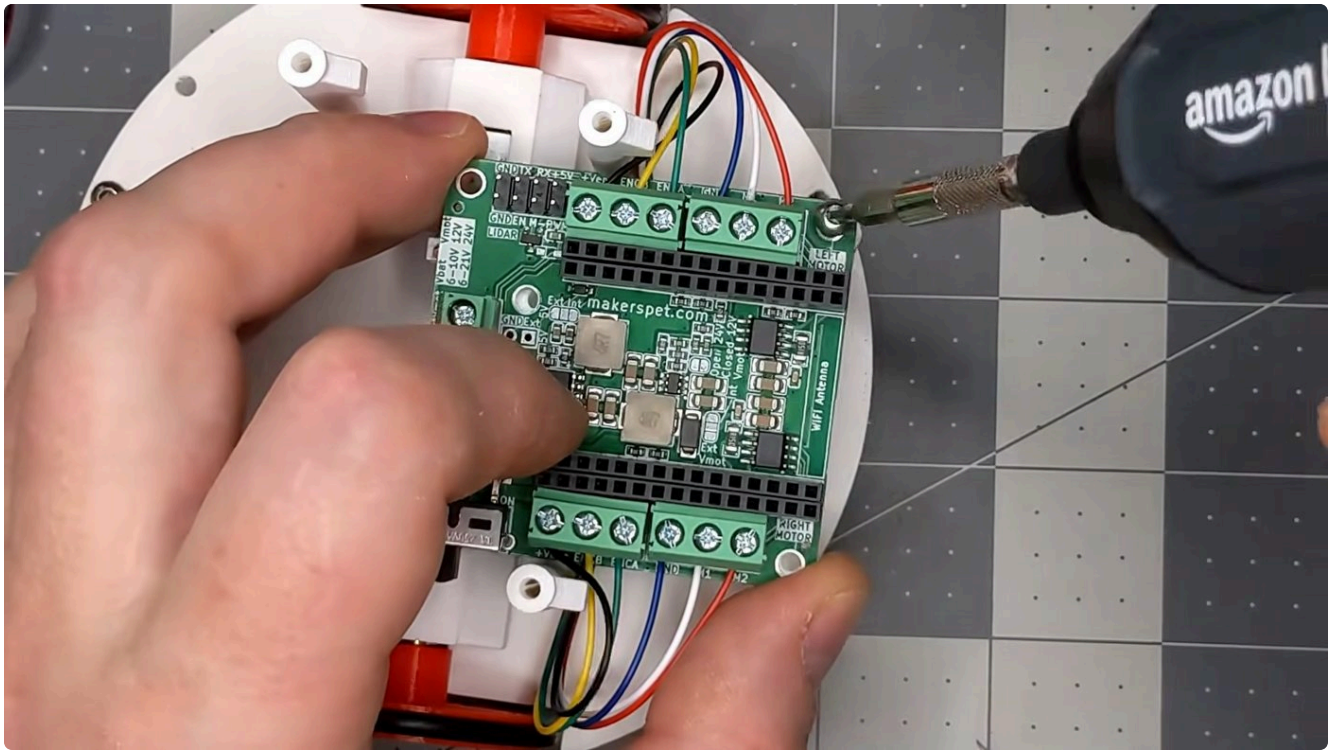
Press the plug in gently.

2. Route the cables under the board and gently bend the wires down at the motor plugs.



Route cables under the board. Bend the wires down at the motor plugs gently.

3. Attach the board to the posts with **four hex button M3 screws.**

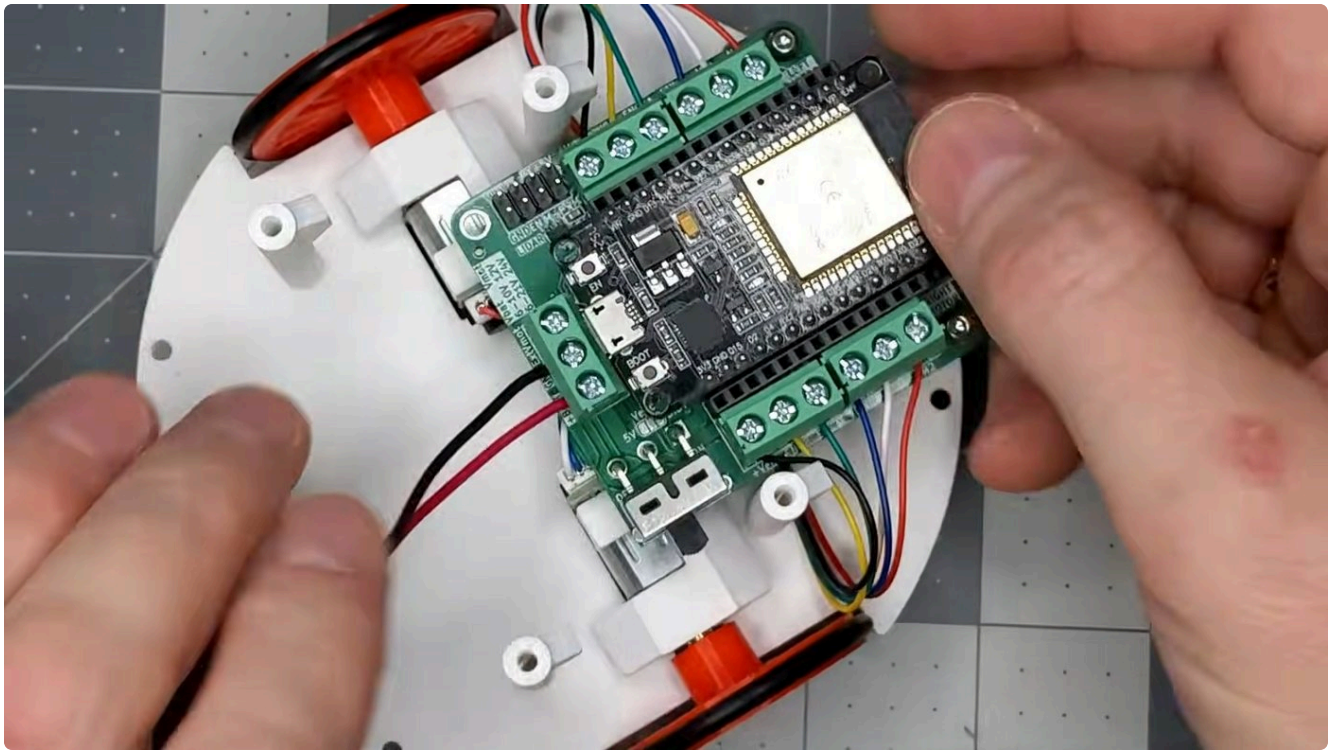


Attach the board to the posts.

> **TIP**
> Before plugging in each cable, check the motor's connector: its plastic housing can slide up off its pins. See Troubleshooting.

Connect the battery

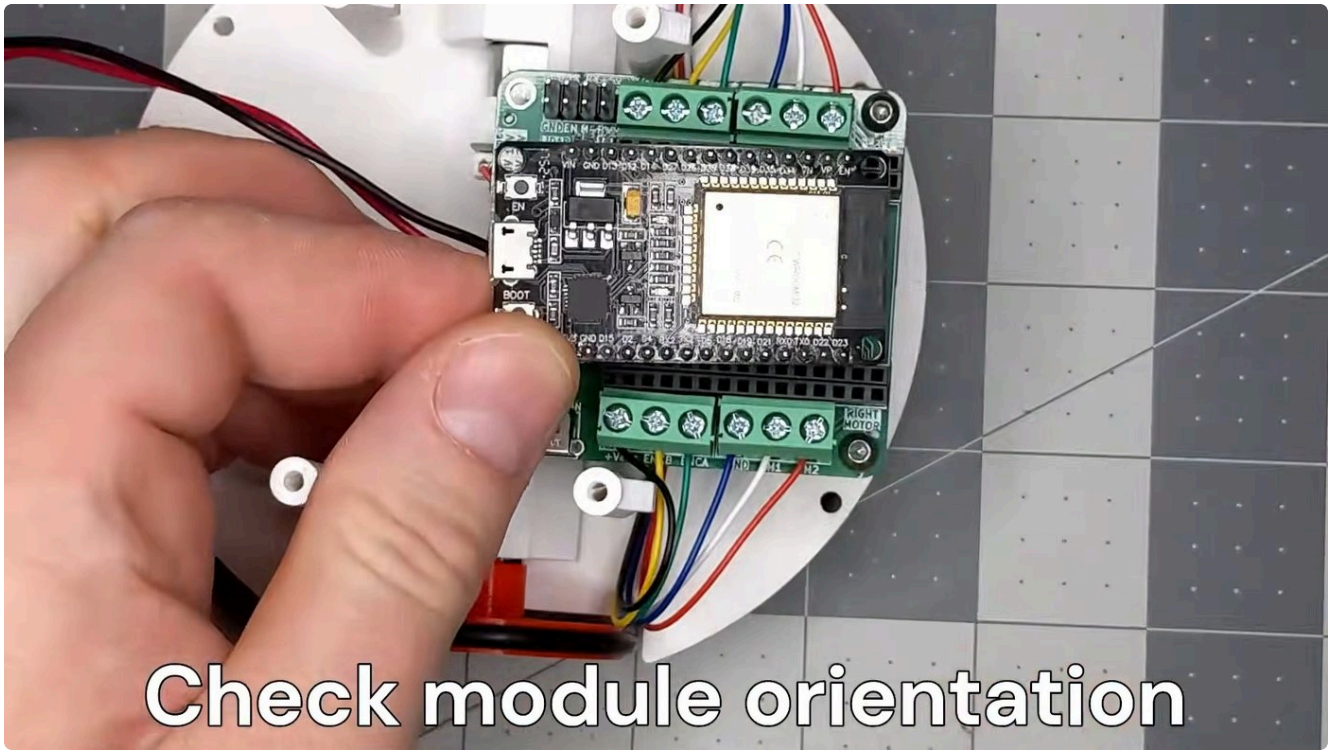
1. Connect the battery case wires to the `+BAT`, `GND`, `ExtVmot` screw terminal: red to `+BAT`, black to `GND`.
2. Check the polarity **before** switching on. Reversed polarity will burn the board.
3. This time, **don't** fold back the stripped conductors; insert them straight.



Battery wires in the screw terminal: red to +BAT, black to GND.

Insert the ESP32 module

Plug the **Maker's Pet ESP32-E 30-pin dev kit module** into the board's socket with its USB connector toward the **battery screw terminals**. Check before pressing it fully in: backwards, it gets the wrong power pins.

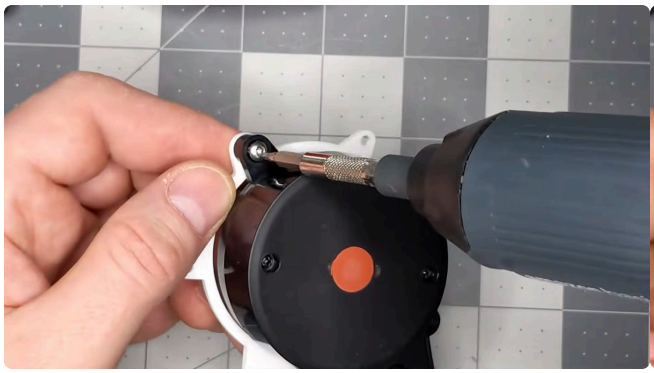


Insert the ESP32 module. Check the orientation.

> **NOTE**
> I test and support the kit with the Maker's Pet ESP32-E, which is compatible with the original 30-pin ESP32 DOIT DevKit v1. Clones from other makers might work, but aren't supported.

Prepare the LiDAR

1. Attach the **LiDAR skirt** to the LiDAR with **three M3 hex button screws**, leaving no gaps.



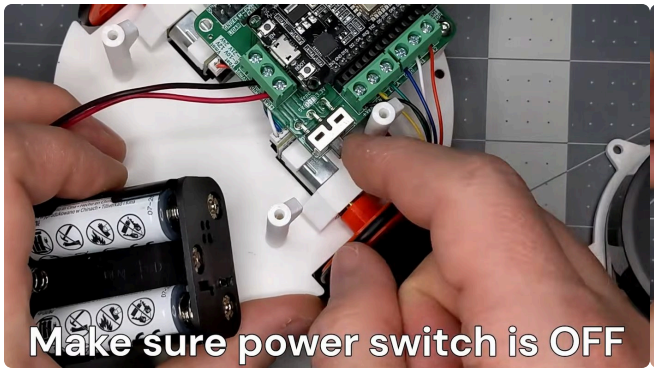
Attach the LiDAR skirt.



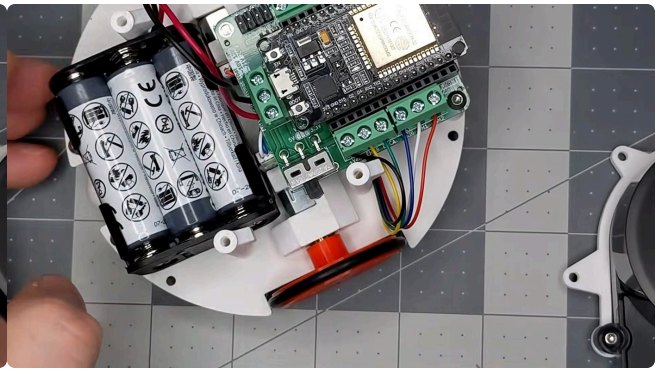
No gaps between skirt and LiDAR.

Batteries

1. Make sure the board's power switch is **OFF**.
2. Put 6 × AA batteries into the case and place it on the base as shown.



Make sure the power switch is OFF.



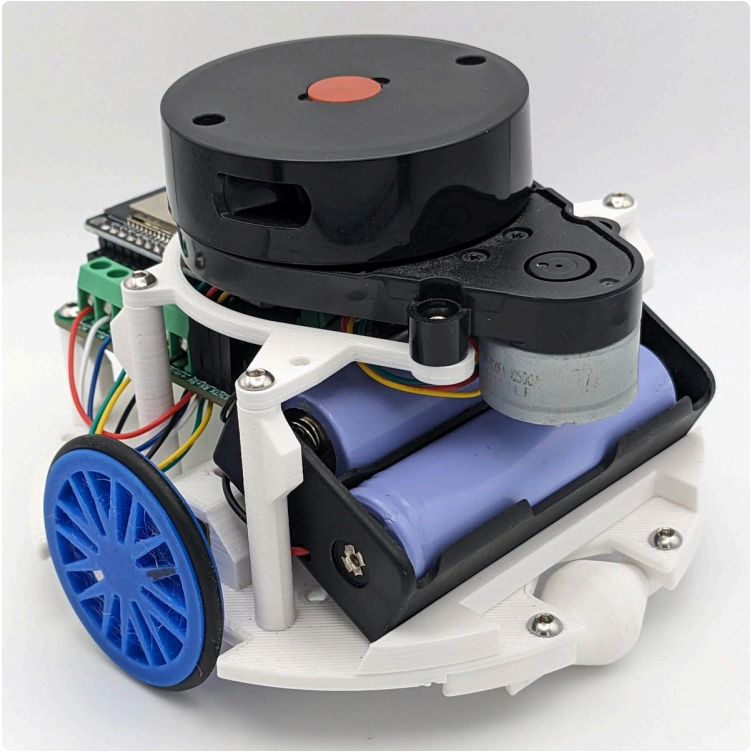
Place the battery case.

SINCE THE VIDEO WAS RECORDED: 18650 RECHARGEABLE BATTERY HOLDER MOD

The BDC-30P is designed for **non-rechargeable alkaline** batteries (6 × AA) and has no protection for rechargeable cells, which can be dangerous when mishandled. **Using rechargeable batteries is at your own risk.**

If you still use a 2-cell 18650 holder (it fits the 120 mm base):

- Use **protected** cells only (short-circuit, overcurrent, overcharge and overdischarge protection), connected **in series**.
- Charge the cells out of the holder in your own charger.
- Choose cells rated for around 3 A continuous and 5 A peak.
- Mount the rear posts in the **outer spare holes** of the base.
- Mount the battery holder backstop behind the battery in the two spare holes, with two M3 countersunk screws.
- Use the **wider LiDAR skirt** that matches the new rear post positions.



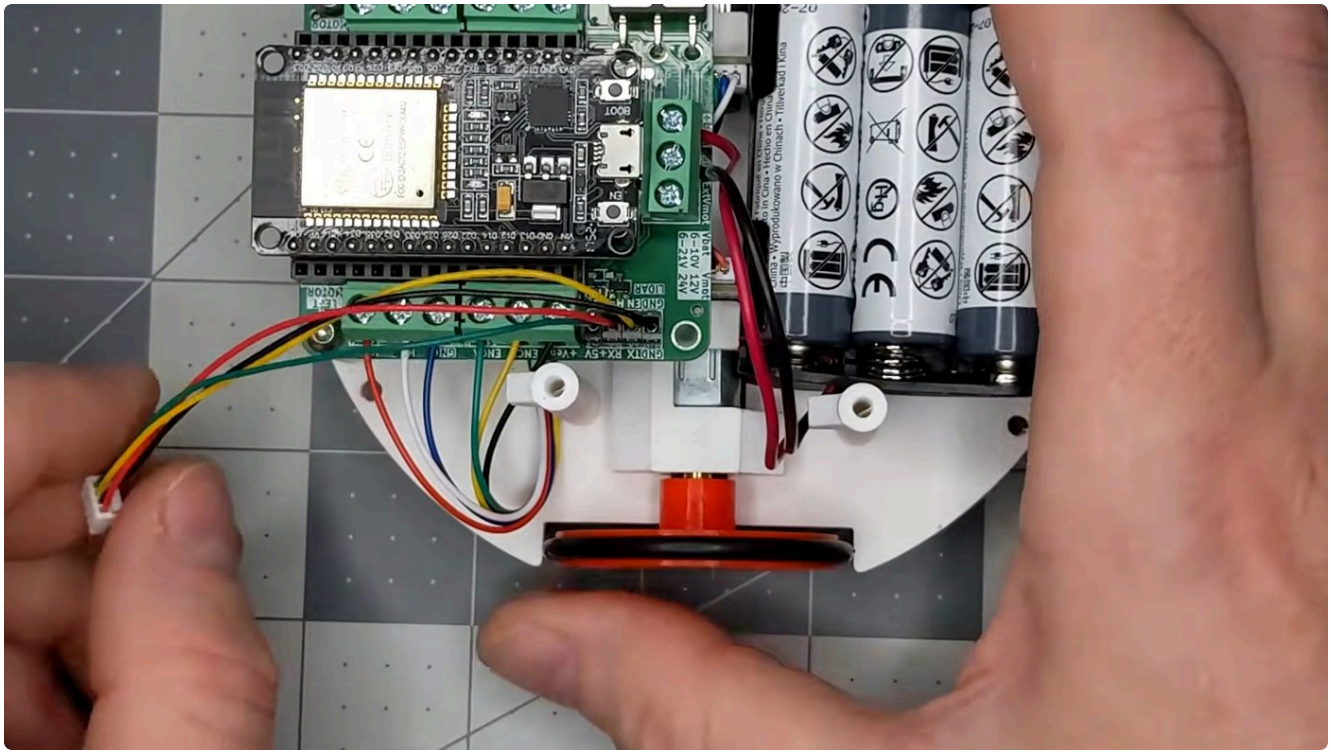
The 18650 battery holder mod.

Connect the LiDAR

1. Connect the LiDAR breakout cable wires to the board's `LIDAR` header:

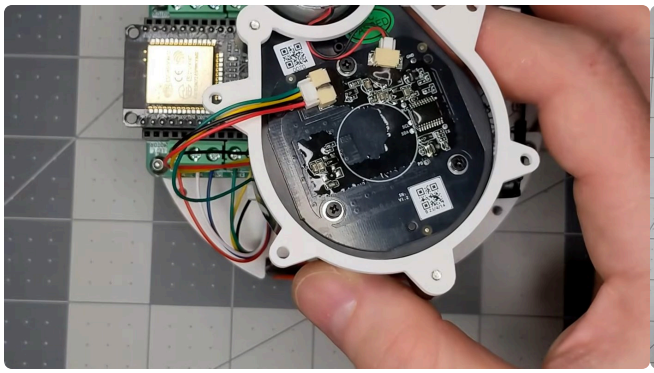
Header pin	GND	TX	PWM	+5V
Wire color	black	yellow	red	green

Restocked cables may use other colors; go by the pin labels.

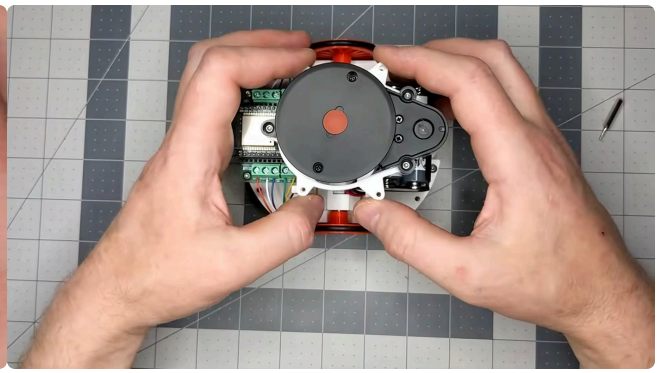


Connect the LiDAR wires to the board.

2. Line the plug up with the LiDAR's connector and push it in gently until it clicks.
3. Fold the wires neatly and place the LiDAR on the robot.



Align the plug and push it in gently until it clicks.

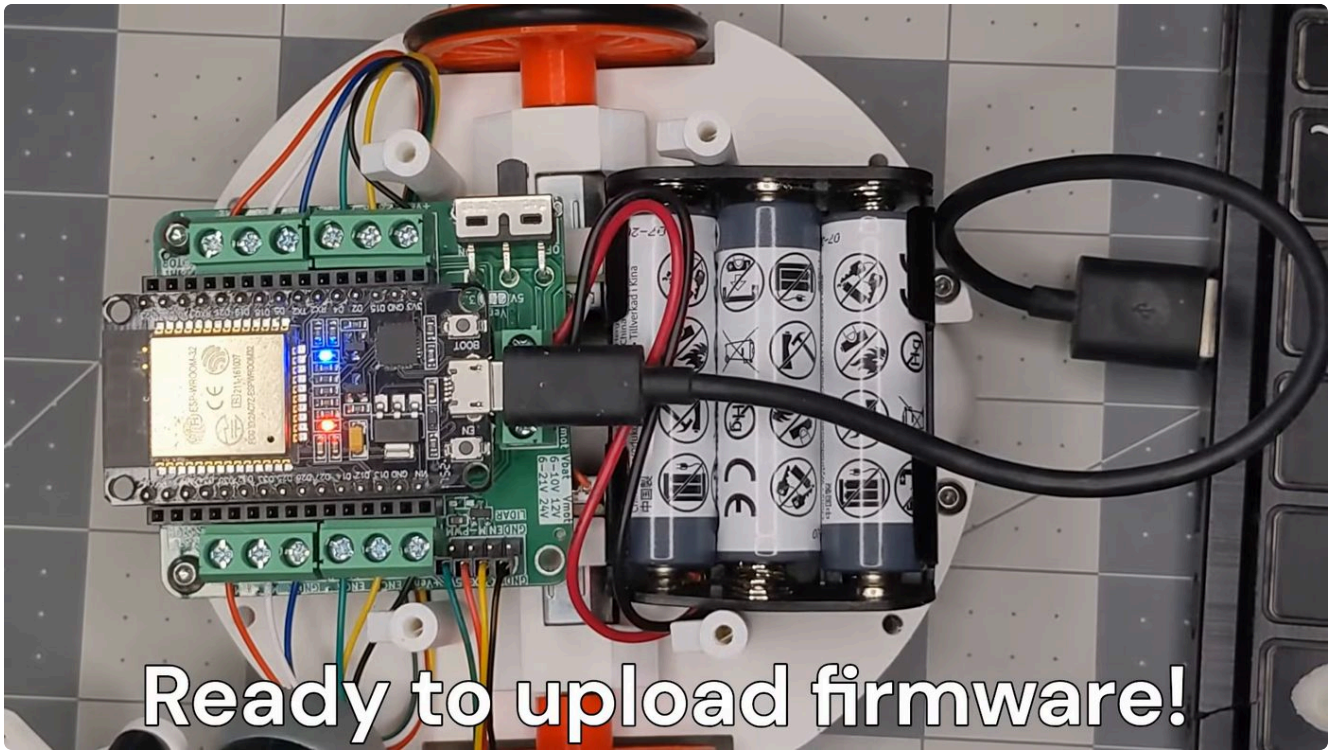


Fold the wires and place the LiDAR.

Don't fix the LiDAR in place until after the firmware upload.

Connect to your computer

Place the LiDAR carefully aside on the table, still connected to the board, to reach the ESP32-E's USB connector. Connect the ESP32 to your computer with a USB cable. The robot is ready for the firmware upload.

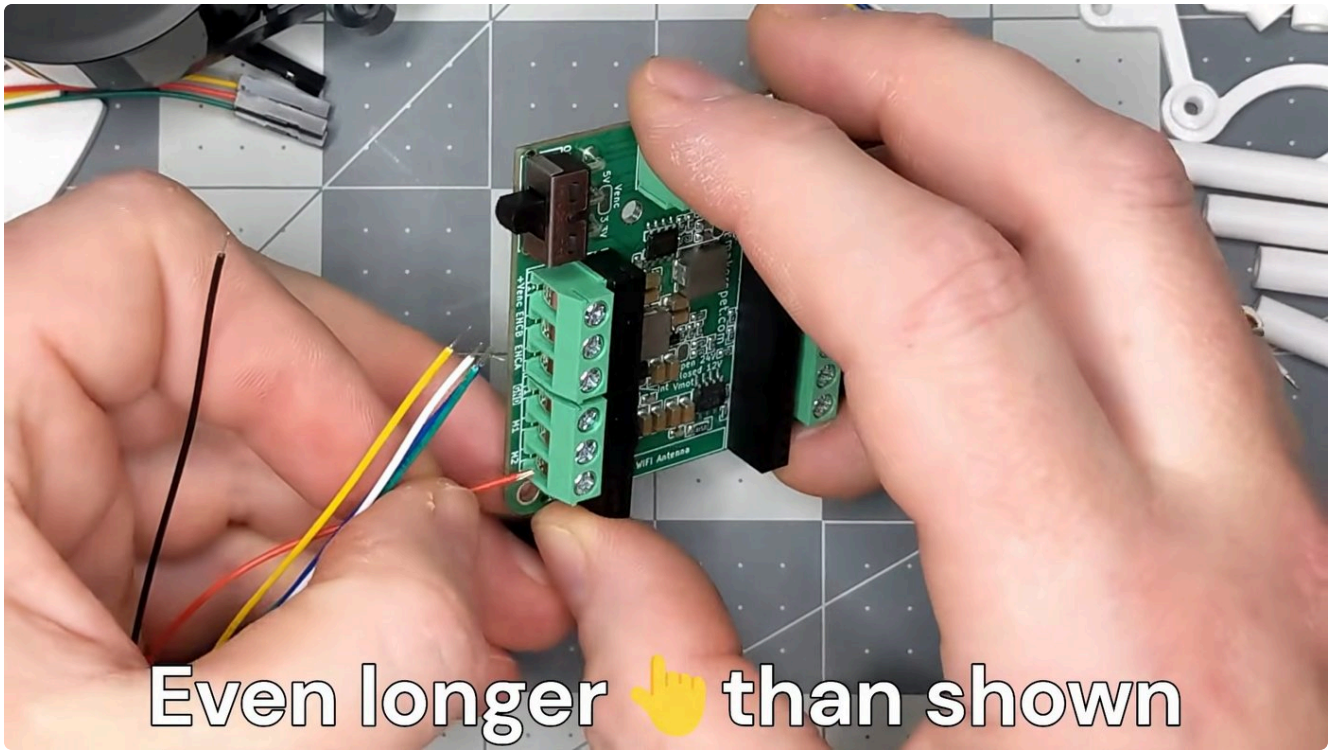


Ready to upload the firmware!

> **TIP**
> If your computer doesn't see the ESP32, try another USB cable. Some carry power only.

Troubleshooting tips

A screw terminal connection fails. Usually the stripped end is too short. Strip it even longer than in the video.



Strip wires even longer than shown.

A screw won't hold in plastic. Overtightening stripped the thread. Push a small wood splinter into the hole and drive the screw back in.



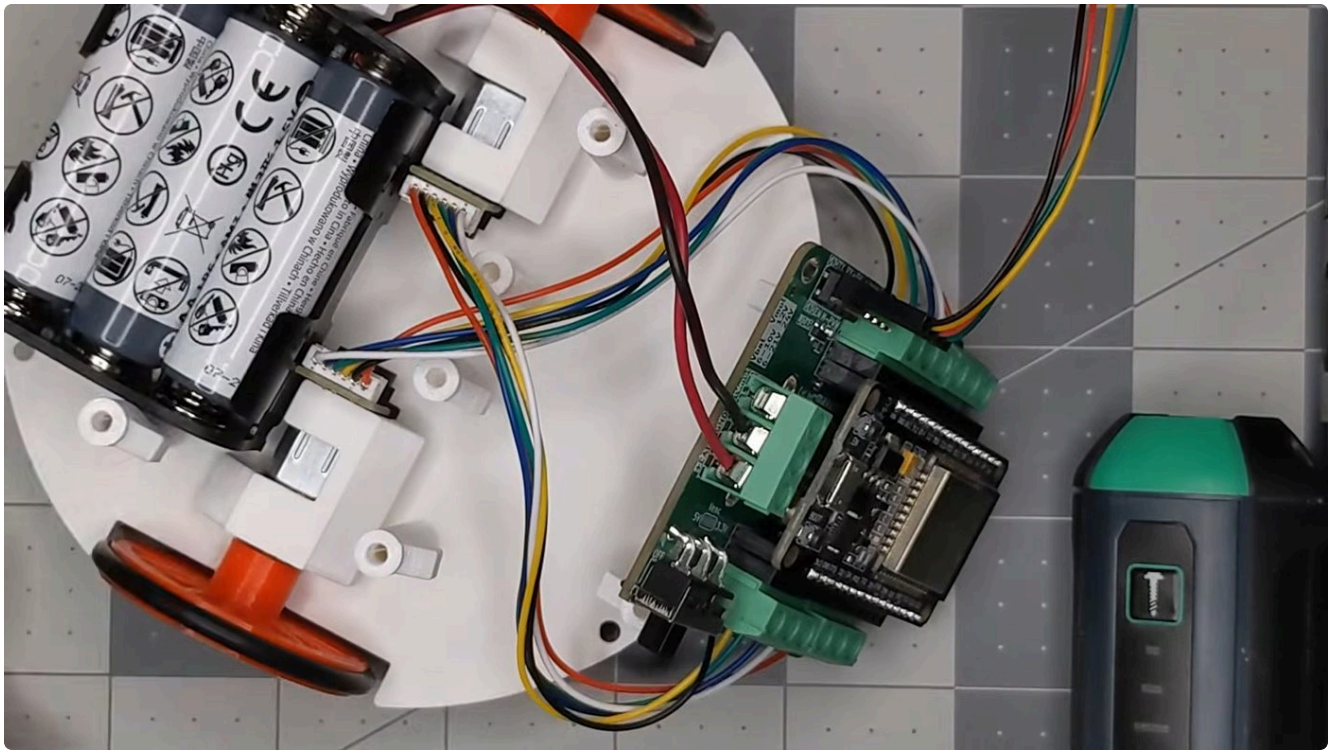
Insert a wood splinter to fix the thread.

The motor connector housing has slid up. The plastic housing can slide up, off the connector pins. Press it back down fully before plugging in the cable.



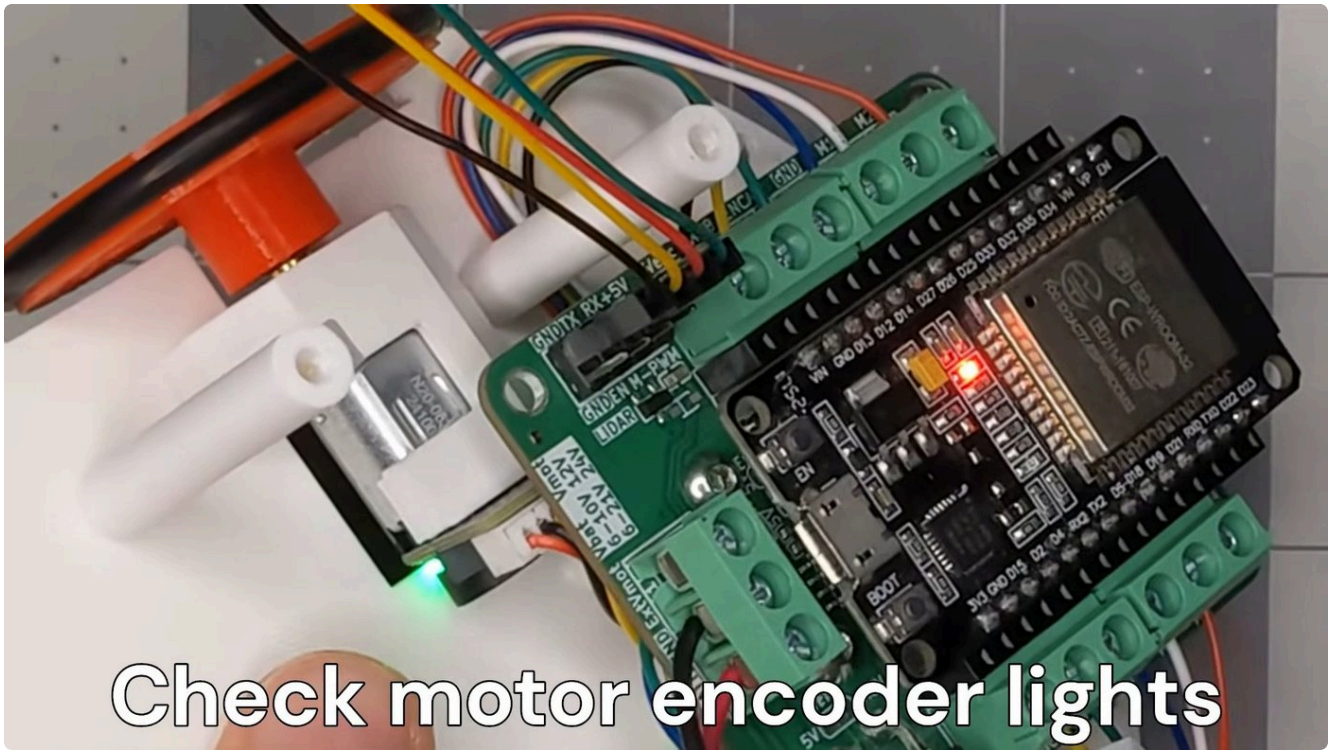
The connector housing slid up fully; press it back down.

One motor works, the other doesn't. Swap the two motors' connections. If the problem moves sides, the motor (or its cable) is at fault; if it stays, check that side's screw terminals.



Swap the two motors' connections.

Check the green encoder lights. With the battery on, each motor's encoder board should show a green light. If one is off, that encoder has no power: check the battery, the switch, and the motor-to-cable and cable-to-board connections.



Check the motor encoder lights.

SINCE THE VIDEO WAS RECORDED: MORE MOTOR CHECKS FROM THE GUIDE

- No **board post tab touches a motor** (see *Motors*).
- The **power switch is ON**, the battery is connected and not low. The ESP32's power LED should light.
- The motor wires are in the **correct screw terminals**, none broken or loose.
- **No debris in the gearbox**; it can seize the gears and burn out the motor.
- **Motor screws not overtightened**; bent plastic can catch the gears or encoder disk.
- Gently spin the magnet to turn the motor's core shaft. It should turn freely; if it doesn't, the motor or its gearbox may be jammed.
- Encoder sensors can get bent and touch the encoder wheel. Carefully bend them back and check the encoder magnet turns freely.
- No cable rubs on a motor's encoder magnet.

Checks that need the firmware and ROS2 running, such as a motor that spins only at full speed or in one direction, are in the bring-up chapter.

SINCE THE VIDEO WAS RECORDED: BATTERY CONNECTED BUT NO POWER

- Check the board's ON/OFF switch.
- Check the battery wires are in the right terminals with good contact; measure the voltage at the board's power terminal.
- Check the **polarity**. Reversed polarity will burn the board.
- Check the batteries make good contact in the holder (wiggle tight ones) and measure each one.

Next, set up the software and upload the firmware.



The finished robot. Happy building!

2. Set Up the Software on a Windows PC

Video: Step by Step Arduino Self-Driving Robot software setup

This chapter has two halves. First, install the Arduino IDE with the ESP32 toolchain and the robot firmware, and check that the firmware compiles. Second, install the PC side that controls the robot: ROS2 runs on Linux, so you add WSL2, Docker Desktop, an X server (to see Linux GUI programs) and PowerShell, then download the robot software image.

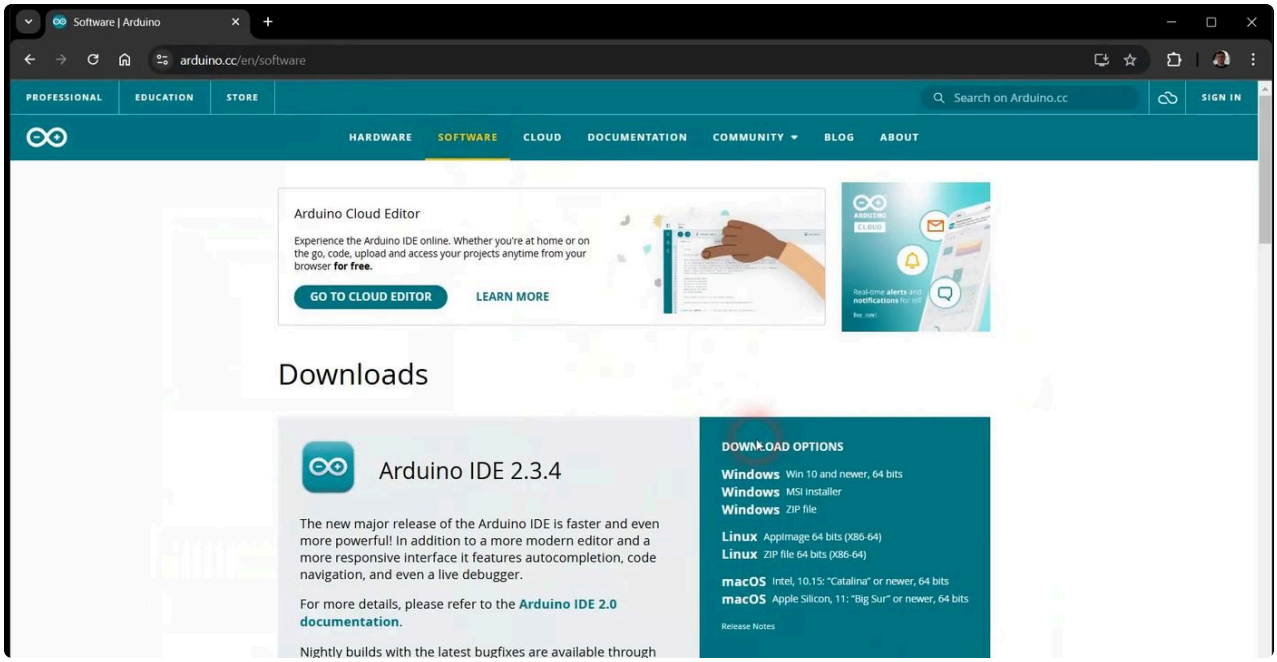
You need a Windows 10 (or later) PC with at least 8 GB of RAM and an internet connection, but not the robot. Expect large downloads and a couple of reboots.

> **NOTE**
> On Ubuntu Linux, skip to Chapter 8 instead. For simulation only (Chapter 6), skip the Arduino part: install WSL2, Docker Desktop, the X server and PowerShell, and download the Docker image.

> **WHICH ARDUINO IDE?**
> Both Arduino IDE 2.x and 1.8.19 work. I recommend **1.8.19**: it's tried and true, and frozen, so it won't change under you, while 2.x has given me some headaches. This chapter covers 2.x first, then 1.8.19.

Install Arduino IDE 2

1. On arduino.cc, open **Software** and under **Arduino IDE 2.x** download the first **Windows** option (Win 10 and newer, 64 bits). The video used Arduino IDE 2.3.4.



Arduino IDE 2 download options.

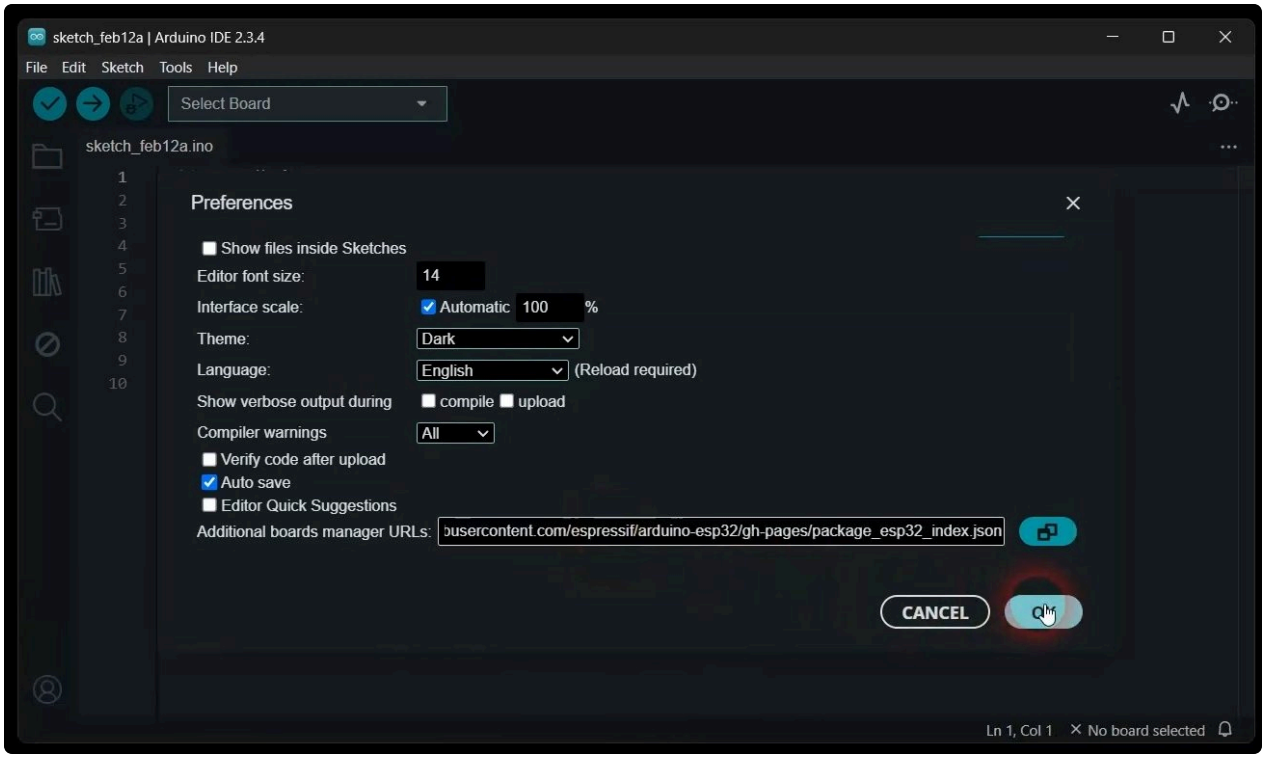
2. Run the installer, accept the defaults, and launch the IDE. If Windows Firewall asks, click **Allow**.

Add the Espressif ESP32 toolchain

The robot's brain is an ESP32, so the IDE needs Espressif's ESP32 compiler and SDK.

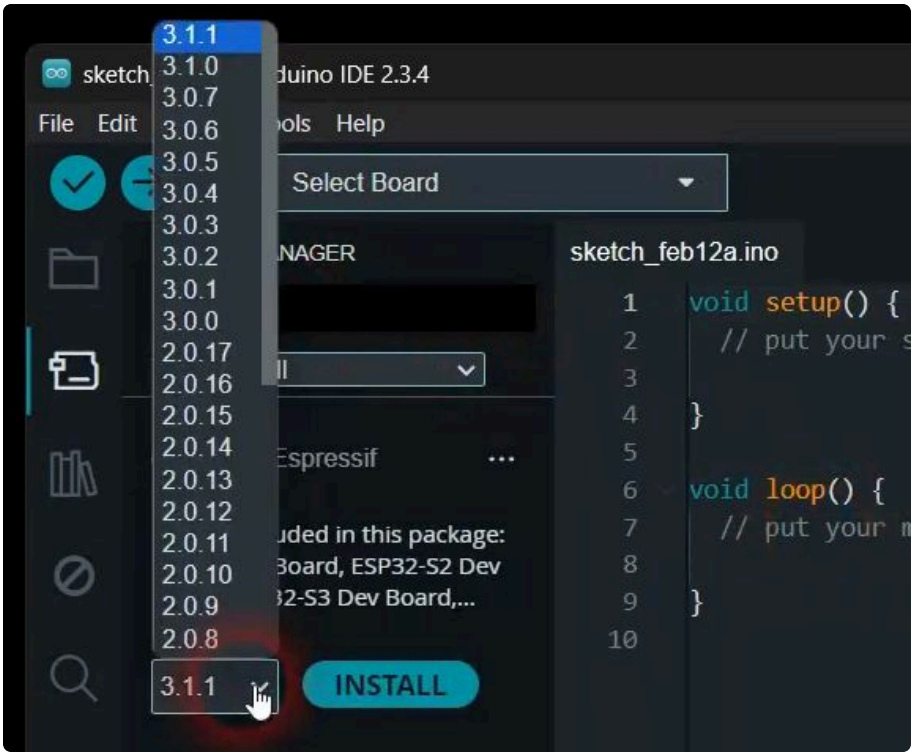
1. Go to **File → Preferences**. In **Additional boards manager URLs**, paste this URL and click **OK**:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
```



File → Preferences: paste the Espressif URL and click OK.

2. Go to **Tools → Board → Boards Manager**. 3. Search for `Espressif`. For **esp32 by Espressif Systems**, select version **2.0.17**, not the newest, and click **INSTALL**. Then close the IDE.



Select version 2.0.17.

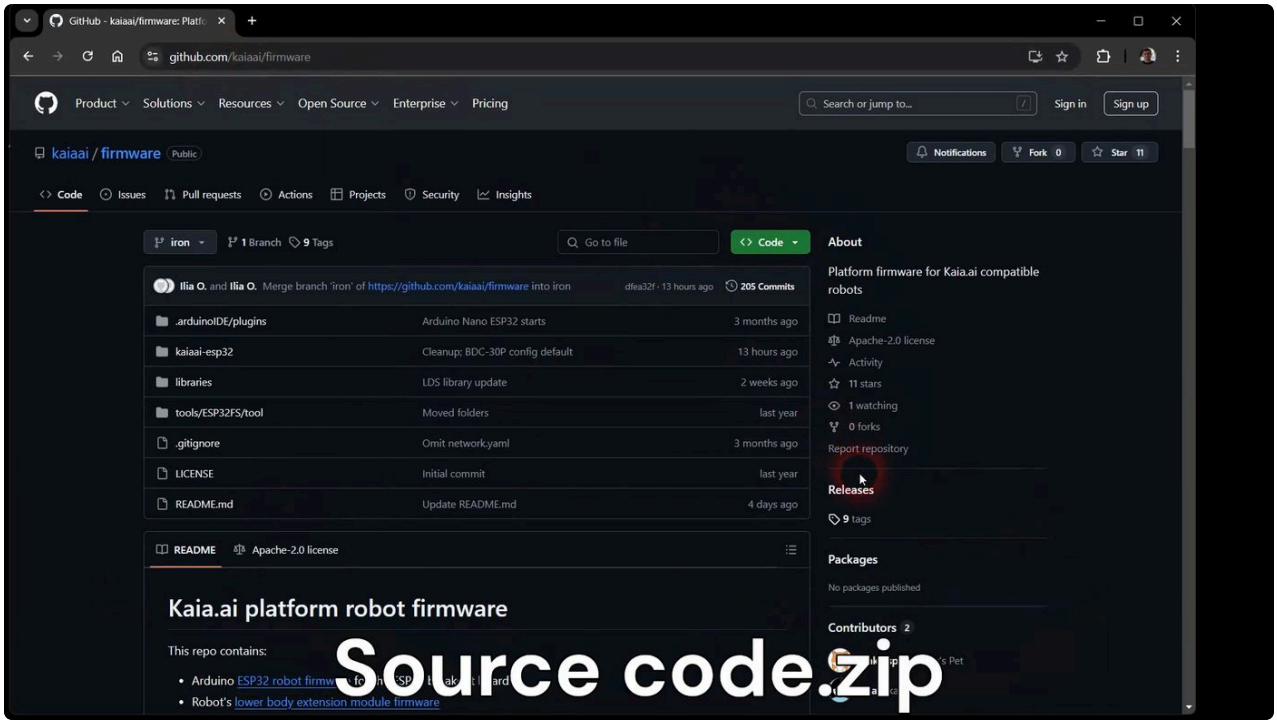
SINCE THE VIDEO WAS RECORDED

Stay on ESP32 board package **2.0.17**:

- Not **2.0.15**: its HardwareSerial is broken, so the firmware compiles but the LiDAR returns no data. 2.0.16 or newer works; 2.0.17 is recommended.
- Not **3.x**: it removed `ledcAttachPin` / `ledcSetup`, which the firmware uses.

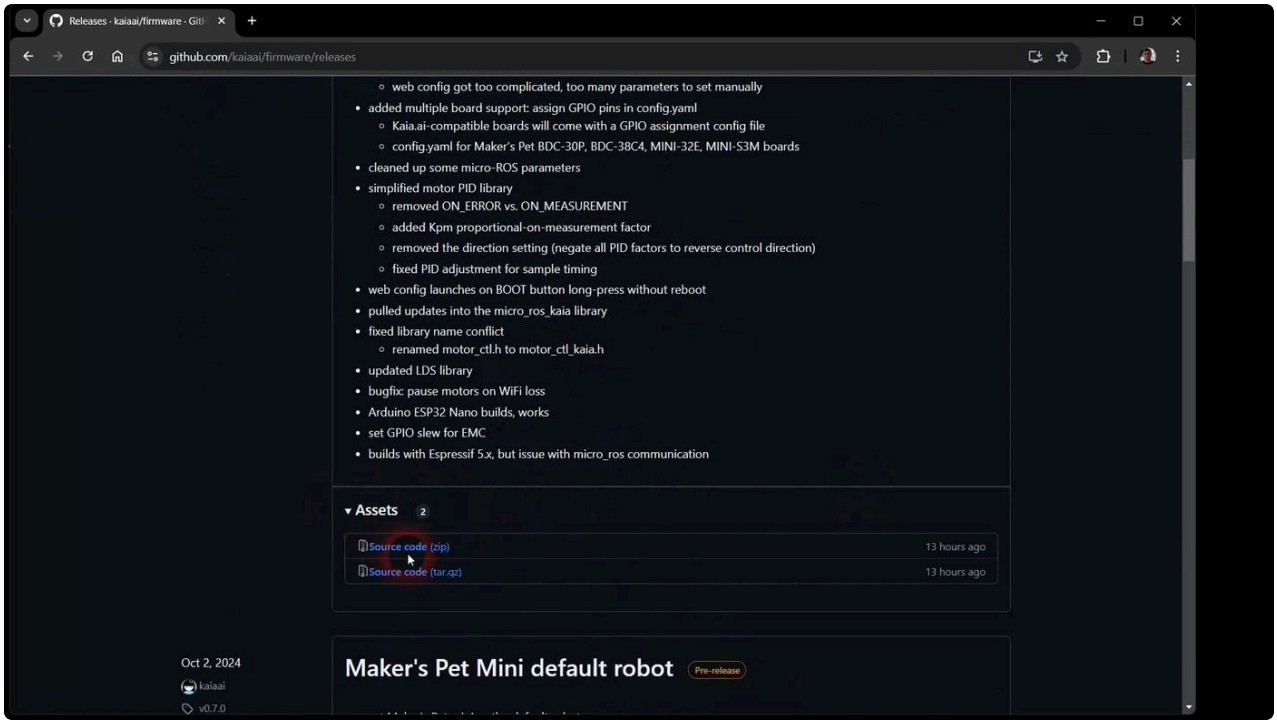
Download the robot firmware project

1. Browse to github.com/kaiaai/firmware and click **Releases**.



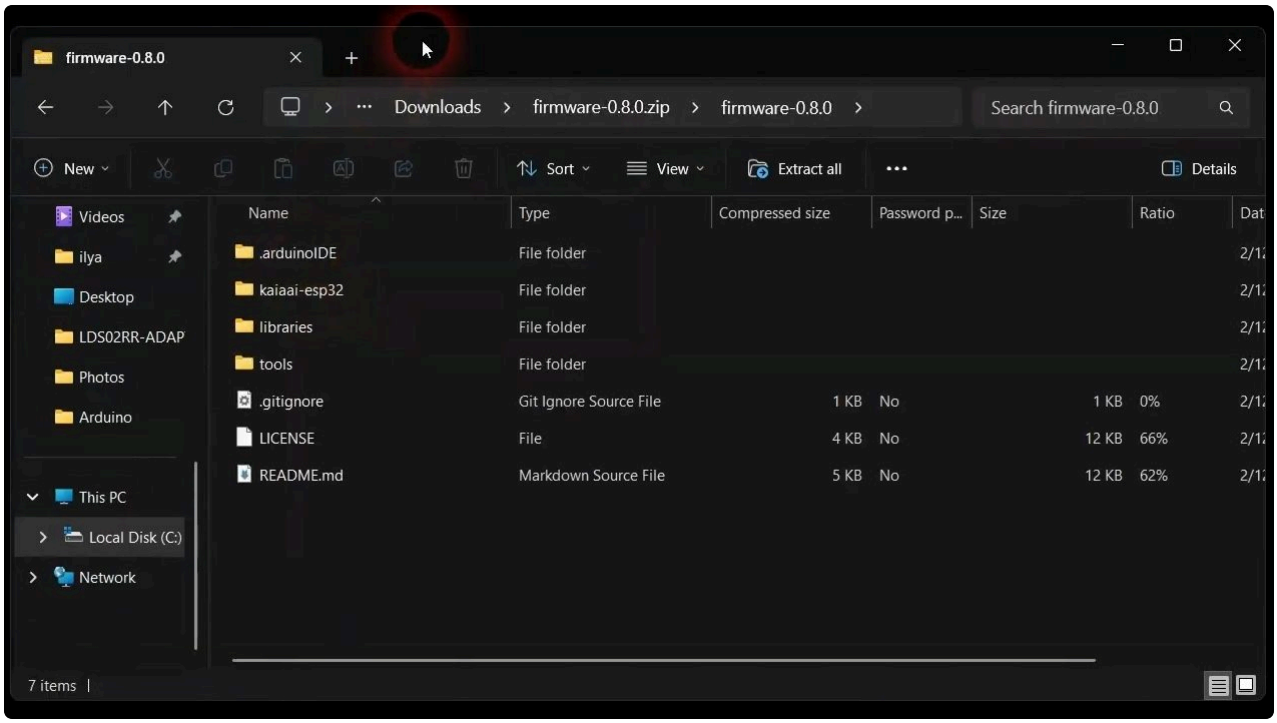
The kaiaai/firmware repository; Releases is on the right.

2. In the latest release, expand **Assets** and download **Source code (zip)**.



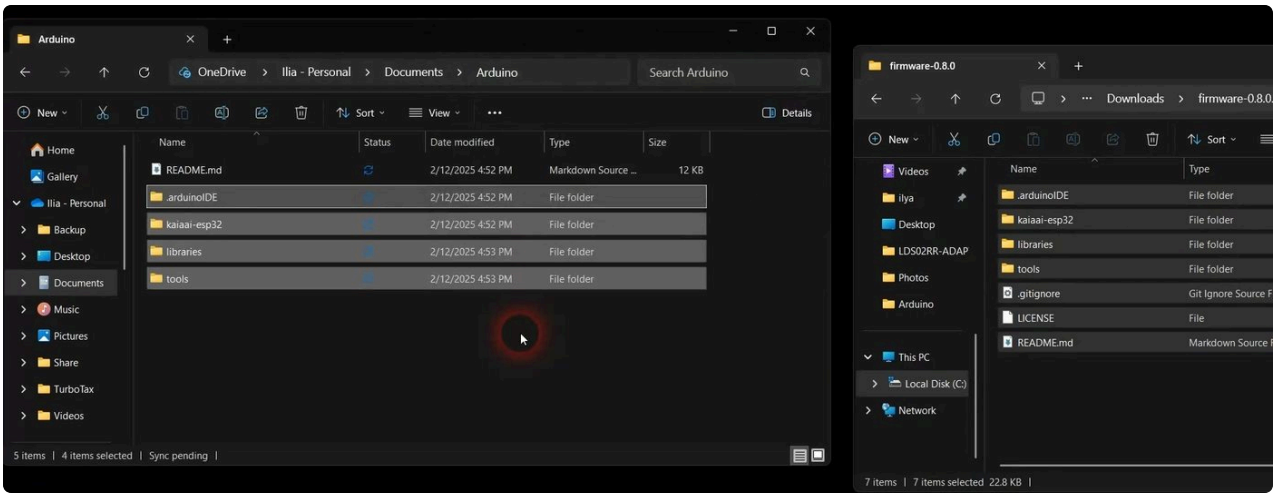
Download Source code (zip).

3. Open the ZIP (`firmware-0.8.6.zip` at the time of writing) and step into its `firmware-*` folder. It holds `.arduinoIDE` , `kaiaai-esp32` , `libraries` , `tools` and a few files.



Inside the firmware-* folder.

4. Copy **everything** in that folder into your Arduino sketch folder, usually `Documents\Arduino`. This installs the firmware project and the tested versions of the libraries it needs.



Copy everything into your sketch folder.

> **TIP**
> Your sketch folder is shown as **Sketchbook location** in **File → Preferences**. With OneDrive, it may be under OneDrive, as in the video.

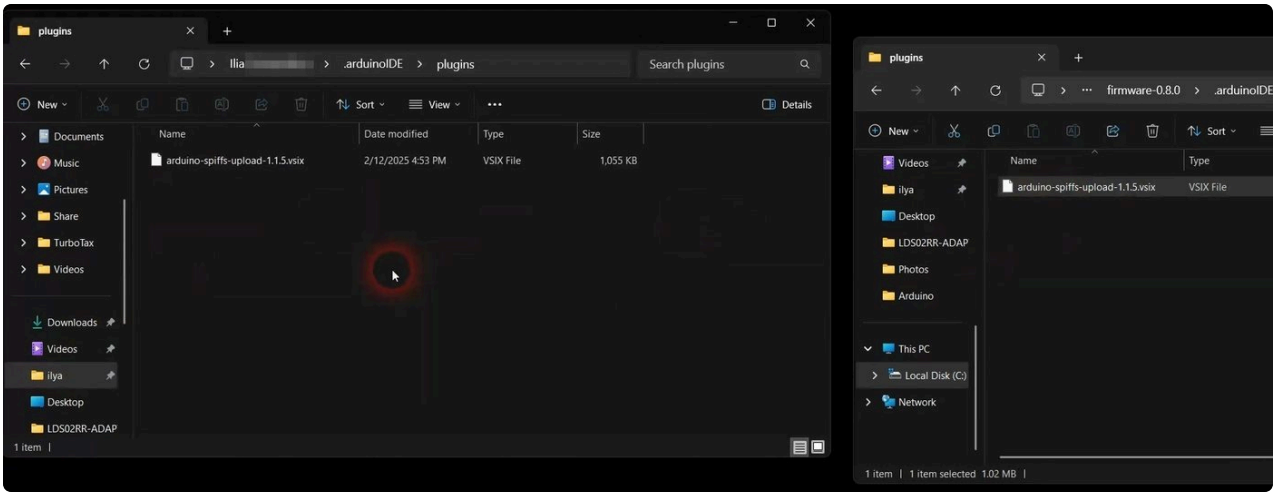
Install the Arduino IDE plugin

The `arduino-spiffs-upload` plugin uploads the robot's configuration files to the ESP32 later.

1. In the ZIP, go into `.arduinoIDE`, then `plugins`.
2. Copy everything in it into your Arduino IDE plugins folder, creating it if needed:

```
C:\Users\YourUserName\.arduinoIDE\plugins\
```

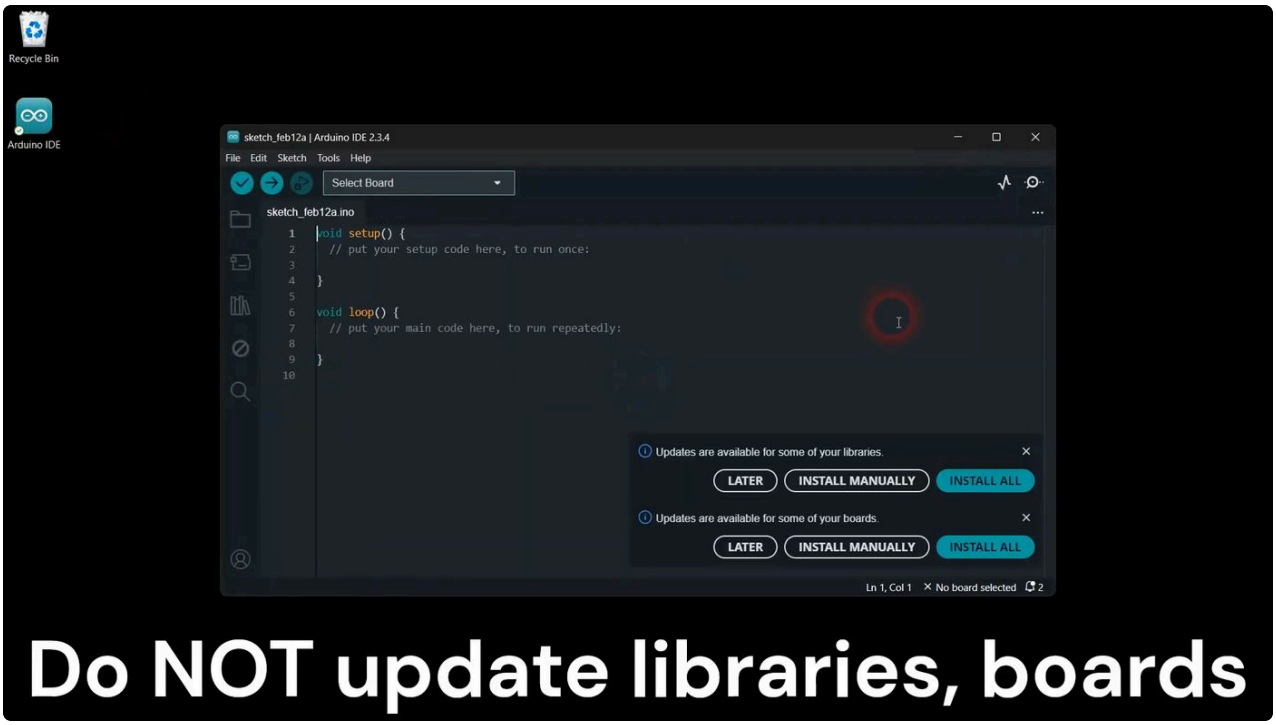
Replace `YourUserName` with your Windows user name.



Copy arduino-spiffs-upload-1.1.5.vsix into your .arduinoIDE\plugins folder.

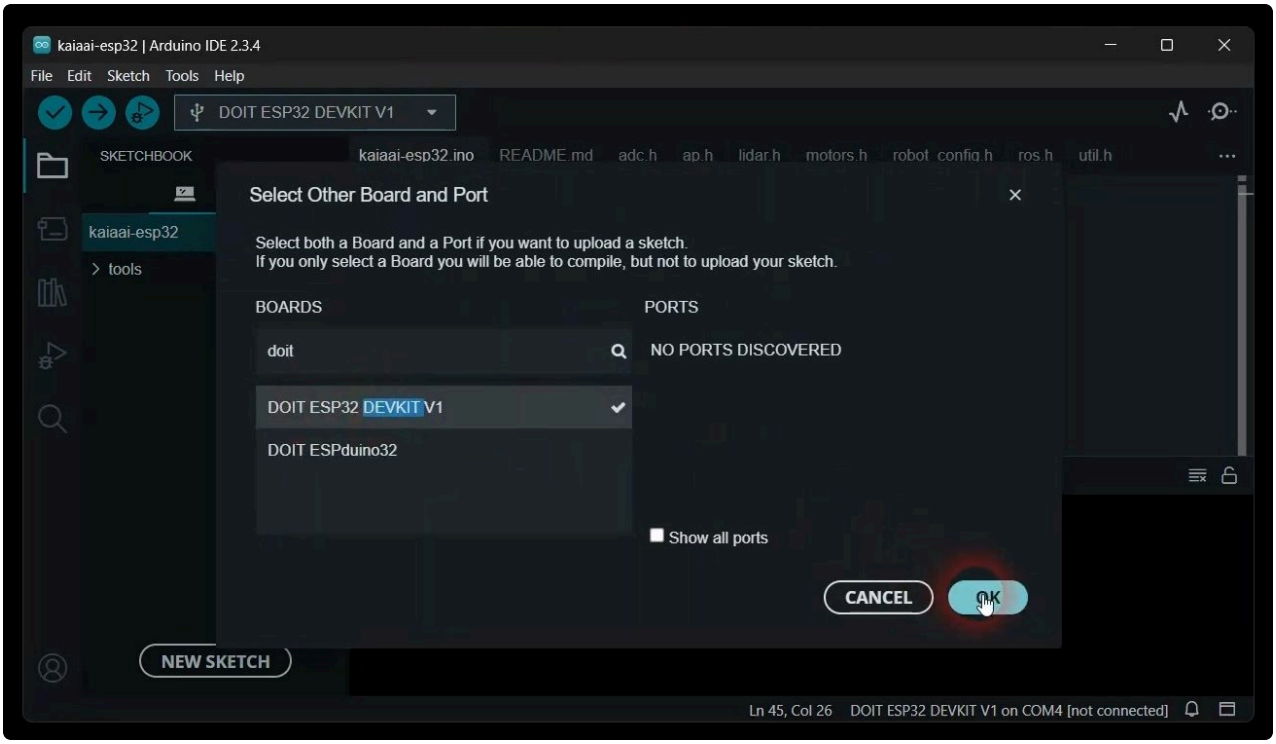
Compile the firmware

1. Launch the Arduino IDE. If it offers library or board updates, click **LATER**. Newer versions can break compilation.



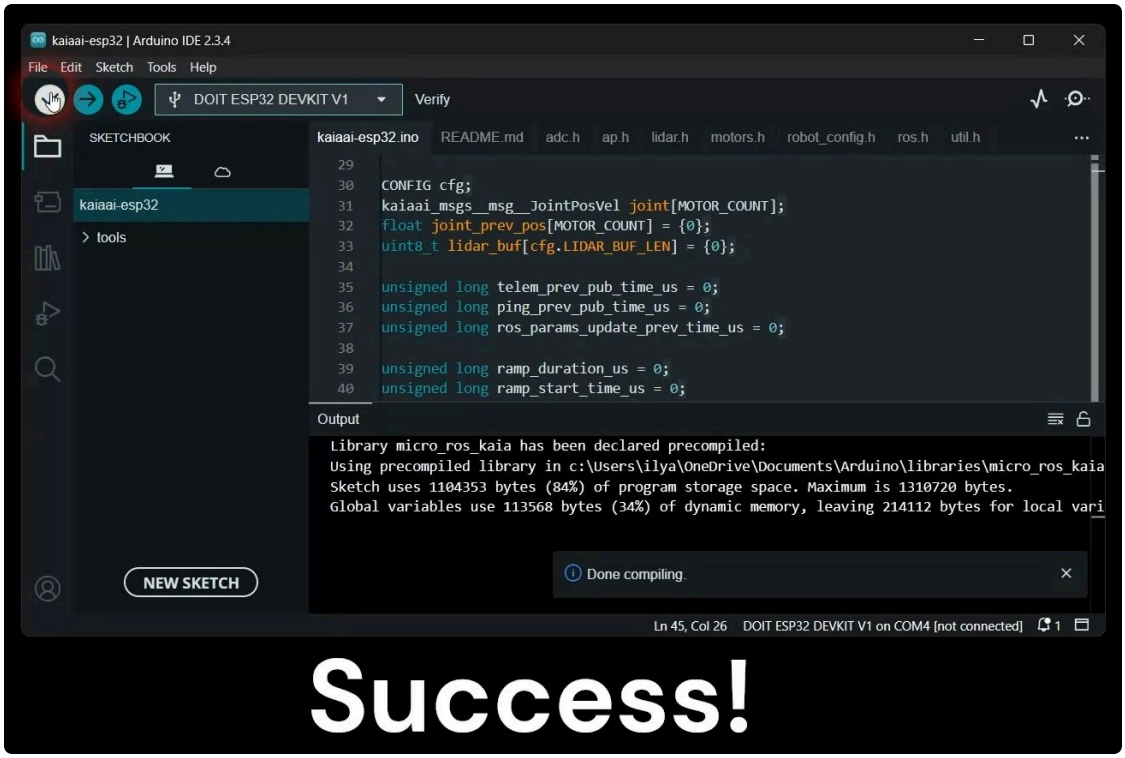
Click LATER on update offers.

2. Go to **File → Open** and open the `kaiaai-esp32` sketch from your sketch folder.
3. Click the board selector, choose **Select other board and port**, search for `doit`, select **DOIT ESP32 DEVKIT V1** and click **OK**. No port is needed to compile.



Select DOIT ESP32 DEVKIT V1.

4. Click **Verify** (check mark). Compiling can take a few minutes and should end with no errors and a memory summary.



The firmware compiled.

SINCE THE VIDEO WAS RECORDED

Arduino IDE 2.x's Library Manager can silently replace the bundled `MotorController` and `PID` libraries with same-name libraries by other authors, causing errors like `use of deleted function MotorController()` or `ledcAttachPin was not declared`. Prevent this by renaming the kaiaai libraries in `Documents\Arduino\libraries` (for example to `MotorController_kaia`).

COMPILE ERROR: `#ERROR THIS CODE RUNS ON ESP32`

No ESP32 board is selected. Select it as in step 3. If no ESP32 boards are listed, add the Espressif toolchain first.

If you prefer Arduino IDE 1.8.19

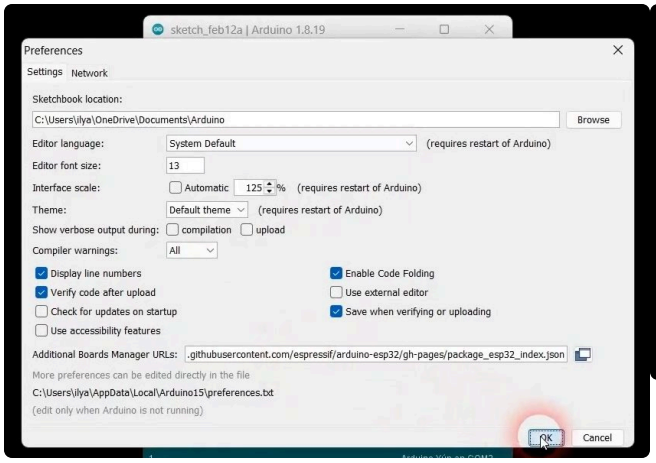
Use this instead of IDE 2; otherwise skip to Install Windows WSL2.

1. On arduino.cc **Software**, under **Legacy IDE (1.8.X)**, download **Arduino IDE 1.8.19** for Windows (Win 7 and newer). Install and launch it.

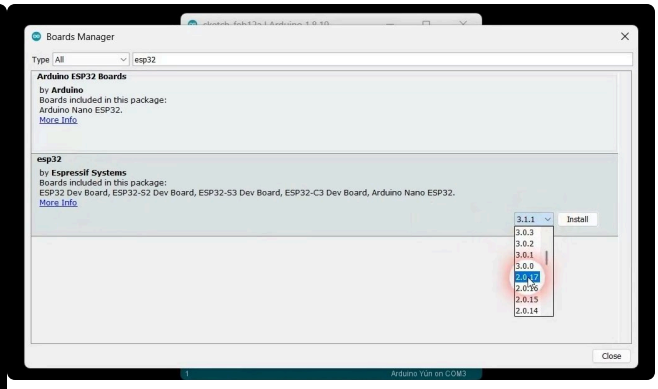


Arduino IDE 1.8.19 in the Legacy IDE section.

2. In **File → Preferences**, paste the same Espressif URL into **Additional Boards Manager URLs**. In **Tools → Board → Boards Manager**, search for `esp32`, select version **2.0.17** of **esp32 by Espressif Systems** and click **Install**.

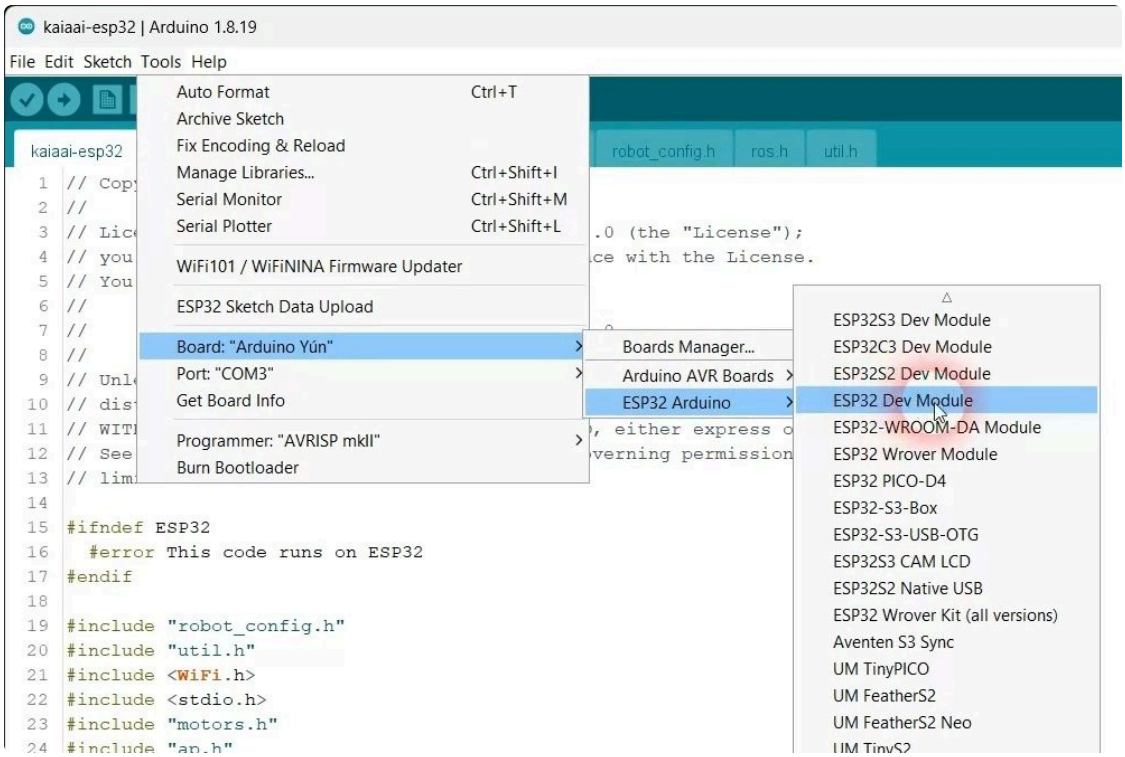


File → Preferences in IDE 1.8.19.



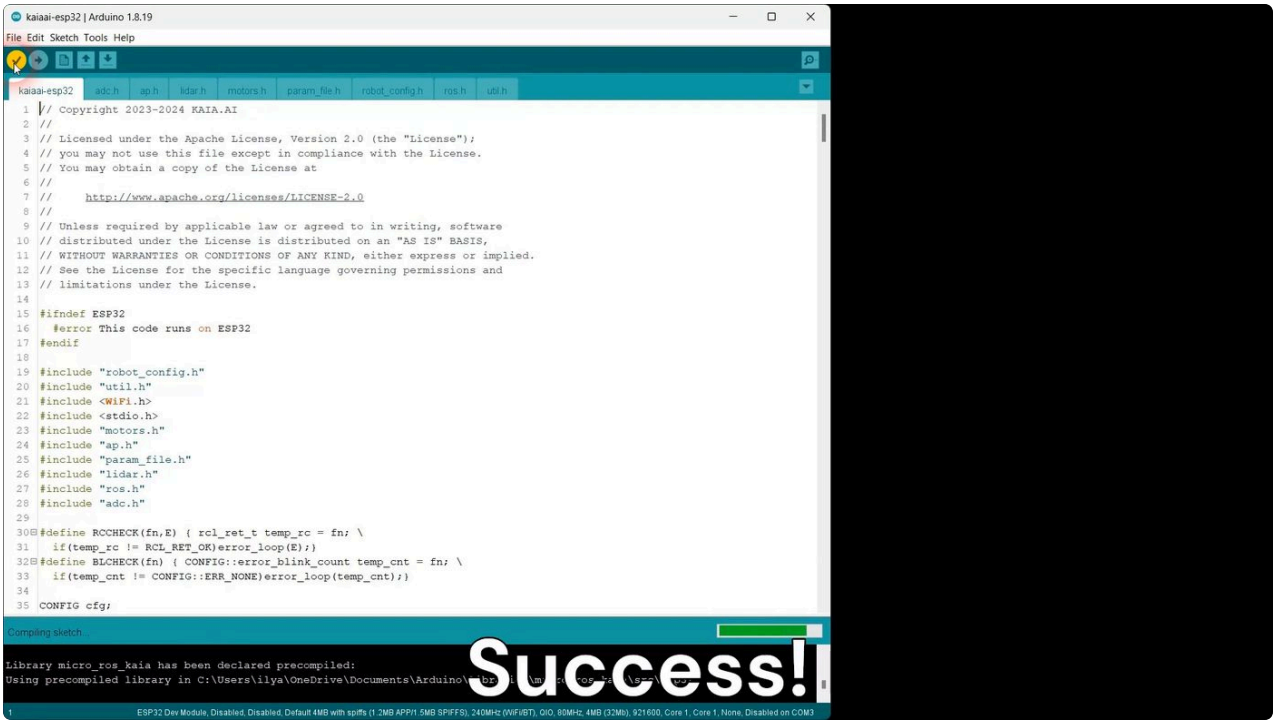
Boards Manager: esp32 2.0.17.

3. Copy the firmware project into `Documents\Arduino` as in Download the robot firmware project. With IDE 1.8.19, the copied `tools` folder adds **Tools → ESP32 Sketch Data Upload**; skip the plugin step.
4. Go to **File → Open** and open `Documents\Arduino\kaiaai-esp32`.
5. Go to **Tools → Board → ESP32 Arduino** and select **ESP32 Dev Module**.



Select ESP32 Dev Module.

6. Click **Verify**. It can take a few minutes and should finish with no errors.

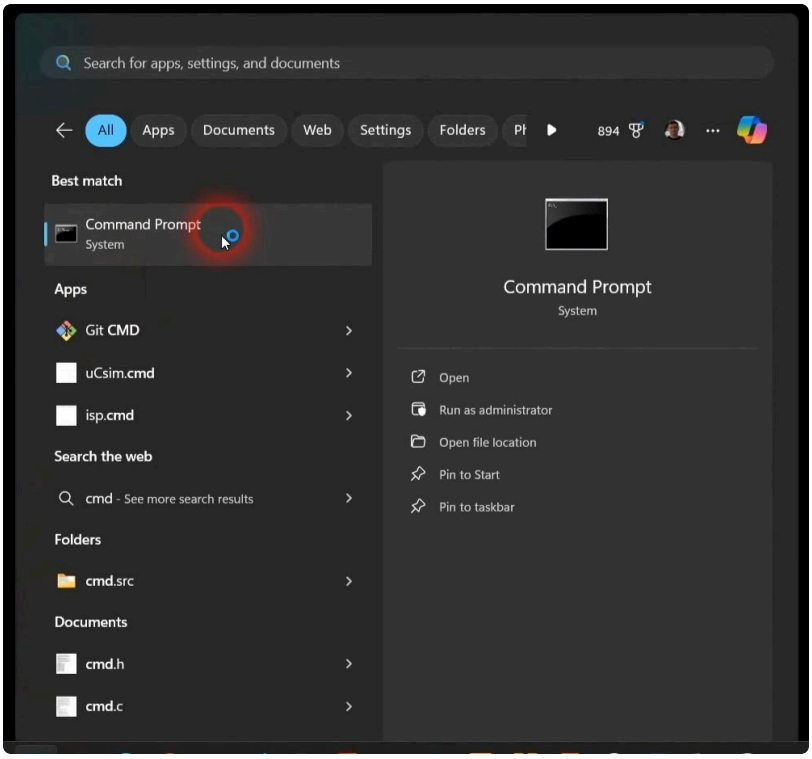


Compiling in Arduino IDE 1.8.19.

Install Windows WSL2

ROS2, the software that controls the robot, runs on Linux. WSL2 (Windows Subsystem for Linux) runs Linux on your Windows PC.

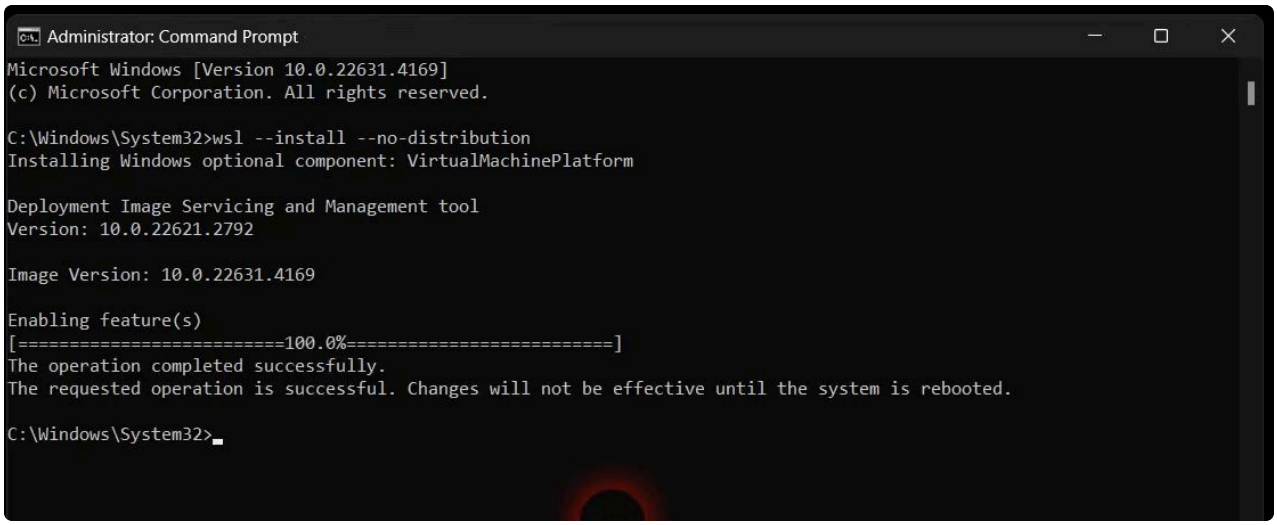
1. Click **Start**, type `cmd`, and on **Command Prompt** choose **Run as administrator**.



Run Command Prompt as administrator.

2. Run this command and wait until it reports success:

```
wsl --install --no-distribution
```



WSL2 installed; reboot to finish.

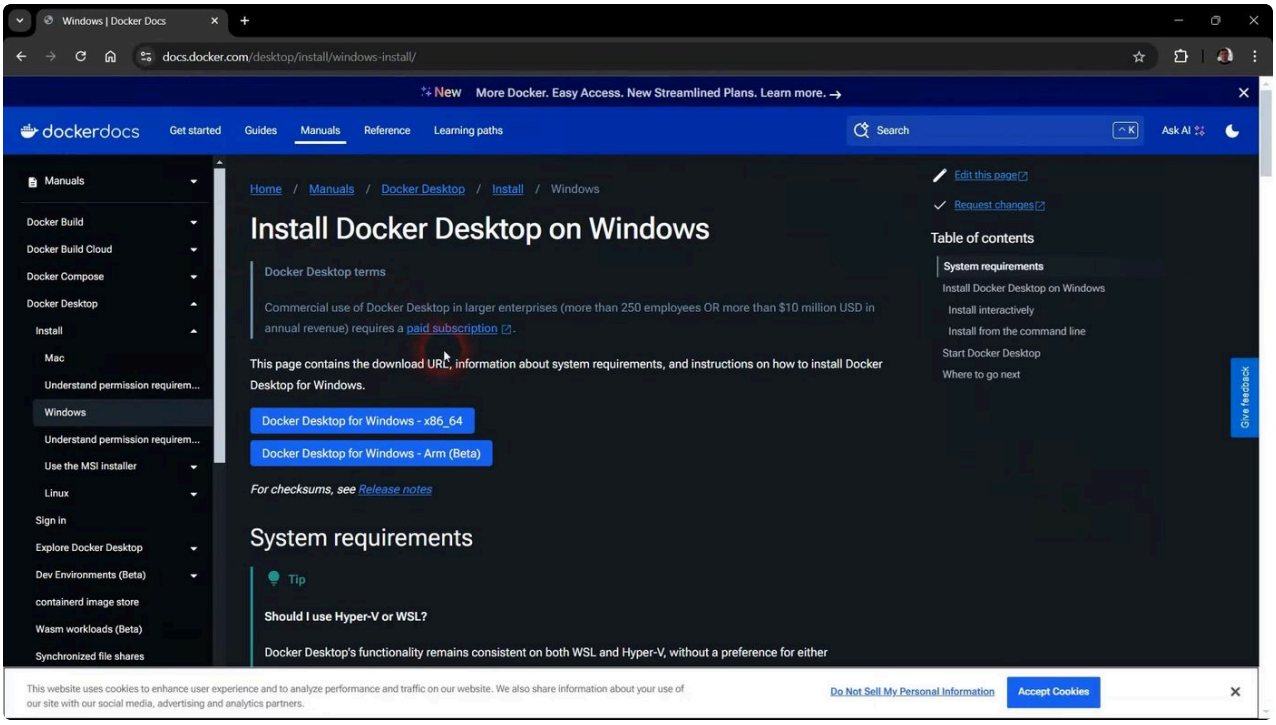
3. Reboot your PC.

> **SINCE THE VIDEO WAS RECORDED**
> If WSL2 fails to install, follow Microsoft's WSL installation and troubleshooting instructions on learn.microsoft.com. As a last resort, run Ubuntu in a virtual machine (Appendix A), though in my experience it makes ROS2 laggy.

Install Docker Desktop

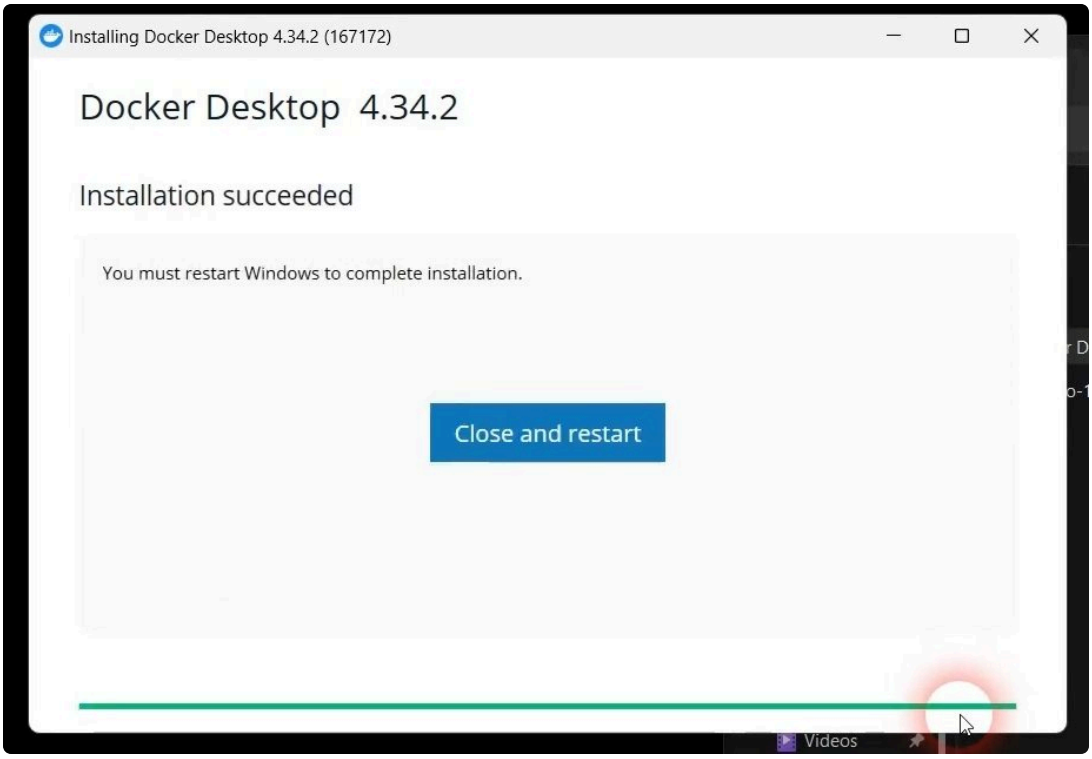
Docker installs and runs a large collection of robotics software in one go.

1. Go to docs.docker.com/desktop/install/windows-install/ and download **Docker Desktop for Windows - x86_64** (for a typical Intel or AMD PC).



Download the x86_64 installer.

2. Run `Docker Desktop Installer.exe`, keep **Add shortcut to desktop** checked and click **OK**.
3. At **Installation succeeded**, click **Close and restart**.



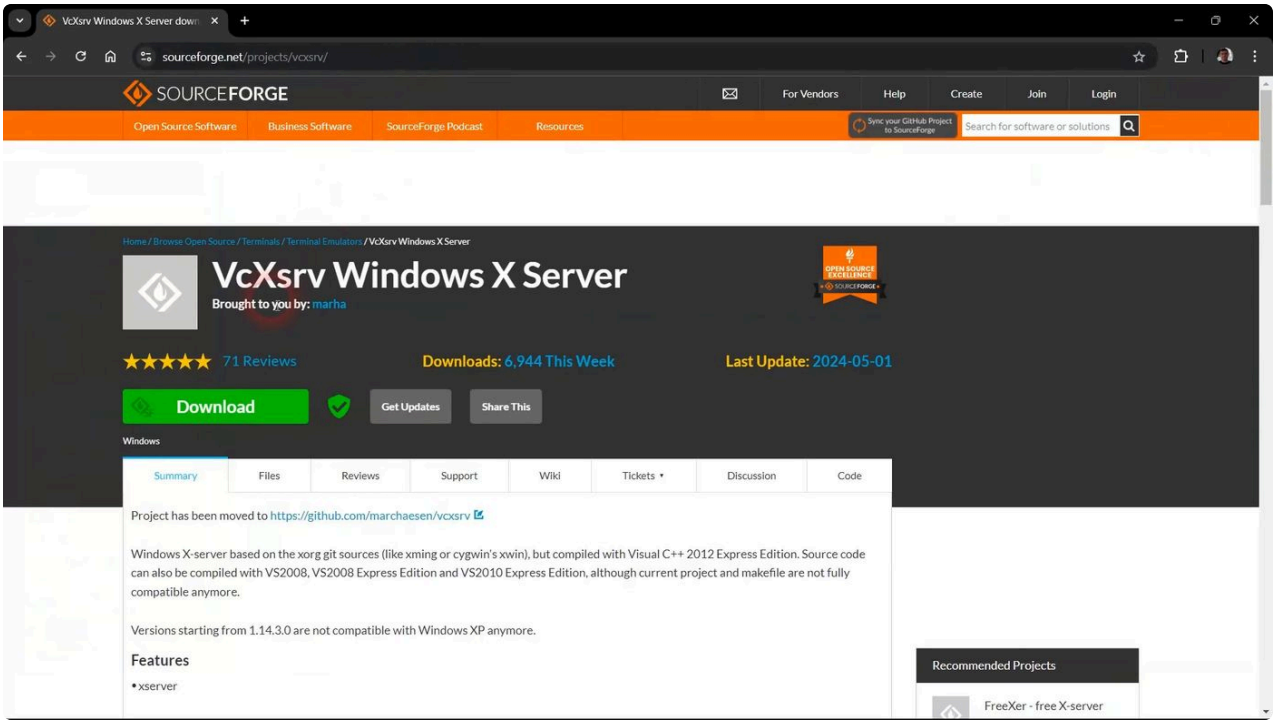
Click Close and restart.

> **NOTE**
> Docker Desktop is free for personal use; commercial use in larger enterprises requires a paid subscription.

Install the X server

The X server shows the Linux robot software's GUI, such as the map view, on your Windows desktop. The video uses VcXsrv.

1. Browse to sourceforge.net/projects/vcxsrv/ and click **Download**.

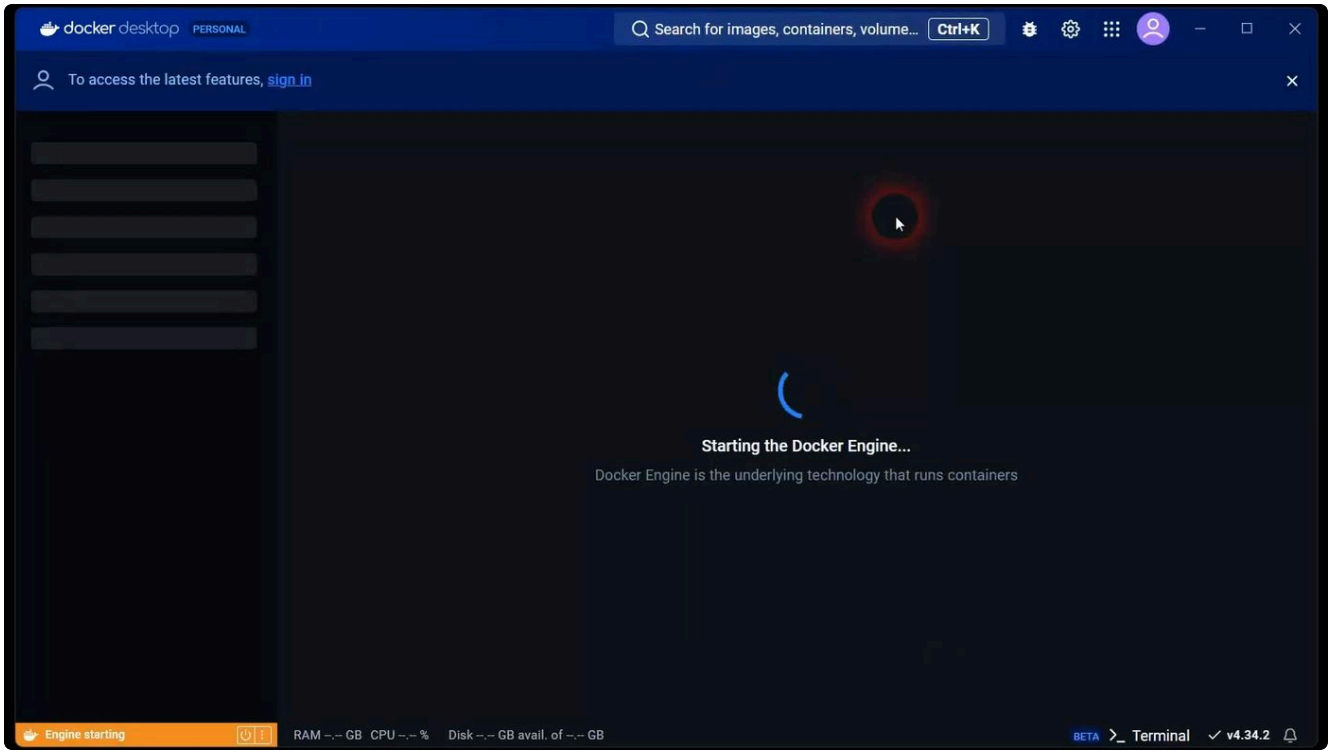


Download VcXsrv from SourceForge.

2. Run the `vcxsrv` installer with the default components and folder.

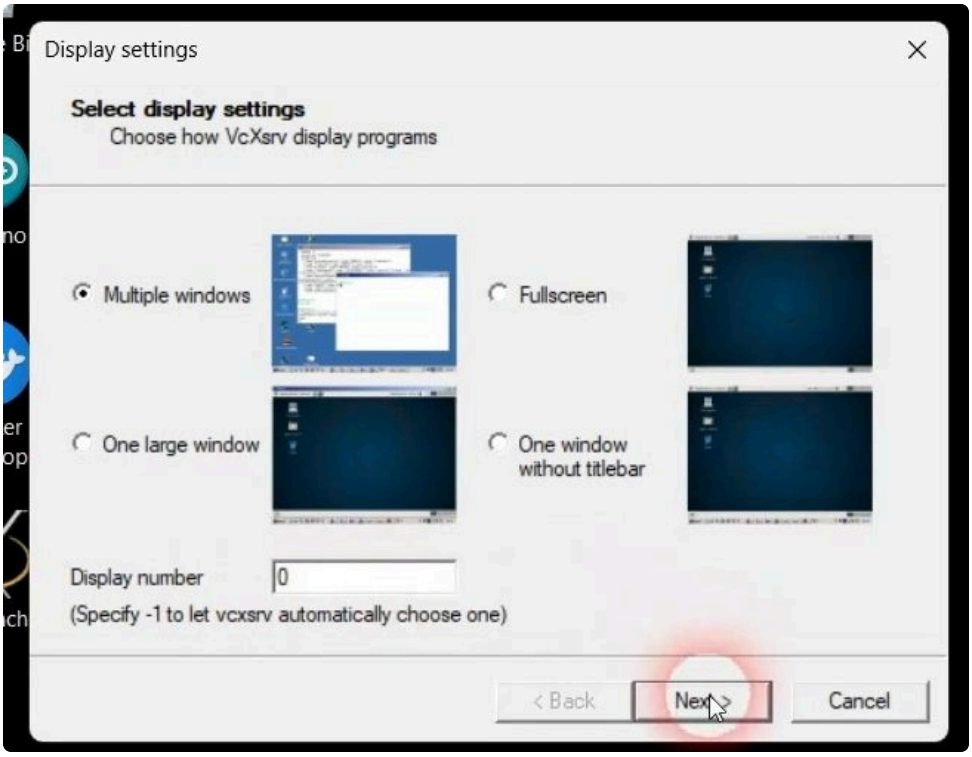
Launch Docker Desktop and the X server

1. Launch **Docker Desktop** from its desktop icon. No account is needed: click **Skip** on the sign-in screen and the survey. Once Docker Engine has started, close the window; the engine keeps running.



Starting the Docker Engine.

2. Launch **XLaunch** from your desktop. On **Display settings**, set **Display number** to `0` (zero), keep **Multiple windows** and click **Next**. Accept the remaining defaults and click **Finish**.



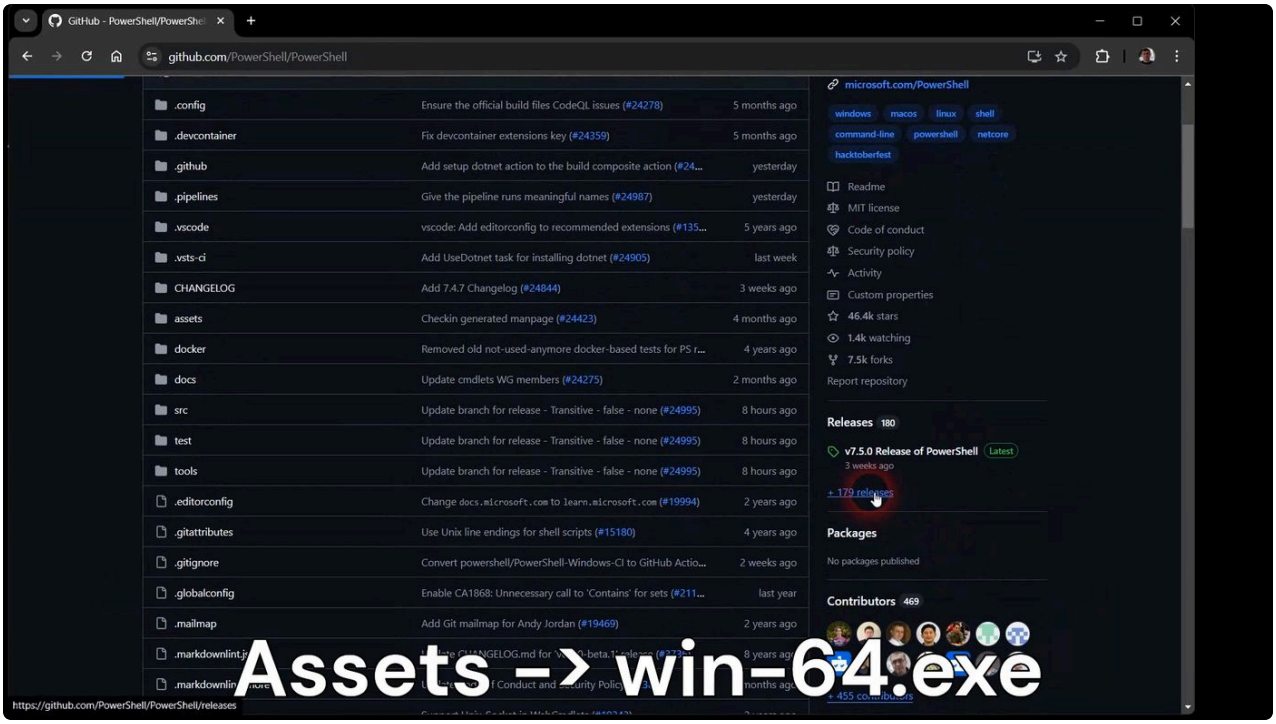
Set Display number to 0.

> **TIP**
> Launch the X server after every PC restart, before starting any robot GUI. If RViz or another ROS2 GUI reports `could not connect to display`, the X server isn't running.

Install PowerShell

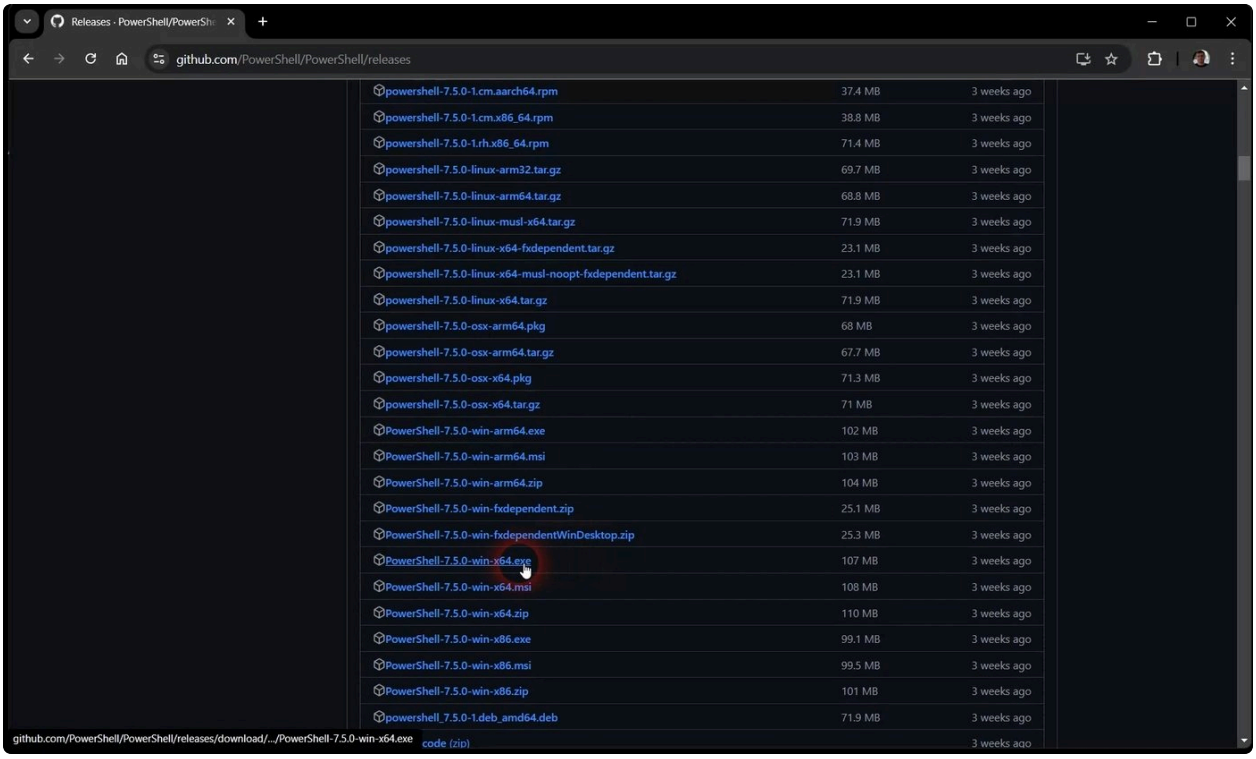
You will type the robot's Docker commands in PowerShell.

1. Browse to github.com/PowerShell/PowerShell and click **Releases**.



Open the PowerShell releases.

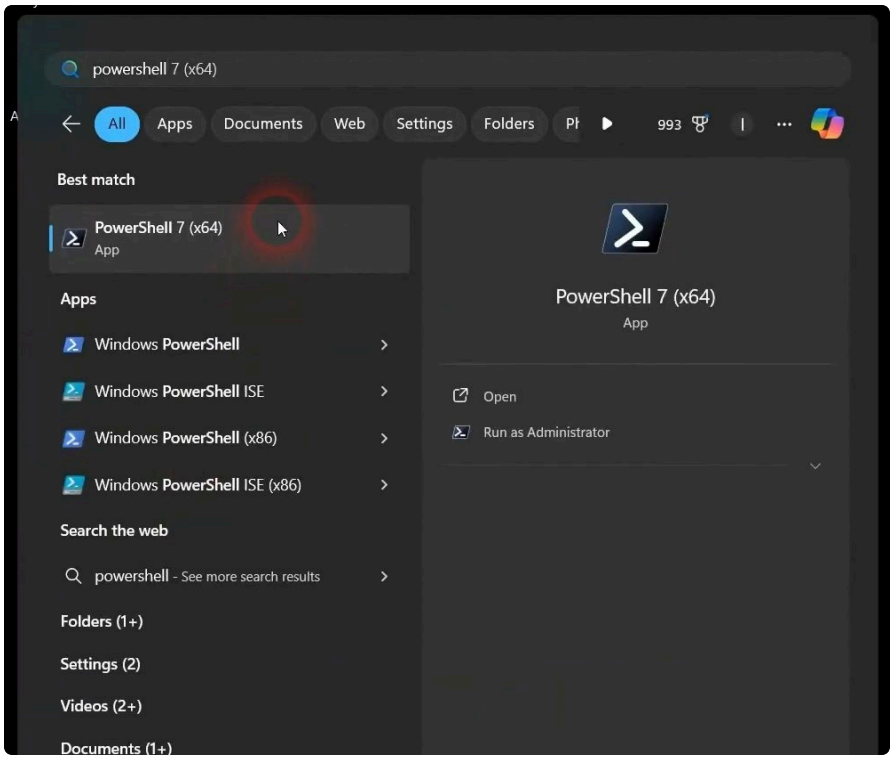
2. In the latest release's **Assets**, download `PowerShell-<version>-win-x64.exe` (in the video, `PowerShell-7.5.0-win-x64.exe`).



Download the win-x64.exe asset.

3. Run the installer.

4. Click **Start**, type `powershell` and launch **PowerShell 7 (x64)**. Pin it to the taskbar.

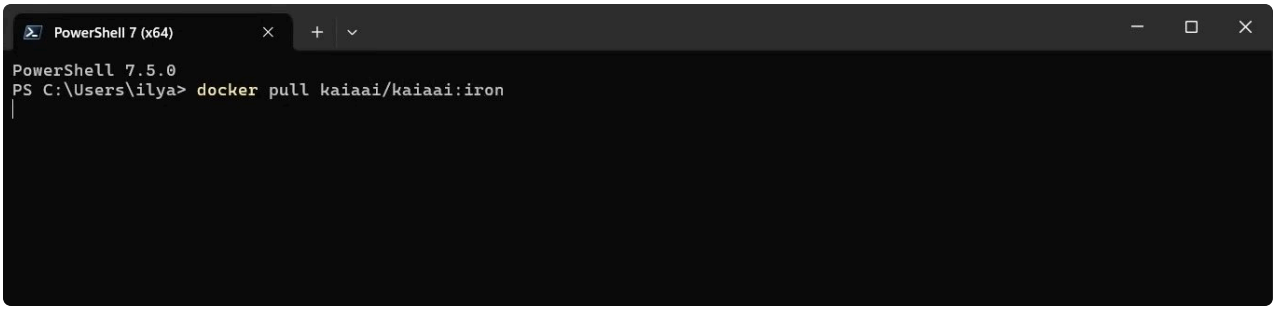


PowerShell 7 (x64), not the older Windows PowerShell.

Download the robot software image

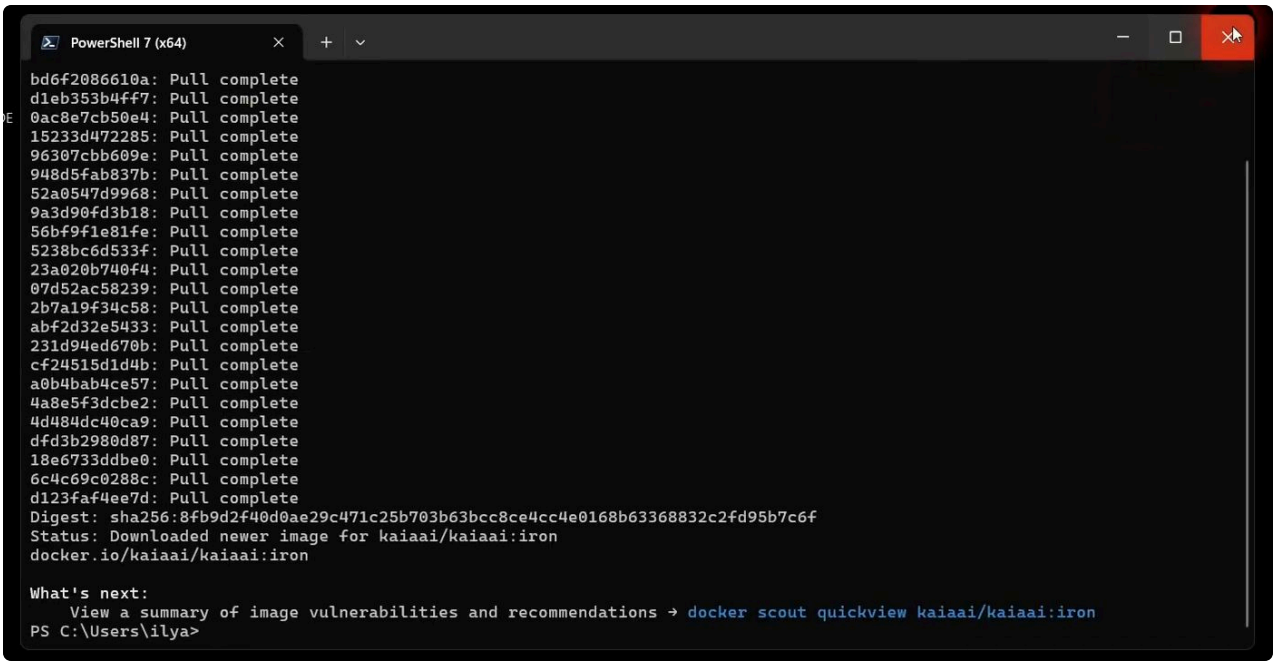
1. Make sure Docker Desktop is running.
2. In PowerShell, type:

```
docker pull kaiaai/kaiaai:iron
```



docker pull in PowerShell.

3. The image is large. When done, PowerShell shows `Status: Downloaded newer image for kaiaai/kaiaai:iron`.



The image has downloaded.

`DOCKER PULL` **FAILS WITH** `ERROR DURING CONNECT`

An error like this:

```
error during connect: Post "http://%2F%2F.%2Fpipe%2FdockerDesktopLinuxEngine/...": open
//./pipe/dockerDesktopLinuxEngine: The system cannot find the file specified.
```

means the Docker engine isn't running. Launch Docker Desktop, wait for the engine, and retry.

SINCE THE VIDEO WAS RECORDED

A ROS2 Jazzy image, `kaiaai/kaiaai:jazzy`, is also available. This book uses the Iron image, `kaiaai/kaiaai:iron`.

The software is installed. Next, you will upload the firmware and bring up the robot.

3. Upload the Firmware and Bring Up the Robot

Video: Step by Step Arduino LiDAR Robot Bringup

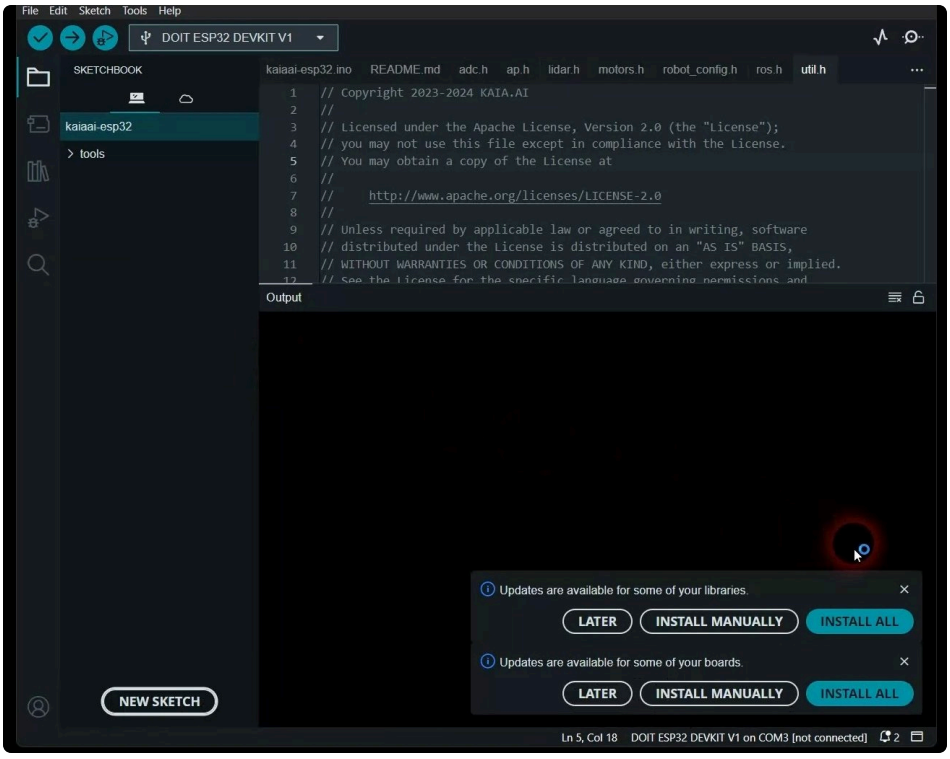
You upload the firmware and its configuration ("sketch data") to the ESP32, connect the robot to your WiFi and start the ROS2 software on your PC. At the end you drive the wheels from your keyboard and see live LiDAR data.

You need the assembled robot (Chapter 1) and the PC software from Chapter 2. Plan for about 30 minutes.

Upload the firmware

These steps use Arduino IDE 2.x. With IDE 1.8.19, see If you use Arduino IDE 1.8.19 below; only the menus differ.

1. Open the `kaiaai-esp32` sketch. If the IDE offers library or board updates, click **LATER**.



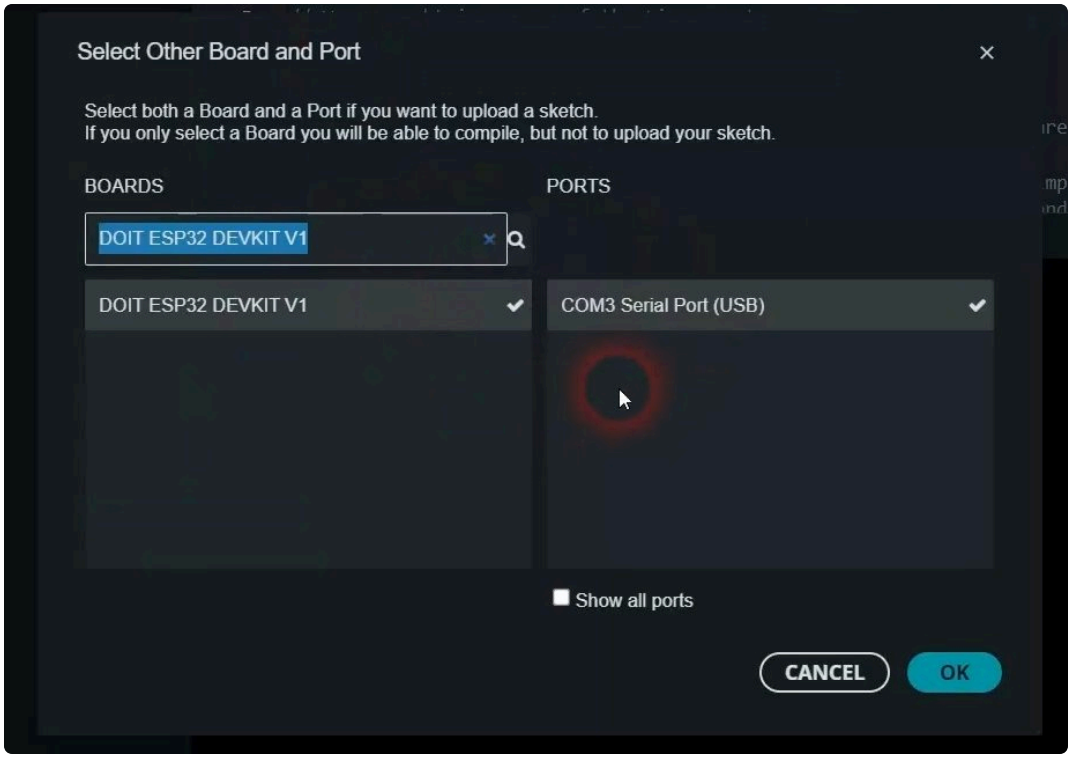
Close the update pop-ups. Don't update.

2. Connect the ESP32 to your PC over USB, with the board's power switch **off**.



Plug the USB cable into the ESP32.

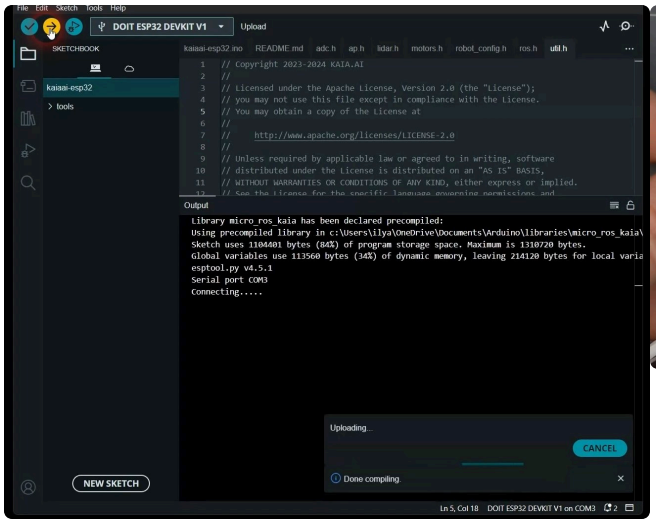
3. In the board selector, choose **DOIT ESP32 DEVKIT V1** and your COM port (`COM3` in the video).



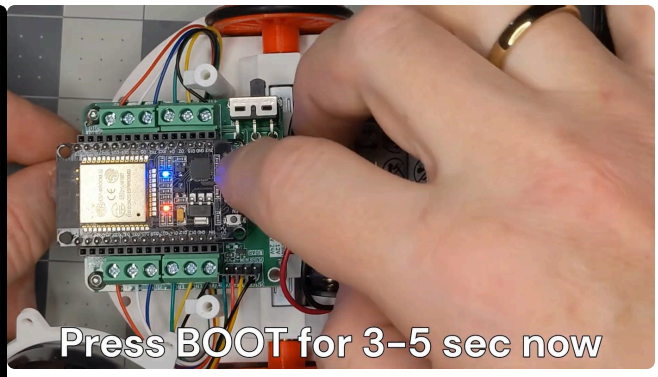
Select the board and COM port.

4. Click **Upload** (the right arrow). The first compile takes a while.

5. When the output shows `Connecting....`, press and hold **BOOT** for 3 to 5 seconds to put the ESP32 into download mode.



Upload starts connecting...

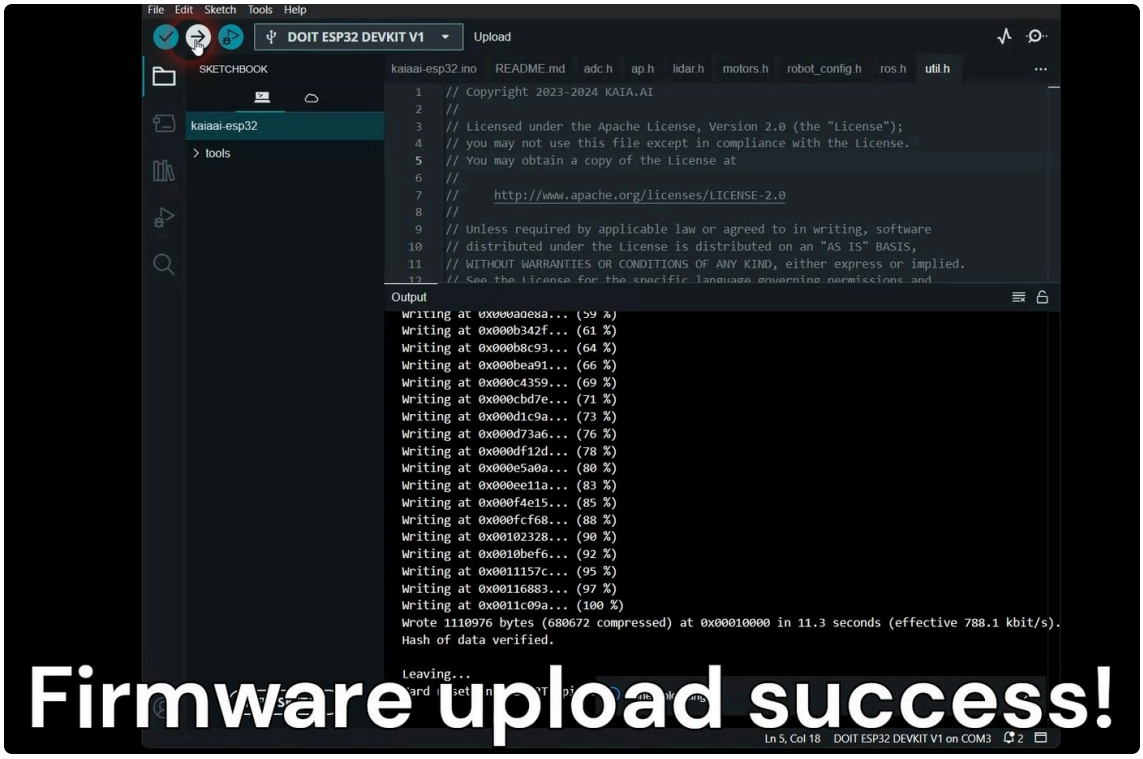


...press BOOT for 3–5 seconds now.

SINCE THE VIDEO WAS RECORDED

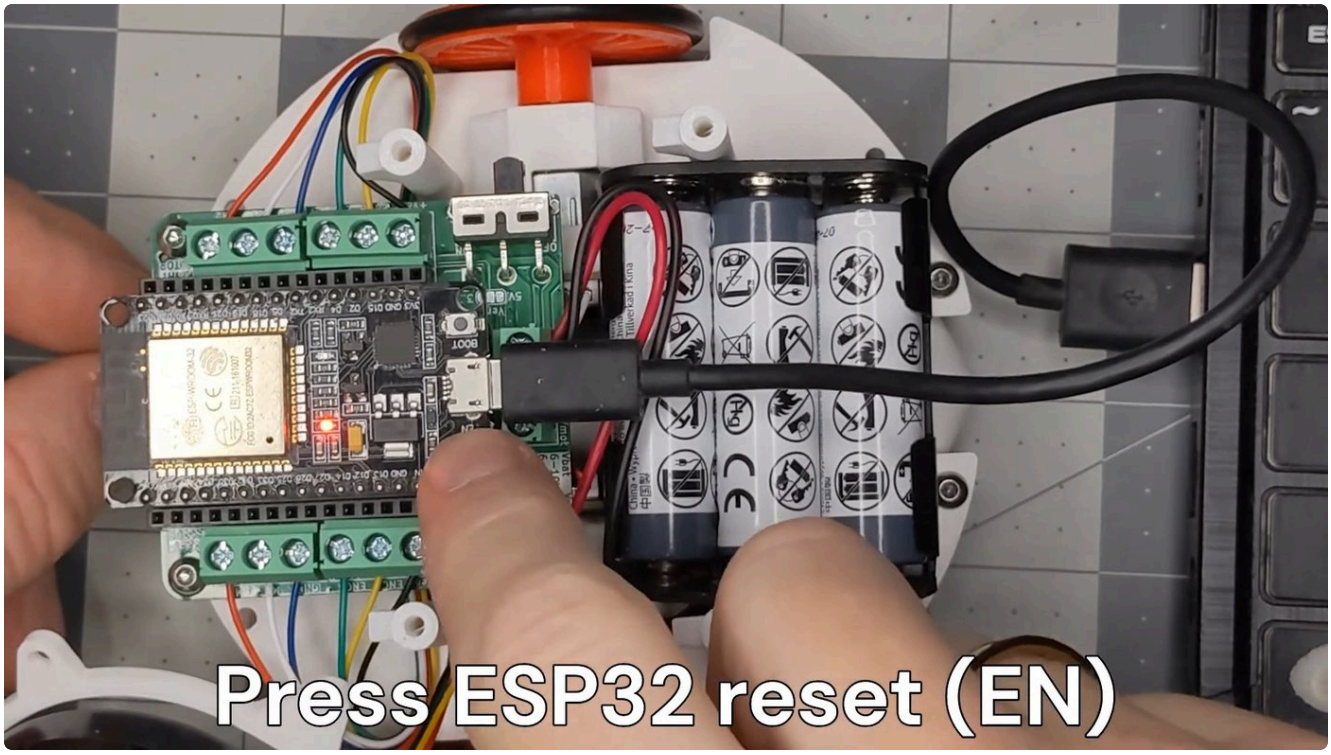
Some ESP32 boards don't need BOOT pressed; if the upload succeeds without it, that's fine. If it fails with `Failed to connect to ESP32: No serial data received`, retry and press BOOT as soon as `Connecting...` appears.

6. Confirm the upload succeeded: the output ends with `Hash of data verified.` and `Hard resetting via RTS pin...`.



Firmware upload success.

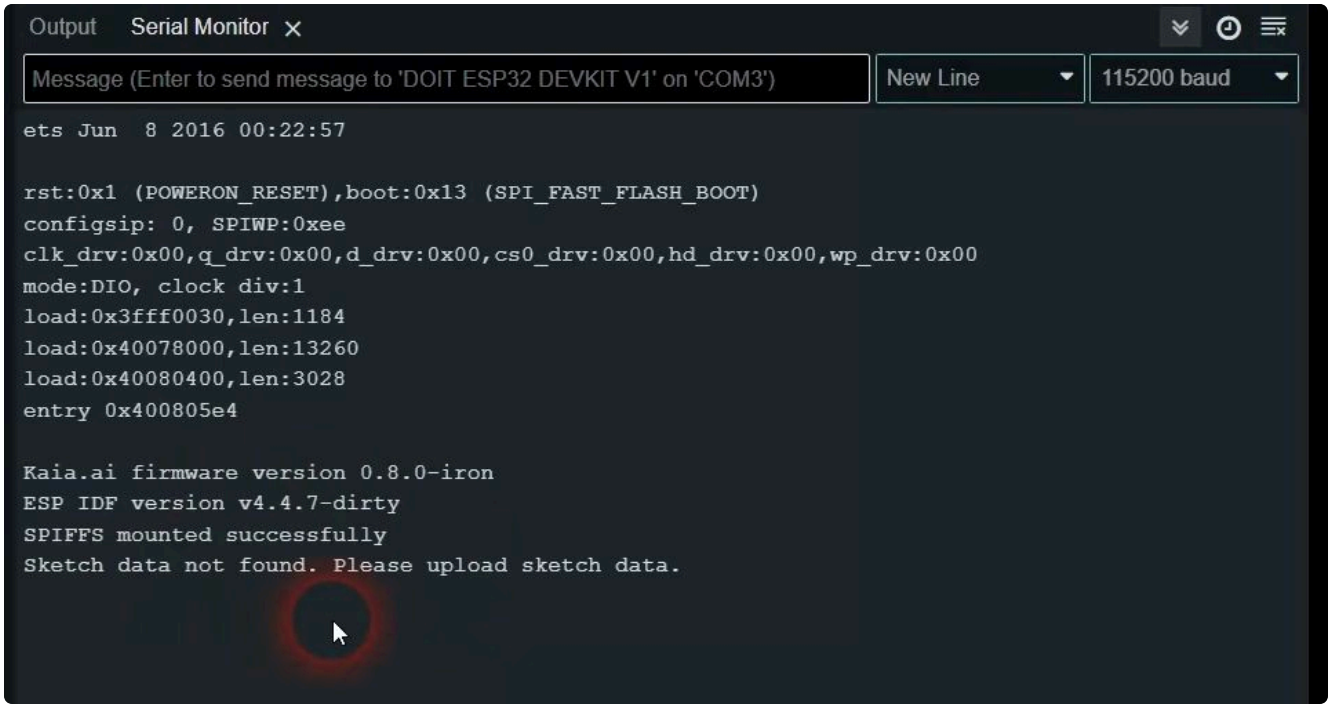
7. Open **Tools → Serial Monitor** at **115200 baud** and press the ESP32's **EN** (reset) button.



Press the ESP32 reset (EN) button.

8. The ESP32 boots and asks for sketch data, as expected:

```
Kaia.ai firmware version 0.8.0-iron
ESP IDF version v4.4.7-dirty
SPIFFS mounted successfully
Sketch data not found. Please upload sketch data.
```

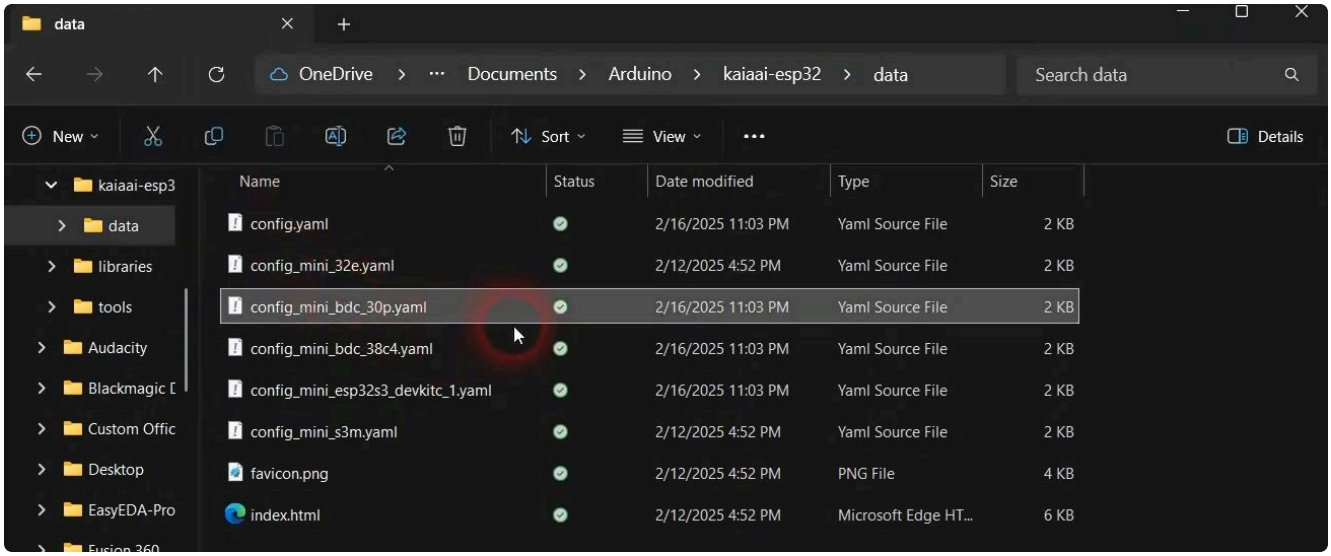


The ESP32 asks for sketch data.

Upload the sketch data

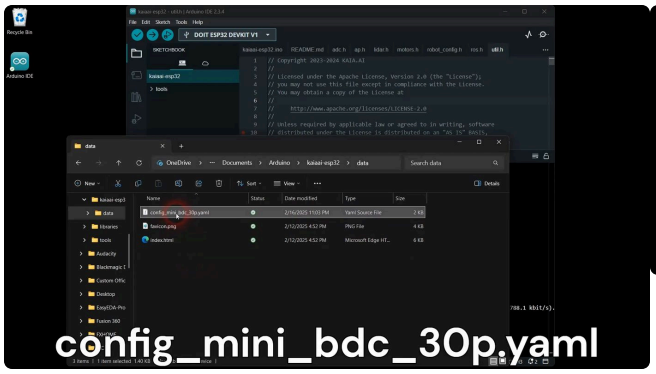
The sketch data is the robot's configuration file, `config.yaml`, plus the WiFi setup web page. It lives in a separate flash area (SPIFFS), so it is uploaded separately.

1. **Close the Serial Monitor**; it holds the COM port and the upload will fail.
2. Open **Sketch → Show Sketch Folder** and go into the `data` folder. It holds `config.yaml`, which the firmware loads at power-up, config files for other hardware, `favicon.png` and `index.html`.

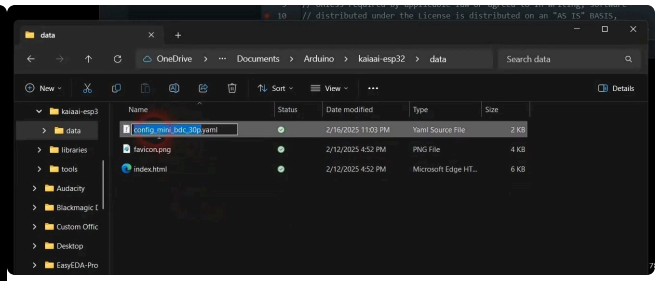


The data folder.

3. Delete every `.yaml` file **except** `config_mini_bdc_30p.yaml`, the config for the BLD-120MM-PACK with the BDC-30P board. Keep `favicon.png` and `index.html`.
4. Rename `config_mini_bdc_30p.yaml` to `config.yaml`. It sets the ESP32 pins, motors and LiDAR model.

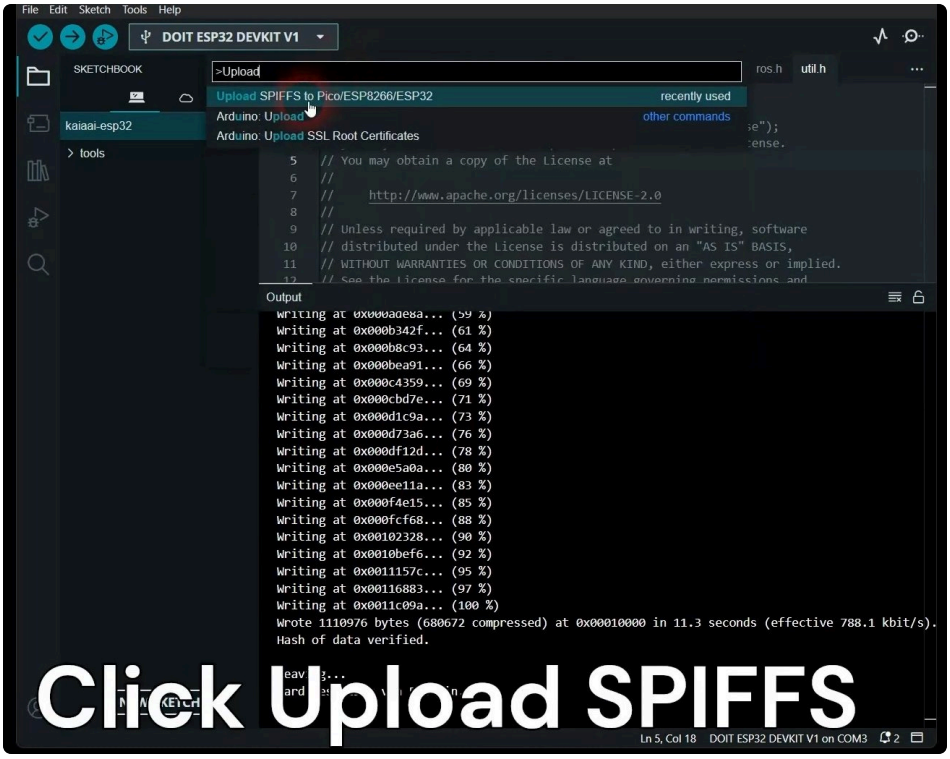


Keep only the BDC-30P config...



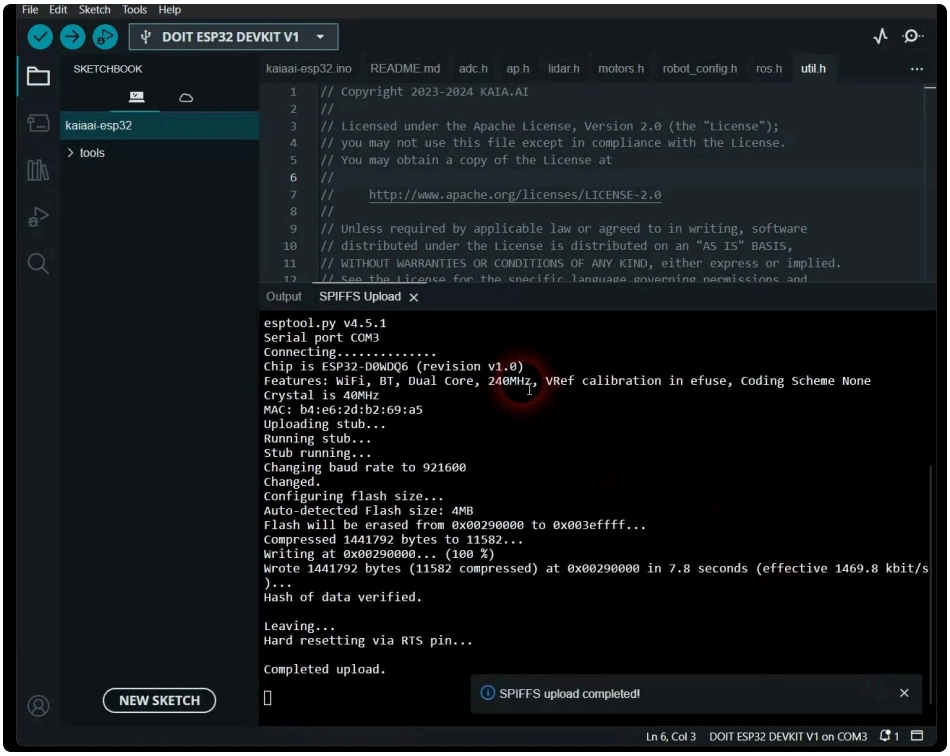
...and rename it to config.yaml.

5. Click inside the code window, press **Ctrl+Shift+P**, type `Upload` and click **Upload SPIFFS to Pico/ESP8266/ESP32**. At `Connecting....`, hold **BOOT** for 3 to 5 seconds.



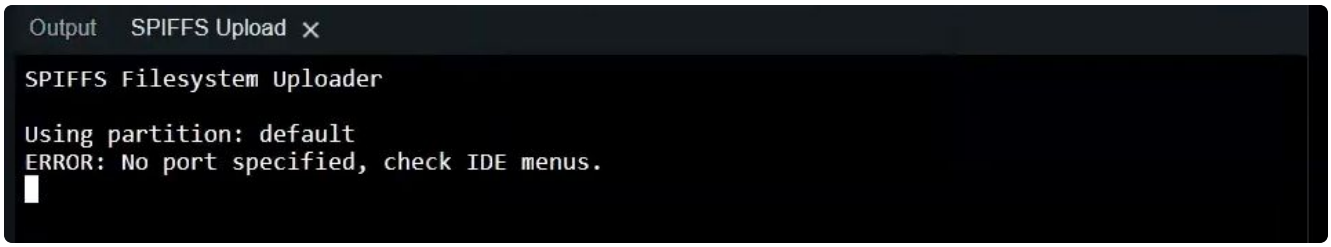
Pick Upload SPIFFS to Pico/ESP8266/ESP32.

Confirm the upload succeeded: the IDE shows **SPIFFS upload completed!**



Sketch data upload completed.

6. If it fails right away with `ERROR: No port specified, check IDE menus.`, restart the IDE and retry. This happens often.



A common SPIFFS upload error.

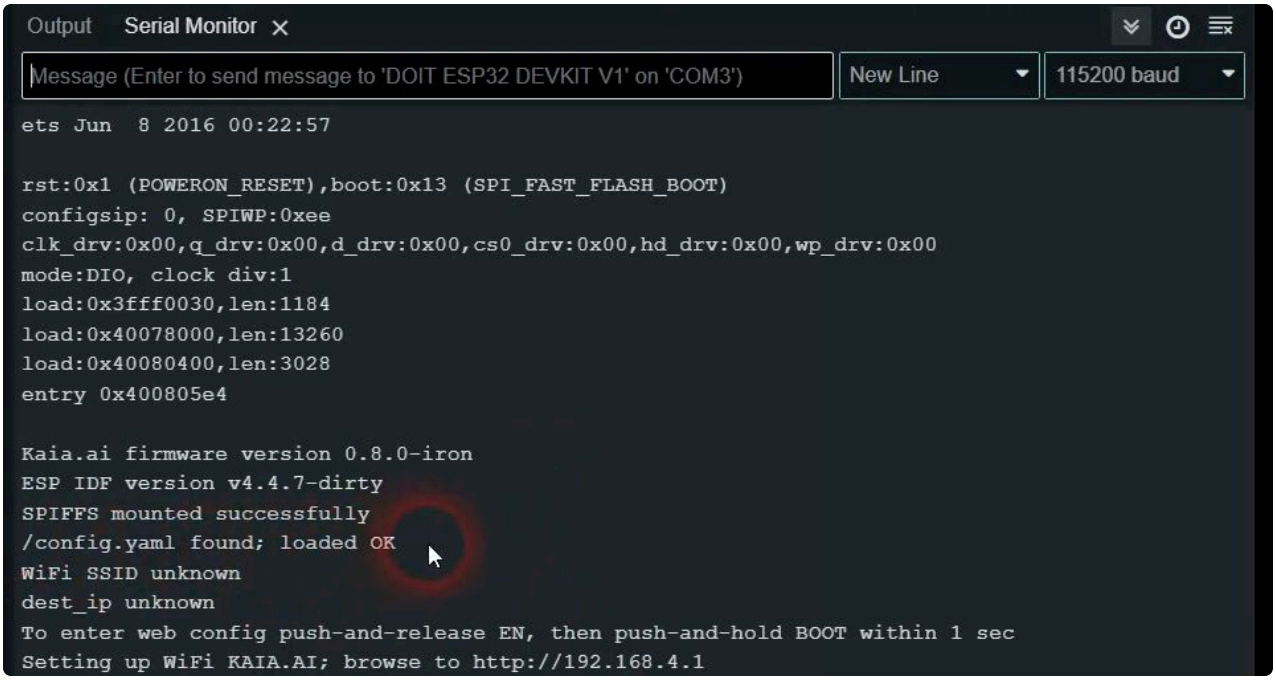
IF THE SKETCH DATA UPLOAD STILL FAILS

- `Could not open COM3, the port doesn't exist` : close the Serial Monitor, check the USB cable, retry.
- `SPIFFS_write error(-10010): unknown / error adding file!` : a file name in `data` is over 30 characters.
- `This chip is ESP32-S3 not ESP32. Wrong --chip argument?` : wrong board selected in the IDE.

See Appendix B for more.

7. Close the SPIFFS Upload output tab, reopen the Serial Monitor and press **EN**. The ESP32 loads `config.yaml` and enters WiFi configuration mode:

```
SPIFFS mounted successfully
/config.yaml found; loaded OK
WiFi SSID unknown
dest_ip unknown
To enter web config push-and-release EN, then push-and-hold BOOT within 1 sec
Setting up WiFi KAIA.AI; browse to http://192.168.4.1
```



Config loaded; ready for WiFi configuration.

If you use Arduino IDE 1.8.19

Skip this section if you use IDE 2.x. With IDE 1.8.19, follow these steps instead of the two sections above.

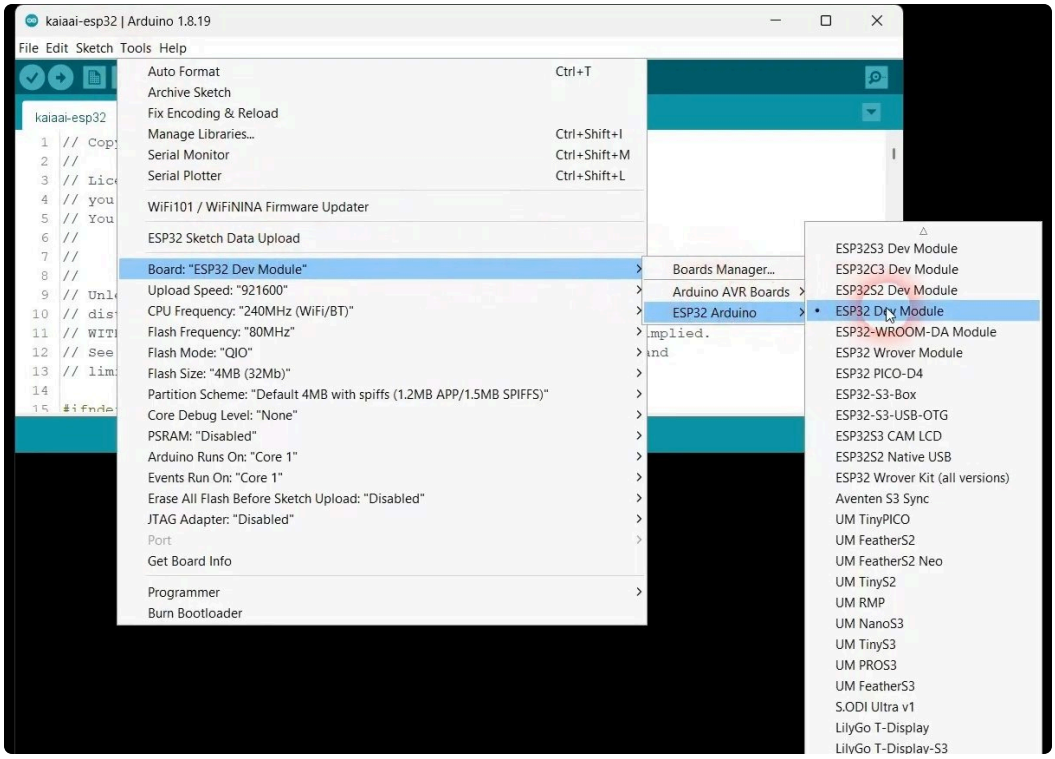
Upload the firmware

1. Open the `kaiaai-esp32` sketch. If the IDE offers library or board updates, decline them.
2. Connect the ESP32 to your PC over USB, with the board's power switch **off**.



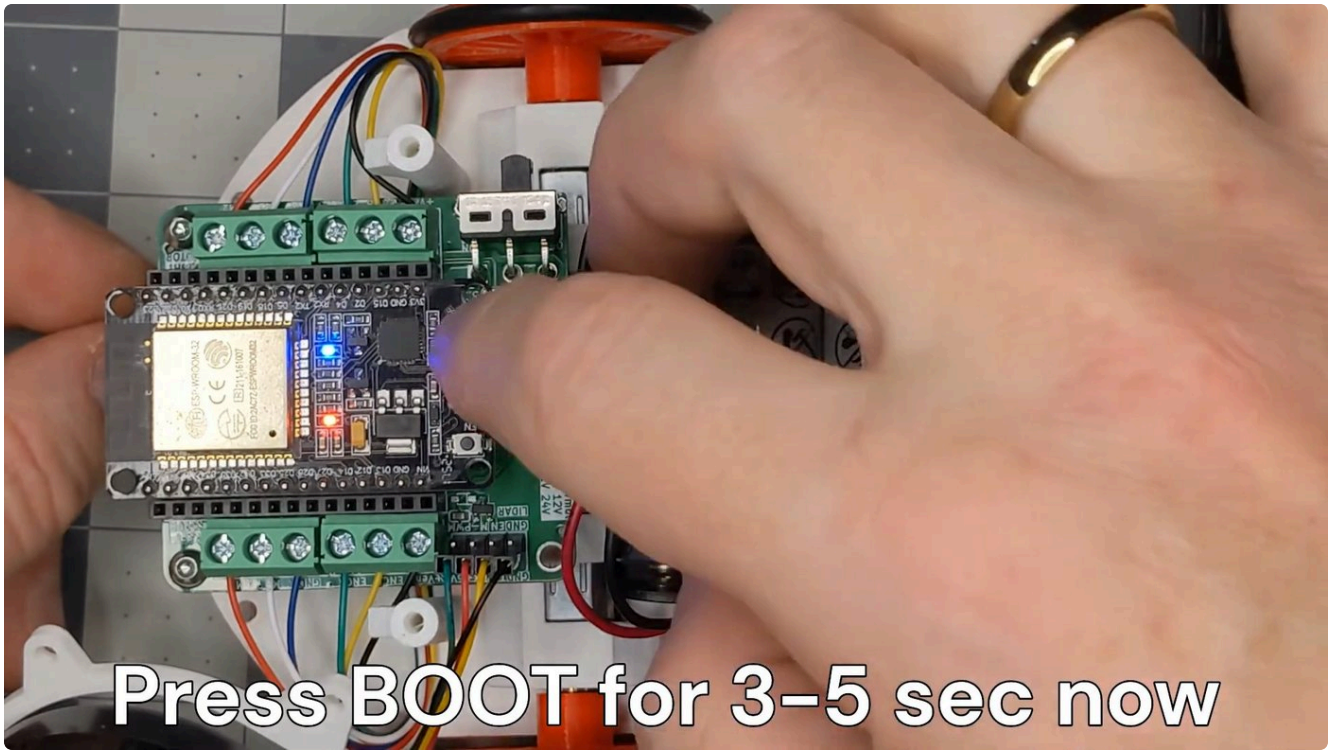
Plug the USB cable into the ESP32.

3. Select **Tools → Board → ESP32 Arduino → ESP32 Dev Module**.



Choose ESP32 Dev Module.

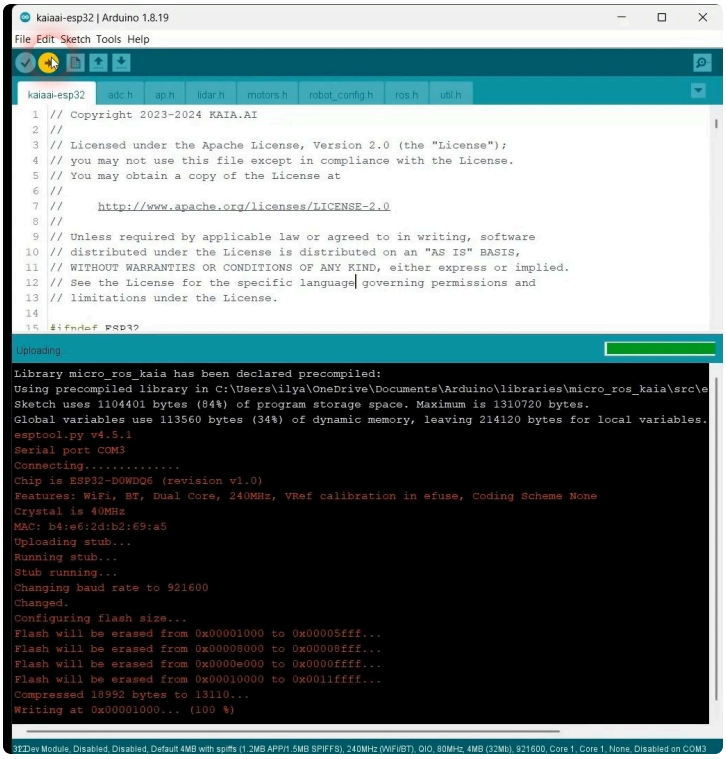
4. Select your COM port under **Tools → Port**.
5. Click **Upload** (the right arrow). The first compile takes a while.
6. When the output shows `Connecting....`, press and hold **BOOT** for 3 to 5 seconds.



Press BOOT for 3–5 seconds now.

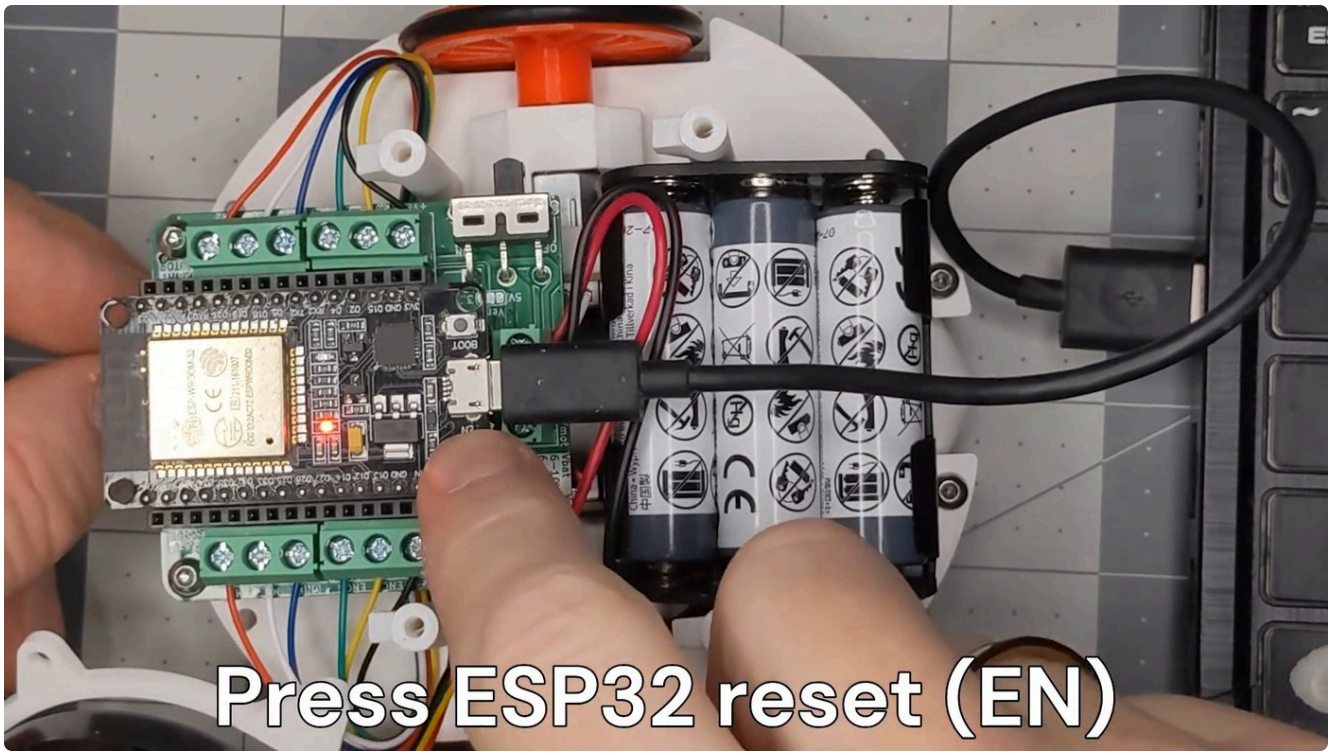
> **SINCE THE VIDEO WAS RECORDED**
> Some ESP32 boards don't need BOOT pressed; if the upload succeeds without it, that's fine. If it fails with `Failed to connect to ESP32: No serial data received`, retry and press BOOT as soon as `Connecting...` appears.

7. Wait for the upload to finish and confirm it succeeded: the output ends with `Hash of data verified.` and `Hard resetting via RTS pin...`.



Uploading the firmware in IDE 1.8.19.

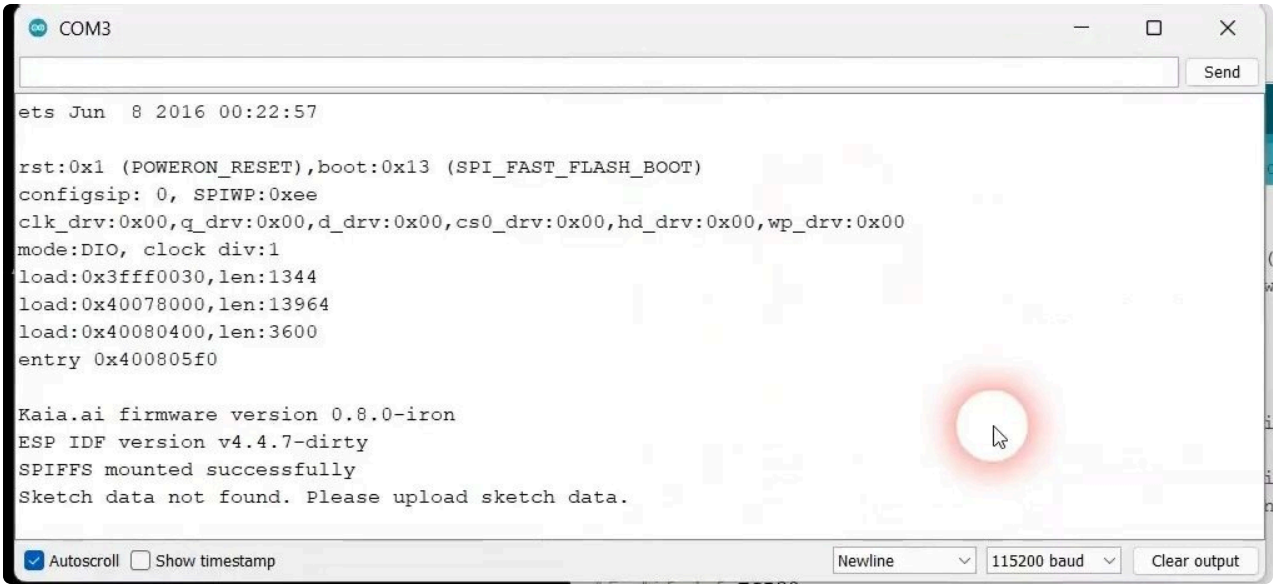
8. Open **Tools → Serial Monitor** and set the baud rate to **115200.**
9. Press the ESP32's **EN** (reset) button.



Press the ESP32 reset (EN) button.

10. Confirm the ESP32 boots the new firmware and asks for sketch data:

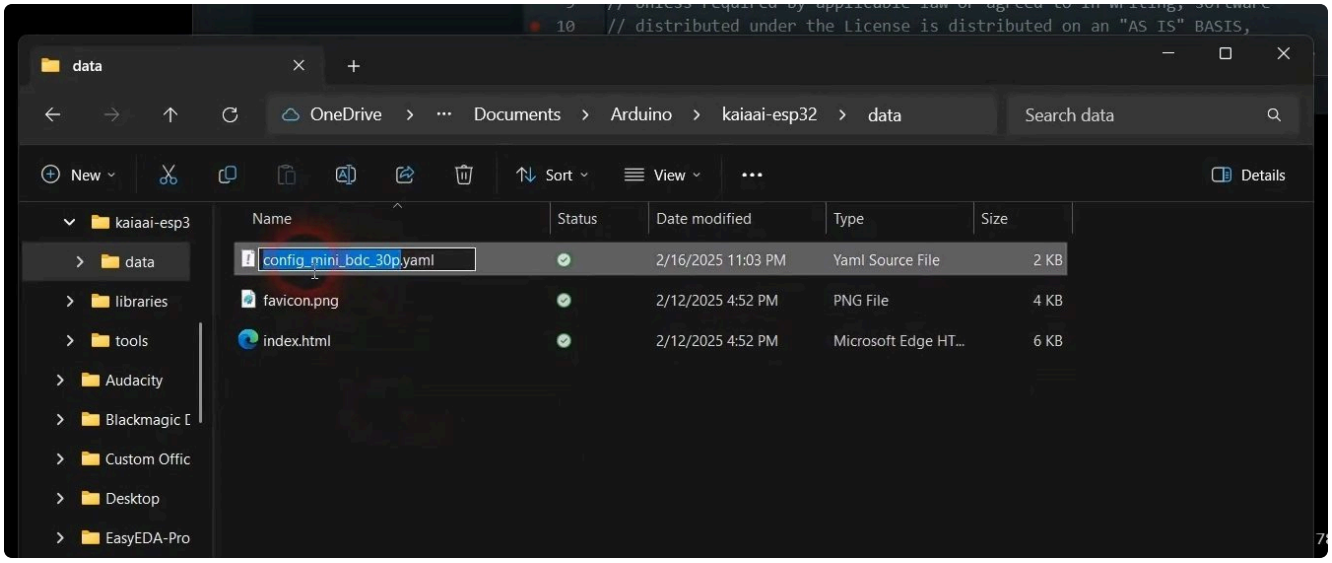
```
Kaia.ai firmware version 0.8.0-iron
ESP IDF version v4.4.7-dirty
SPIFFS mounted successfully
Sketch data not found. Please upload sketch data.
```



The ESP32 boots the new firmware.

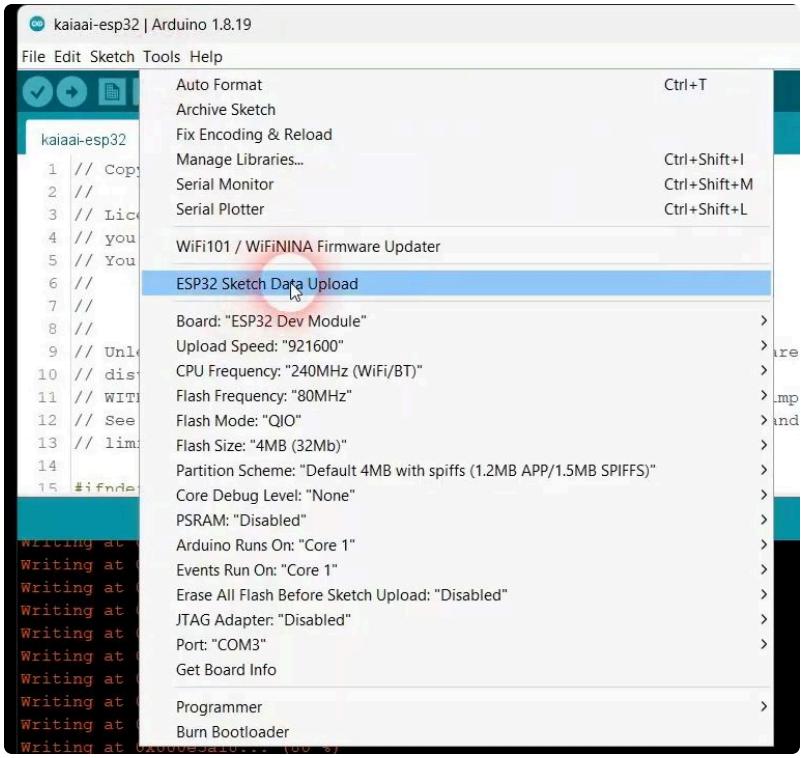
Upload the sketch data

1. **Close the Serial Monitor.** While it's open, it holds the COM port and the sketch data upload fails.
2. Open **Sketch → Show Sketch Folder** and go into the `data` folder.
3. Delete every `.yaml` file **except** `config_mini_bdc_30p.yaml`. Keep `favicon.png` and `index.html`.
4. Rename `config_mini_bdc_30p.yaml` to `config.yaml`.



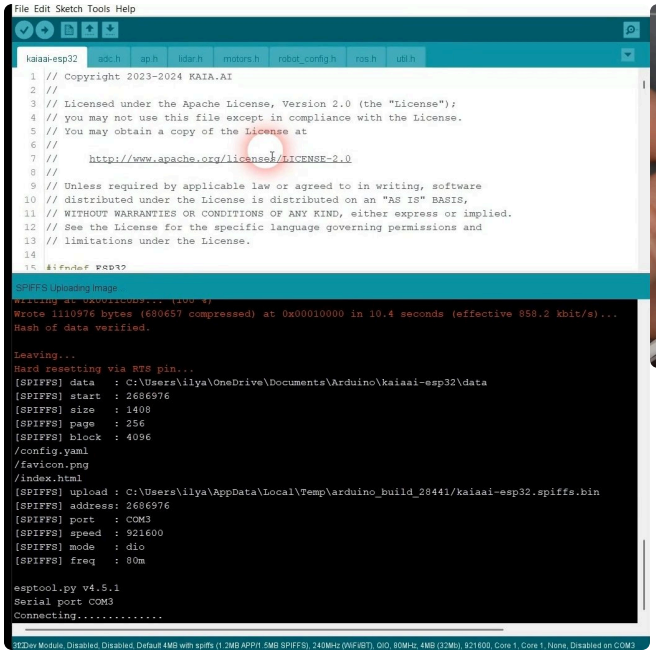
Rename it to config.yaml.

5. Run **Tools → ESP32 Sketch Data Upload.**

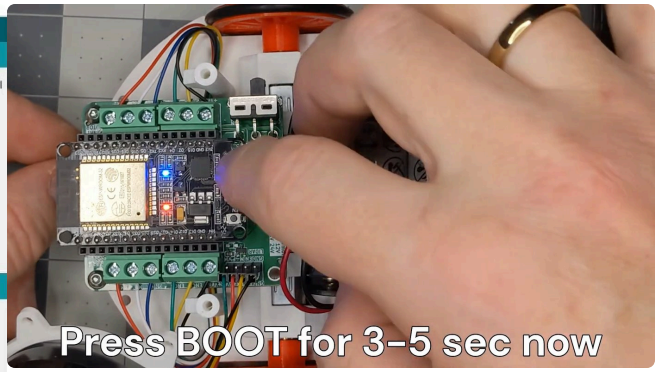


Tools → ESP32 Sketch Data Upload.

6. When the output shows `Connecting....`, press and hold **BOOT** for 3 to 5 seconds.

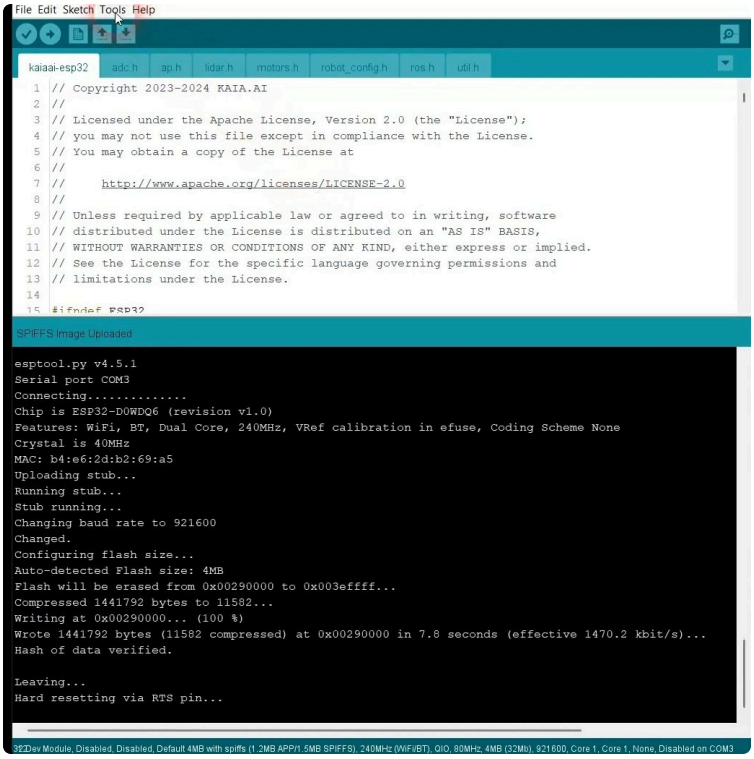


The output shows Connecting....



...press BOOT for 3–5 seconds now.

7. Confirm the upload succeeded: the status bar shows **SPIFFS Image Uploaded** and the output ends with `Hash of data verified.` and `Hard resetting via RTS pin...`.



Sketch data upload succeeded.

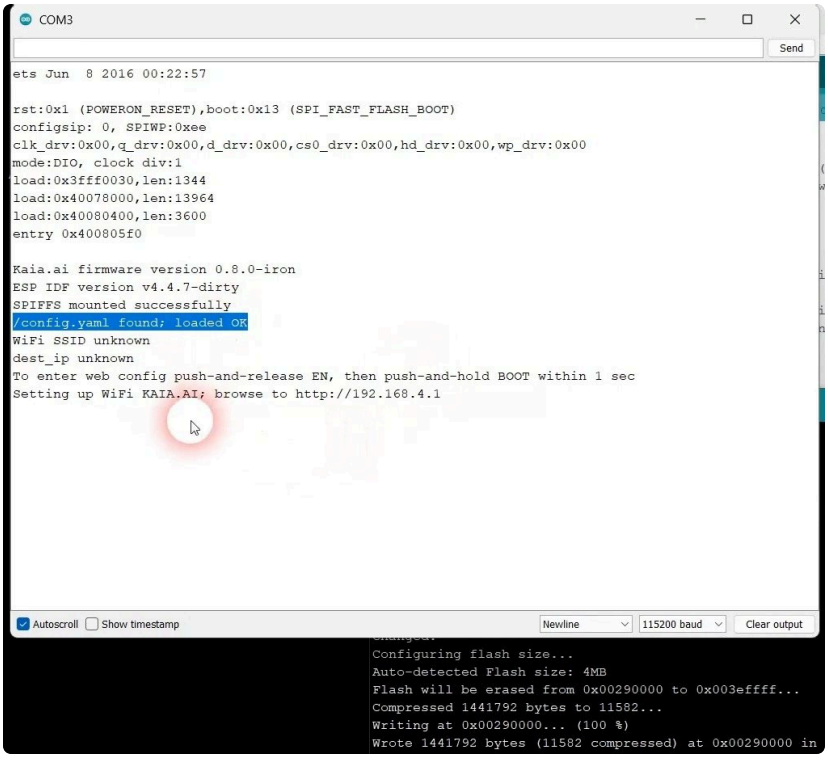
IF THE SKETCH DATA UPLOAD FAILS

- `Could not open COM3, the port doesn't exist` : close the Serial Monitor, check the USB cable, retry.
- `SPIFFS_write error(-10010): unknown / error adding file!` : a file name in `data` is over 30 characters.
- `This chip is ESP32-S3 not ESP32. Wrong --chip argument?` : wrong board selected in the IDE.

See Appendix B for more.

8. Reopen **Tools → Serial Monitor**.
9. Press the ESP32's **EN** (reset) button.
10. Confirm the ESP32 loads `config.yaml` and enters WiFi configuration mode:

```
SPIFFS mounted successfully
/config.yaml found; loaded OK
WiFi SSID unknown
dest_ip unknown
To enter web config push-and-release EN, then push-and-hold BOOT within 1 sec
Setting up WiFi KAIA.AI; browse to http://192.168.4.1
```

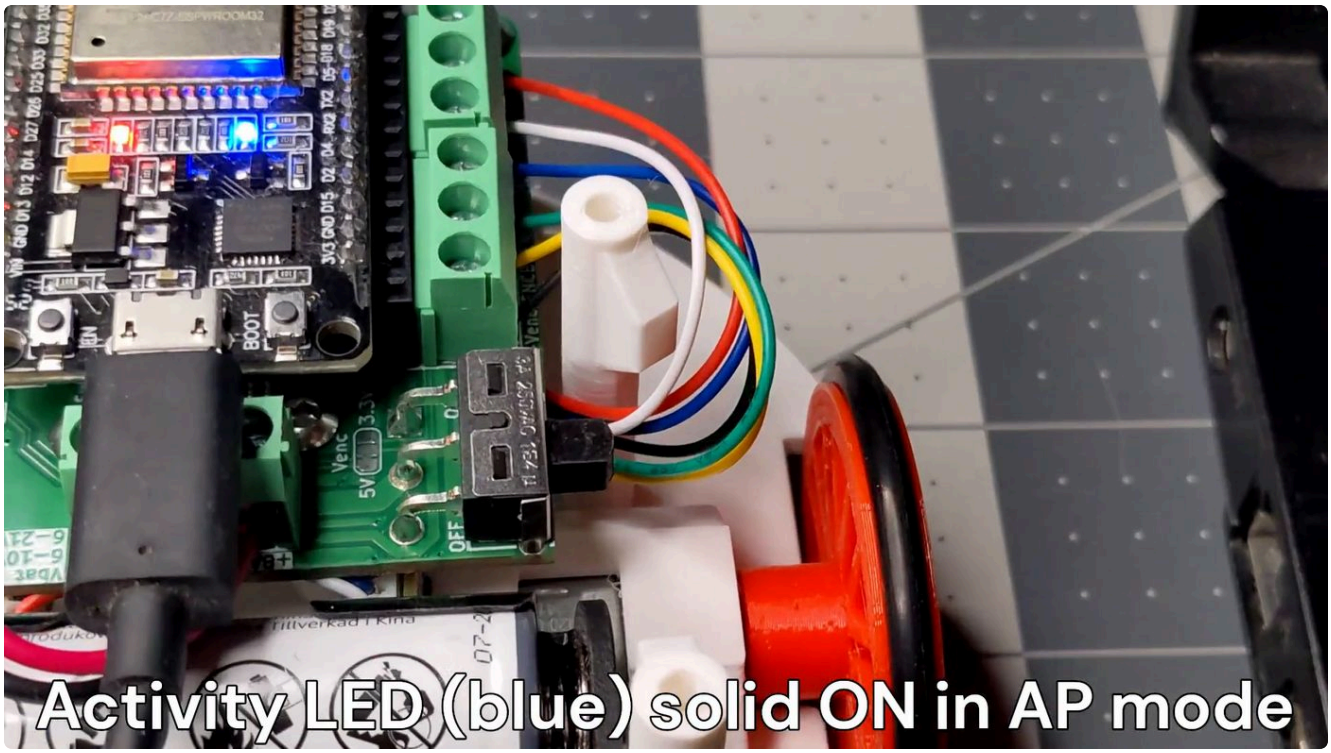


The ESP32 boots and loads config.yaml.

Configure the robot's WiFi

In configuration (AP) mode the ESP32 creates its own hotspot, `KAIA.AI`. Connect to it and use a web page to give the robot your WiFi name and password and your PC's IP address, where the ROS2 software runs.

1. The ESP32's blue activity LED is **solid on** in AP mode.



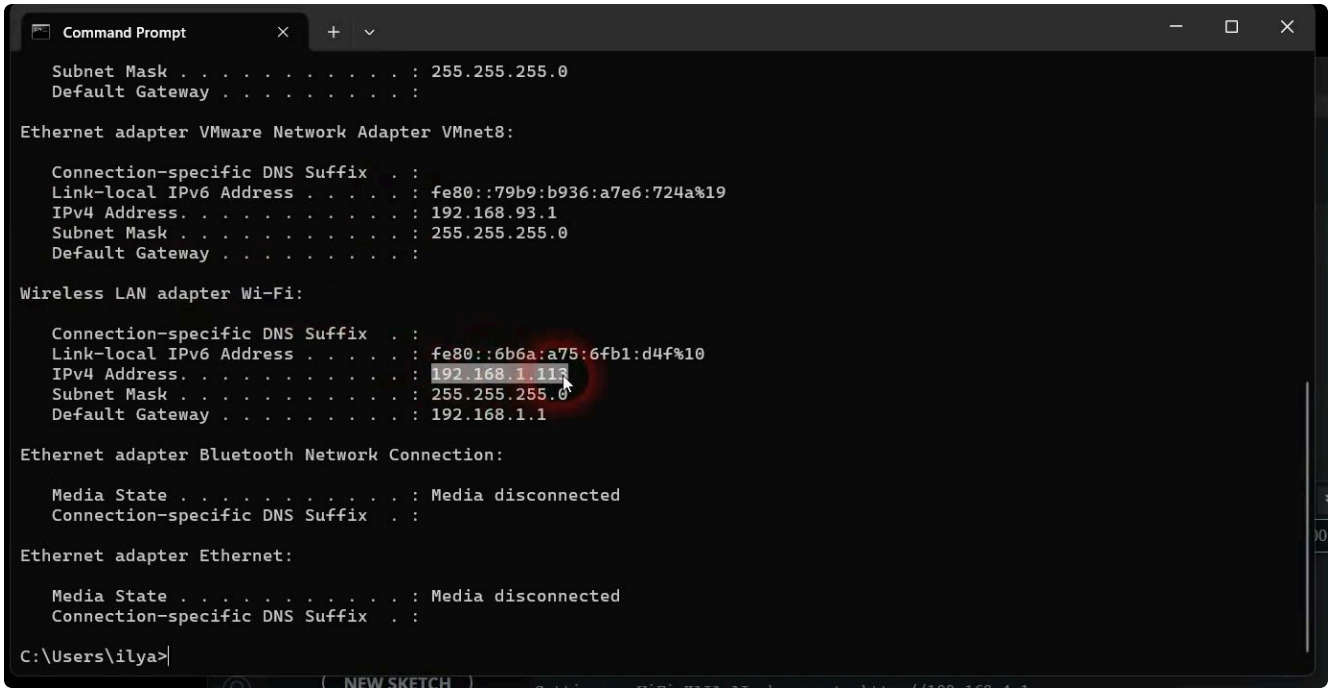
Activity LED solid on: AP mode.

Activity LED	What the robot is doing
Solid on	AP mode, waiting for WiFi configuration
Slow blinking, once per second	Connecting to your WiFi
Very slow blinking, once per 20 seconds	Connected to WiFi, connecting to the ROS2 PC
Fast blinking	Connected to the ROS2 PC

2. On your PC, in PowerShell or `cmd.exe`, run:

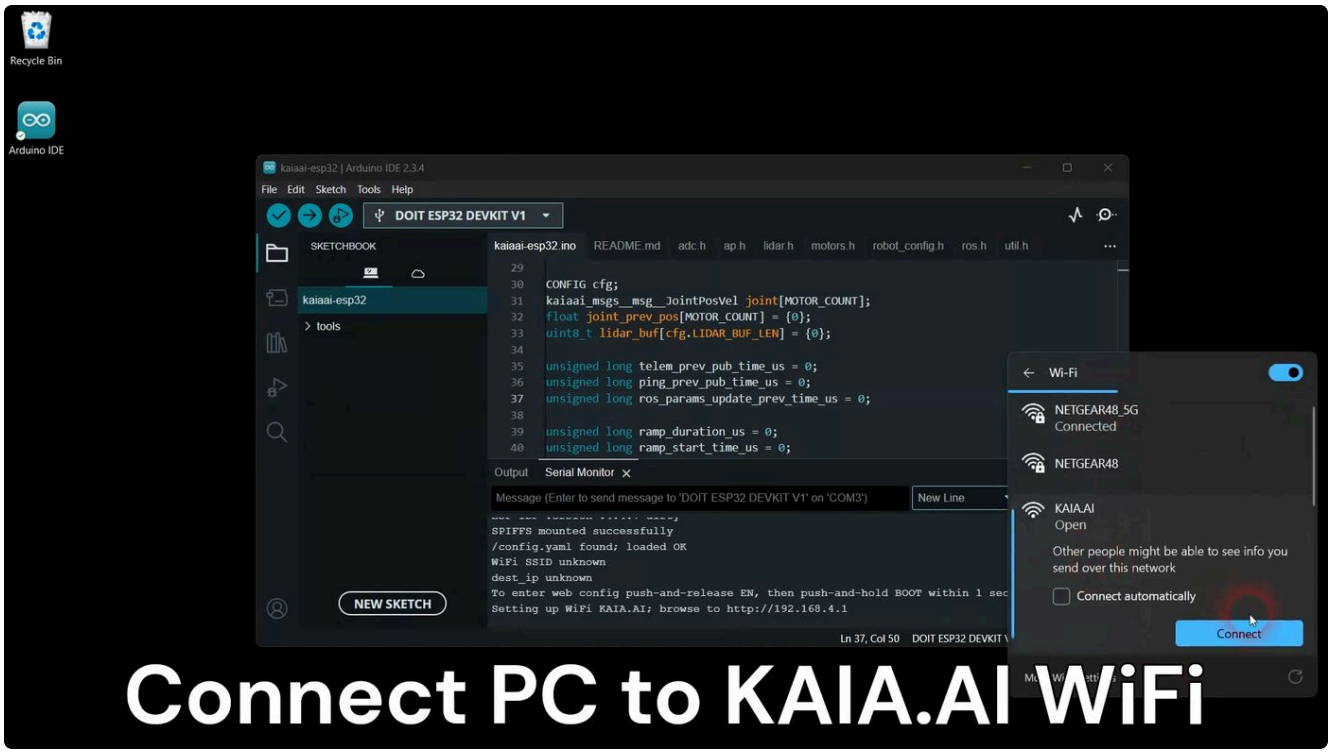
```
ipconfig
```

3. Note the **IPv4 Address** under **Wireless LAN adapter Wi-Fi** (`192.168.1.113` in the video), not a VMware or other virtual adapter.



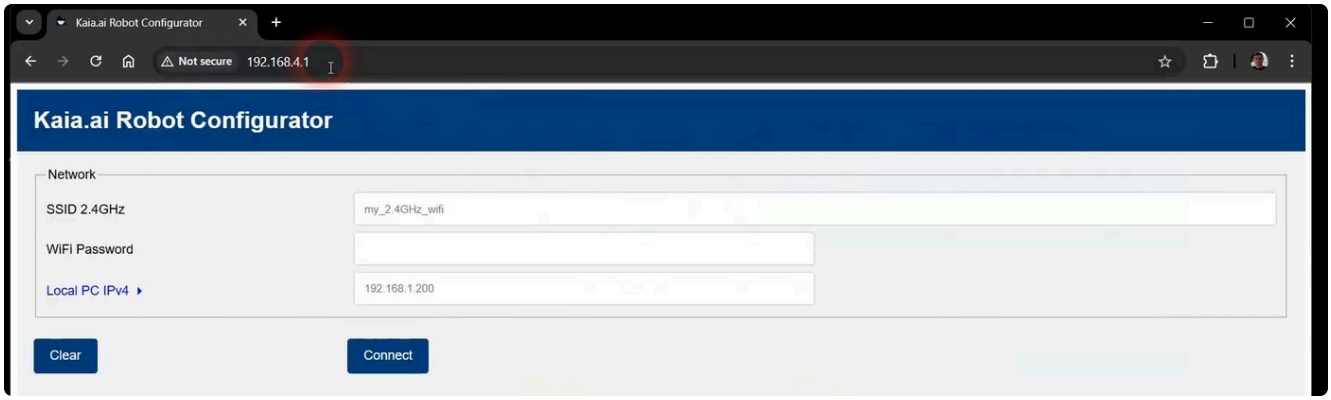
Note the Wi-Fi adapter's IPv4 Address.

4. Connect your PC to the open **KAIA.AI** network.



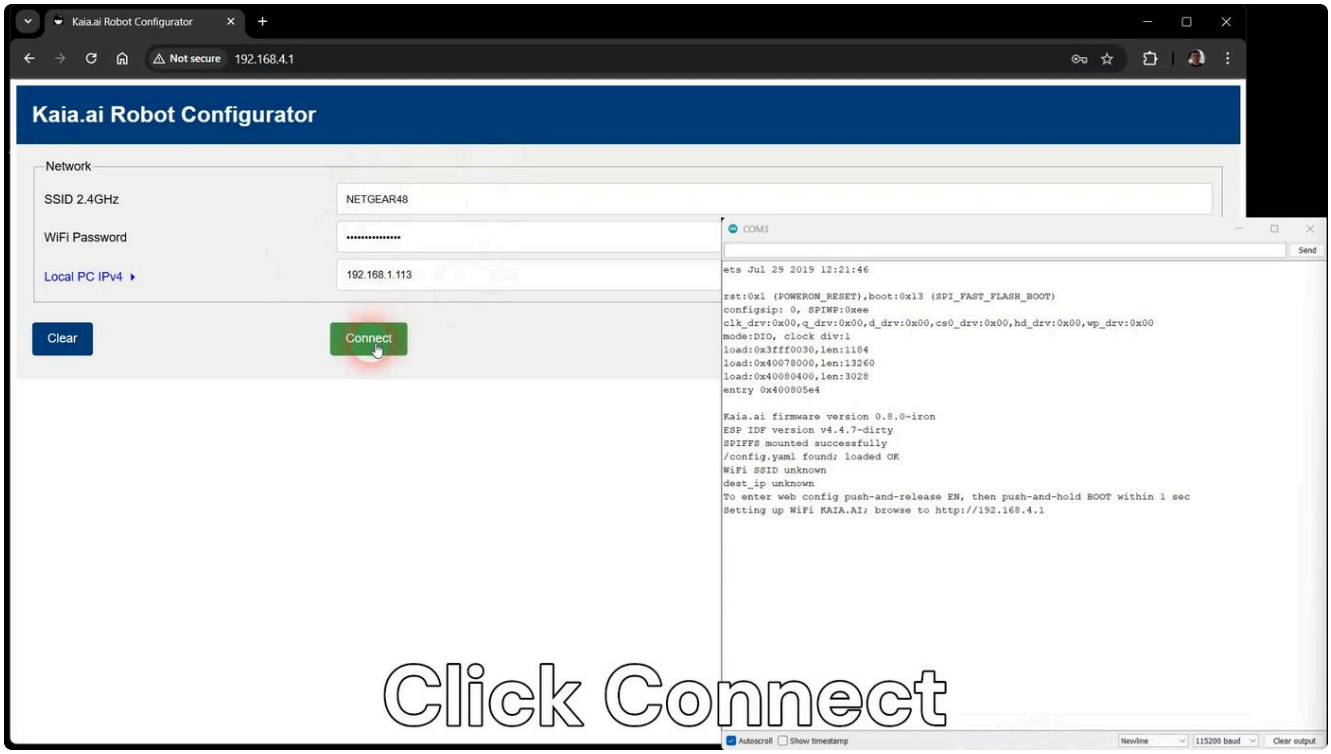
Connect to KAIA.AI.

5. Browse to `http://192.168.4.1` to open the **Kaia.ai Robot Configurator.**



The Kaia.ai Robot Configurator.

6. Enter your WiFi name in **SSID 2.4GHz** and its password in **WiFi Password**.
7. Enter your PC's IPv4 address from step 3 in **Local PC IPv4** and click **Connect**.

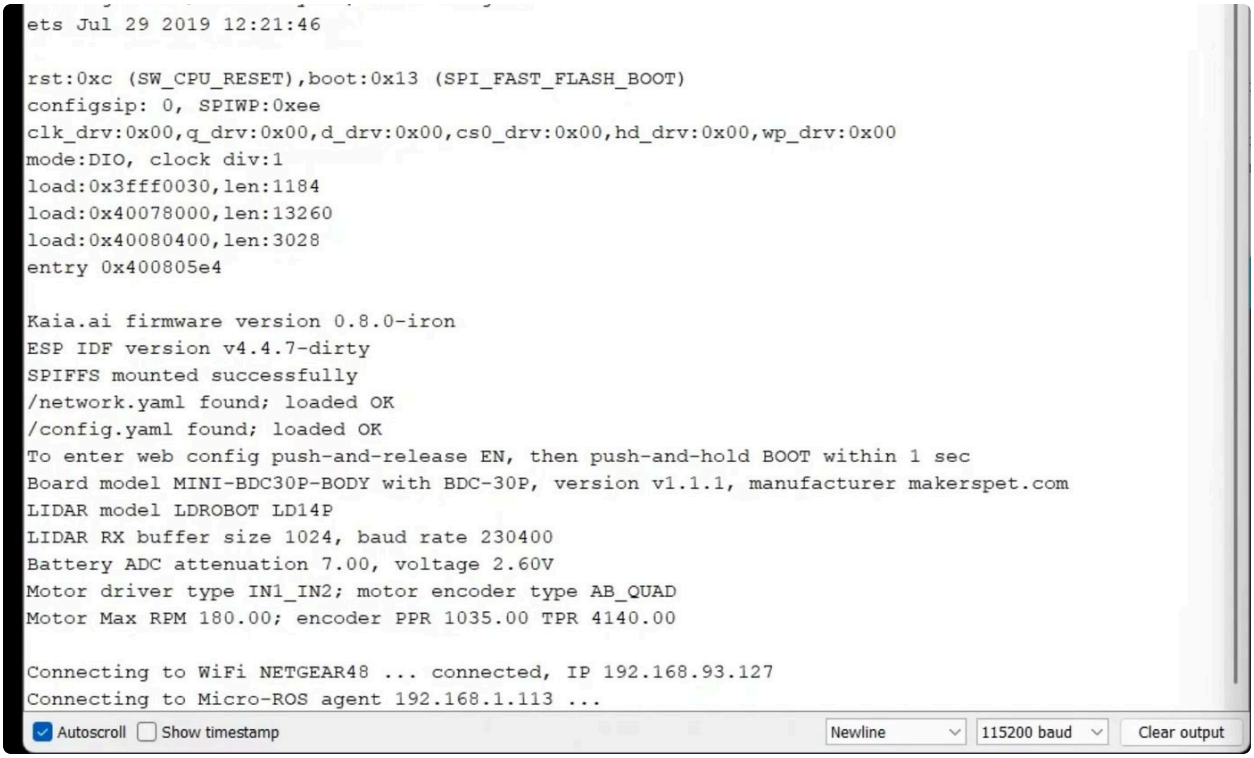


Fill in WiFi and PC IP, then click Connect.

> **2.4 GHZ WIFI ONLY**
> The ESP32 can't use 5 GHz. If your router has separate network names, enter the 2.4 GHz one (`NETGEAR48` , not `NETGEAR48_5G` , in the video).

8. The page shows **Connecting to WiFi...** with the saved settings (`dest_port 8888` is the port the robot uses to reach your PC). The robot saves them to `/network.yaml` , restarts and joins your WiFi:

```
/network.yaml found; loaded OK
/config.yaml found; loaded OK
Board model MINI-BDC30P-BODY with BDC-30P, version v1.1.1, manufacturer makerspet.com
LIDAR model LDROBOT LD14P
Motor driver type IN1_IN2; motor encoder type AB_QUAD
Connecting to WiFi NETGEAR48 ... connected, IP 192.168.93.127
```



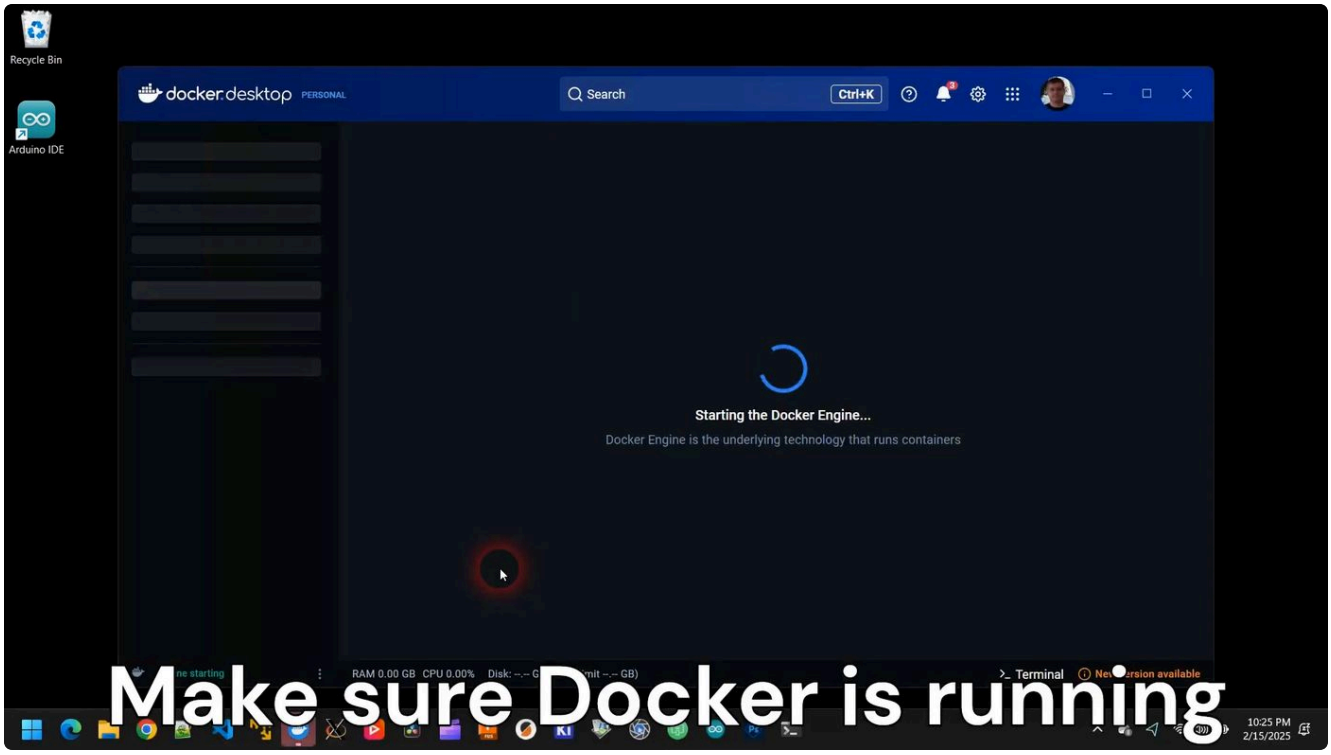
The robot restarted and joined your WiFi.

9. Make sure your PC has reconnected to your WiFi before proceeding.

> **IF YOUR PC'S IP ADDRESS CHANGES**
> For example after a router restart, the robot joins WiFi but can't find the PC. Reset the robot's WiFi settings (end of this chapter) and enter the new address.

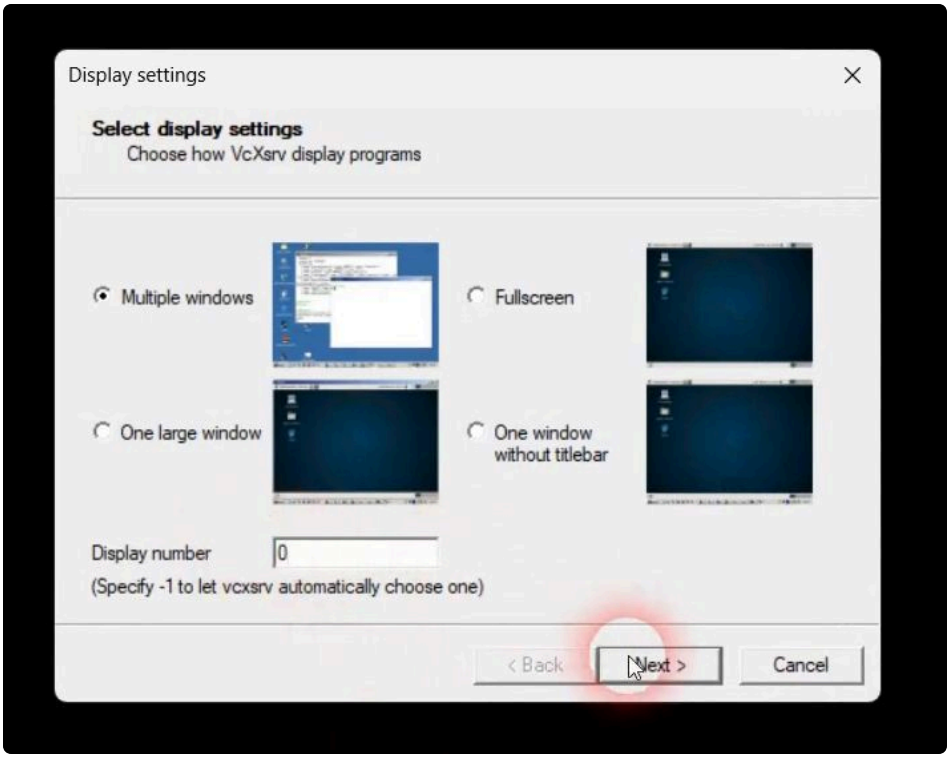
Launch the robot software on your PC

1. Make sure Docker Desktop is running.



Docker Desktop must be running.

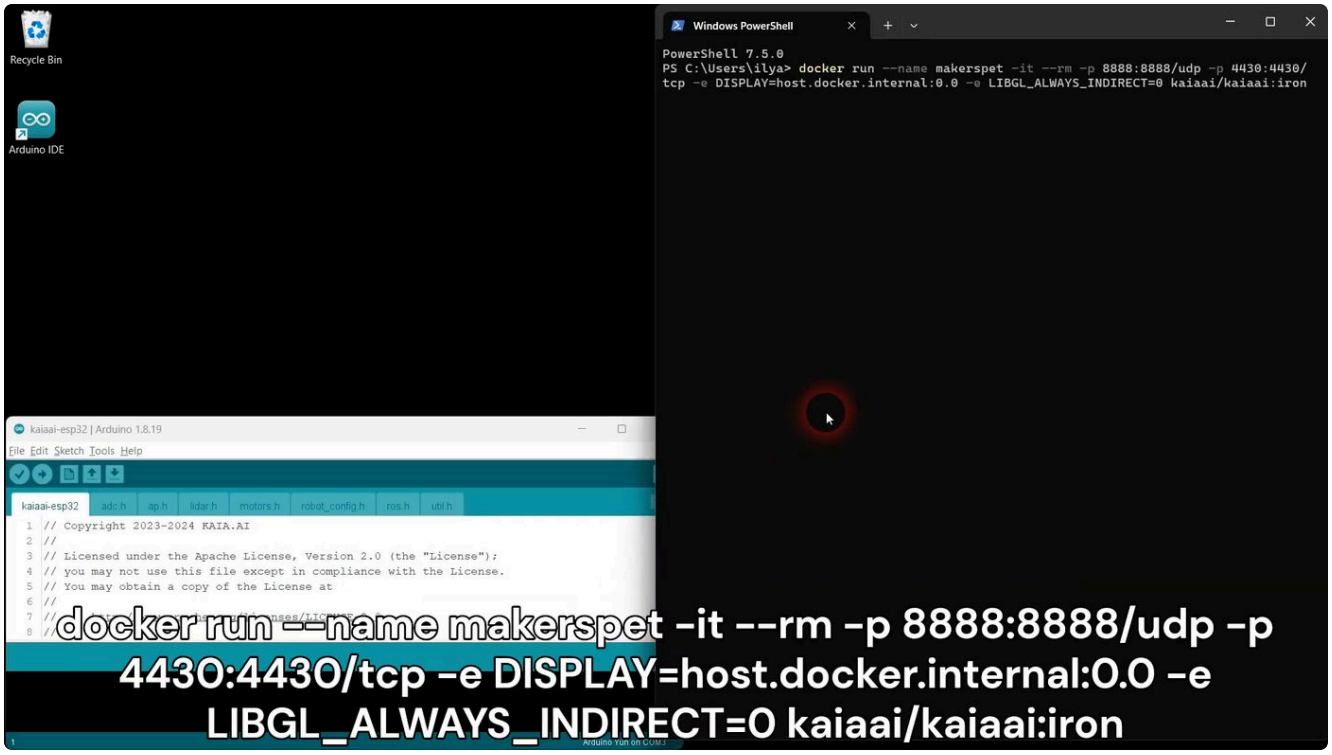
2. Make sure your X server is running; it shows the robot software's windows (RViz) on your desktop. In the video it's VcXsrv, started with XLaunch: set **Display number** to `0`, then click **Next** through to **Finish**.



In XLaunch, set Display number to 0.

3. Open Windows PowerShell and start the robot software container:

```
docker run --name makerspet -it --rm -p 8888:8888/udp -p 4430:4430/tcp -e DISPLAY=host.doc
ker.internal:0.0 -e LIBGL_ALWAYS_INDIRECT=0 kaiaai/kaiaai:iron
```



Start the kaiaai/kaiaai:iron container.

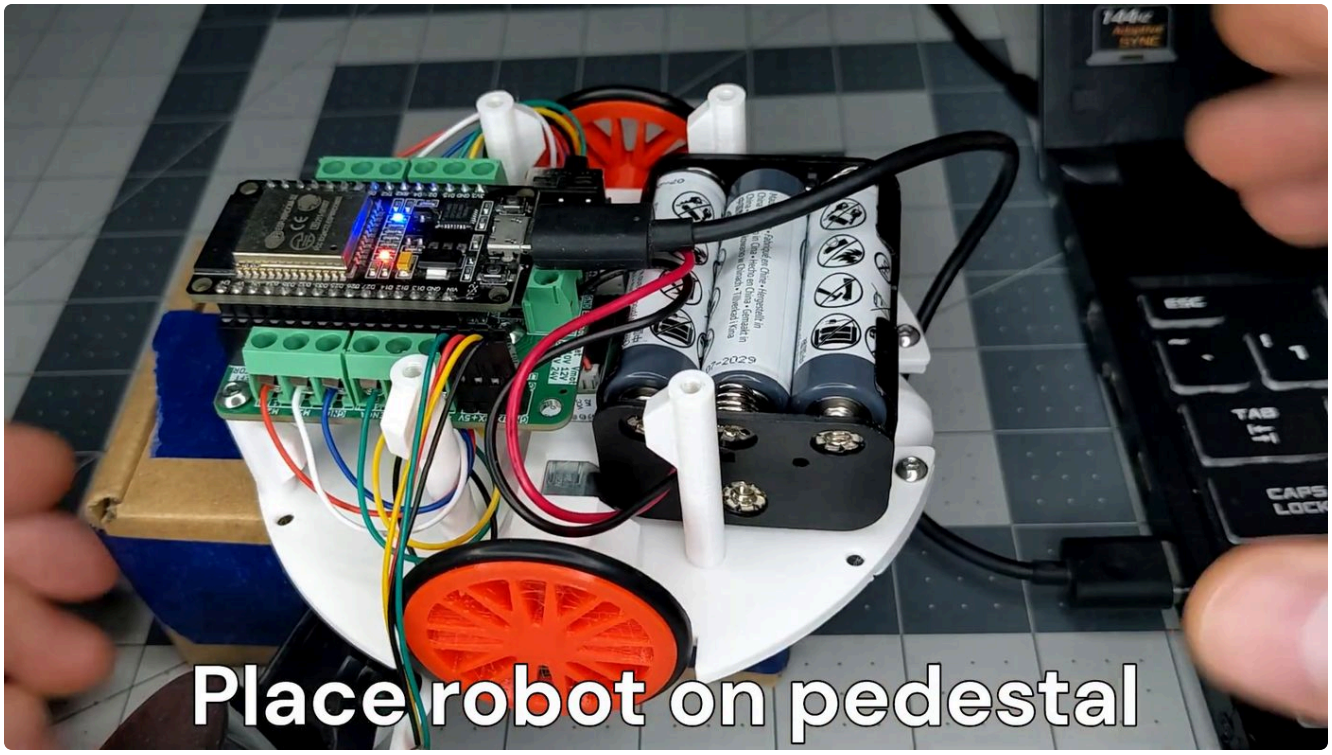
The prompt changes to something like `root@6028bbee19de:/ros_ws#`: a Linux shell inside the container. The robot talks to the PC on port 8888/udp, hence `dest_port 8888`.

IF DOCKER RUN FAILS

- `error during connect: ... dockerDesktopLinuxEngine ...`: launch Docker Desktop.
- `Conflict. The container name "/makerspet" is already in use`: your last session's container is still running. Run `docker container stop makerspet` and retry. Next time, leave each shell with `exit` instead of closing the window.
- `Ports are not available: exposing port TCP 0.0.0.0:4430`: reboot your PC.

Bring up the robot and test the motors

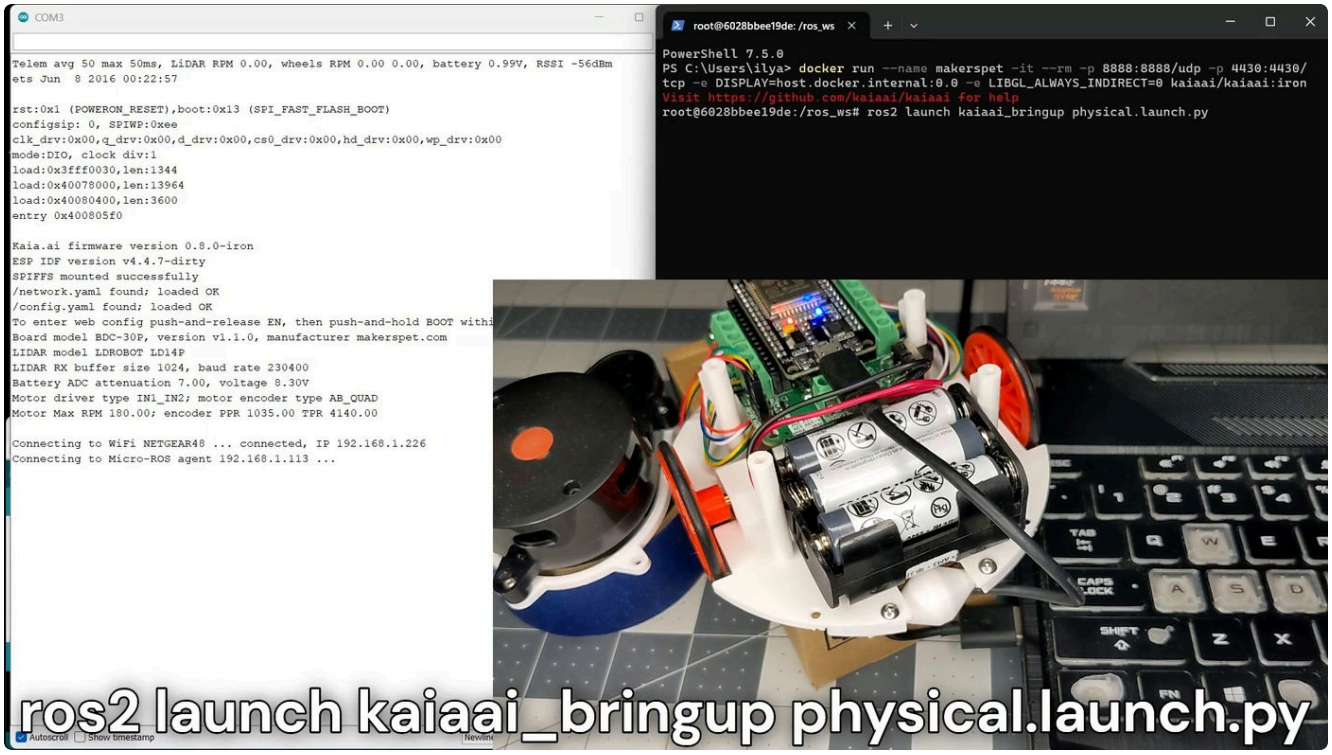
1. Open the Arduino Serial Monitor to watch the robot's messages.
2. Put the robot on a pedestal, such as a small box, so the wheels spin freely.



Wheels in the air.

3. Turn the power switch **on** and press **EN**. The robot joins your WiFi and tries to reach your PC (`Connecting to Micro-ROS agent 192.168.1.113 ...`). If WiFi fails, press **EN** again.
4. In the container shell, run:

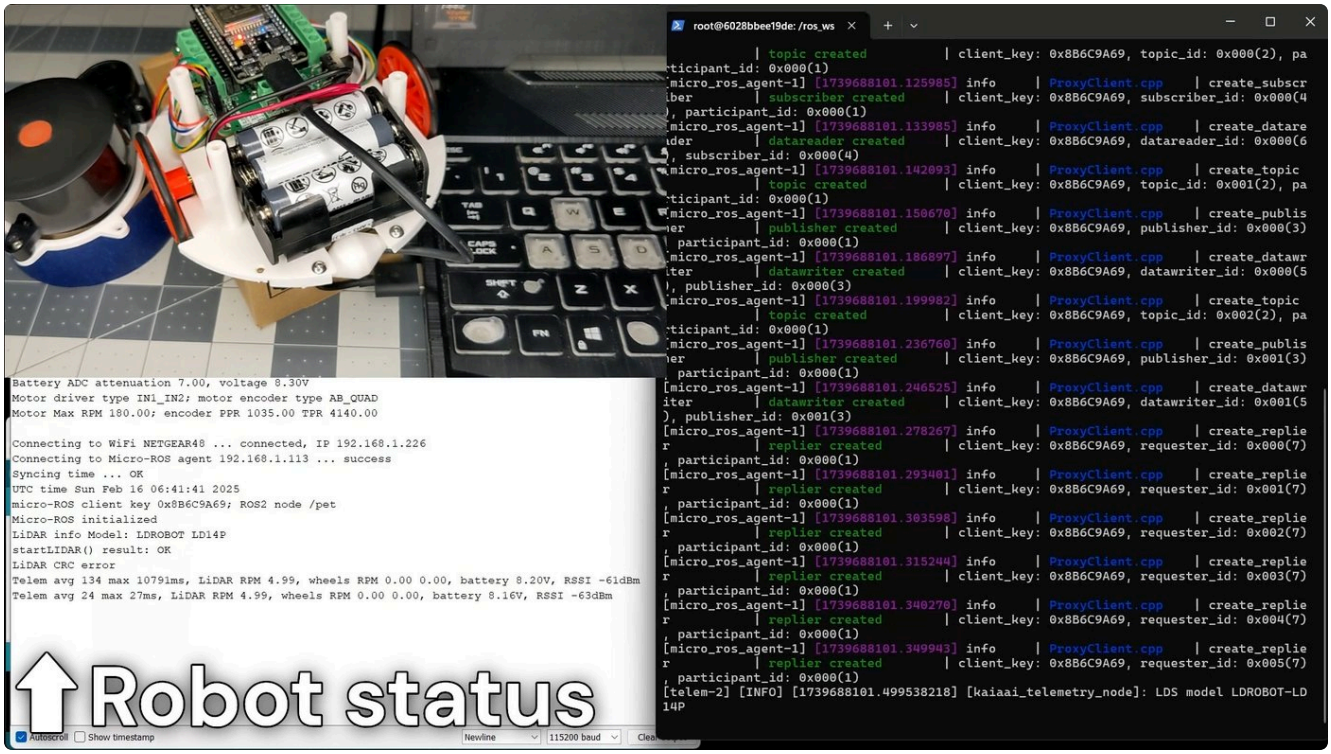
```
ros2 launch kaiaai_bringup physical.launch.py
```



Run the physical robot launch.

5. The robot connects, and the Serial Monitor prints periodic status:

```
Connecting to Micro-ROS agent 192.168.1.113 ... success
Syncing time ... OK
micro-ROS client key 0x8B6C9A69; ROS2 node /pet
Micro-ROS initialized
LiDAR info Model: LDROBOT LD14P
startLIDAR() result: OK
Telem avg 24 max 27ms, LiDAR RPM 5.01, wheels RPM 0.00 0.00, battery 8.14V, RSSI -62dBm
```



The robot reports its status in the Serial Monitor.

6. The activity LED now blinks rapidly.

READ THE STATUS LINE

`Telem` shows LiDAR speed, wheel speeds, battery voltage and WiFi signal (RSSI). **LiDAR RPM must not be zero.** If it stays `0.00`, check the LiDAR wires and that `config.yaml` matches your LiDAR (LD14P by default).

SINCE THE VIDEO WAS RECORDED: ANTIVIRUS

Antivirus software can block the robot's connection to your PC. My Avast started blocking these ports in December 2025. If the robot joins WiFi but never connects to the PC, try disabling your antivirus.

CONNECTED TO WIFI, BUT NOT TO THE PC?

If `Connecting to Micro-ROS agent ...` repeats without `success`:

- Check that `physical.launch.py` is running in the container.
- Check the robot and PC are on the same network, and it doesn't block devices from talking to each other ("client isolation"), as café, restaurant and university networks often do.
- Check your PC's IP address hasn't changed.
- Make sure your PC accepts incoming local connections and your antivirus isn't blocking them (see the box above).

See Appendix B for more.

7. In Windows Terminal, press **Alt+Shift+-** to split the window horizontally into a second PowerShell pane.
8. Click inside it and open another shell in the running container:

```
docker exec -it makerspet bash
```

9. Run the teleoperation app:

```
ros2 run kaiaai_teleop teleop_keyboard
```

```
PowerShell 7.5.0
PS C:\Users\ilya> docker exec -it makerspet bash
root@6028bbee19de:/ros_ws# ros2 run kaiaai_teleop teleop_keyboard
YAML file name : /ros_ws/install/makerspet_mini/share/makerspet_mini/config/teleop_ke
yboard.yaml
Max linear velocity 0.400        Max angular velocity 7.720
Control Kaia.ai-compatible Robot
---------------------------------
Moving around:
      w
 a    s    d
      x
w/x   : increase/decrease linear  velocity
a/d   : increase/decrease angular velocity
s     : keep straight
CAPS  : large step
Space : force stop
CTRL-C to quit
```

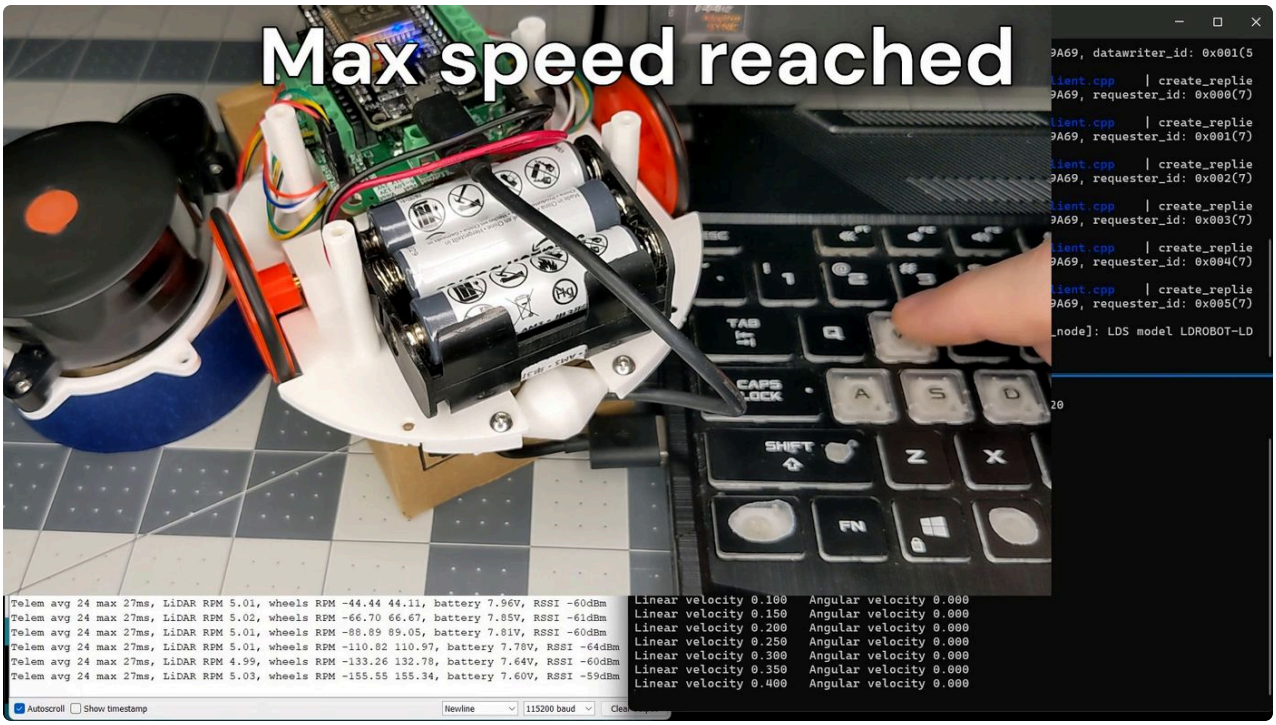
A second shell in the container runs teleop_keyboard.

ADVANCED: COMMAND HISTORY

In a container shell, press the **up** and **down** arrow keys to step through common robot commands. They were preloaded into the bash history when the Docker image was built.

10. With the teleop pane selected, drive the wheels:

- Press **w** repeatedly to speed up forward to maximum. Watch the wheel RPM rise in the Serial Monitor.
- Press **x** several times to slow down and reverse.
- Press **space** to stop, then **d** to spin right.
- Press **space** to stop, then **a** to spin left.
- Press **space** to stop.

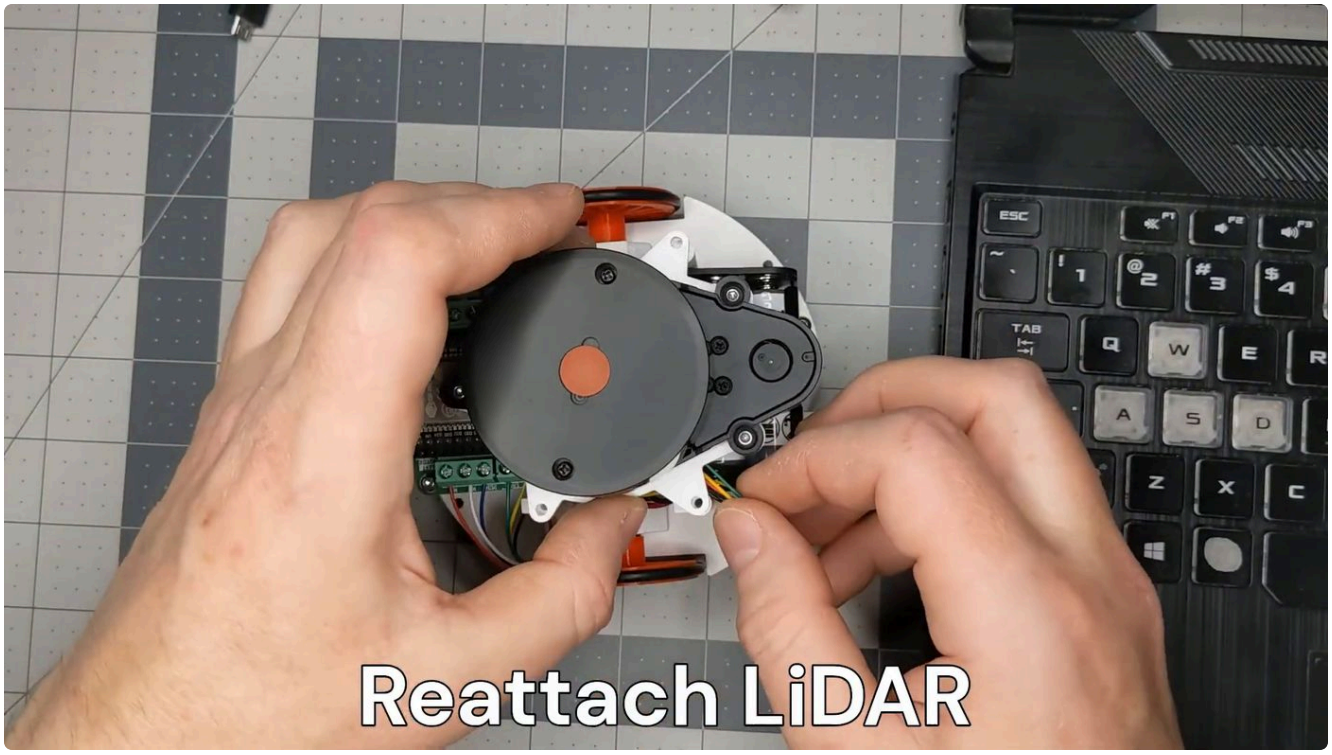


The wheels at maximum speed.

If both wheels turn the right way, the motors work. If a wheel doesn't turn, turns the wrong way or only runs at full speed, see Appendix B.

Check the LiDAR

1. Turn the power off and unplug the USB cable.
2. Reattach the LiDAR to the LiDAR posts with **hex button M3 screws**.



Reattach the LiDAR.

3. Click inside a PowerShell pane and press **Alt+Shift+-** to split it. If a pane opens in the wrong place, type `exit` in it and retry.
4. Open another container shell and start the LiDAR visualization:

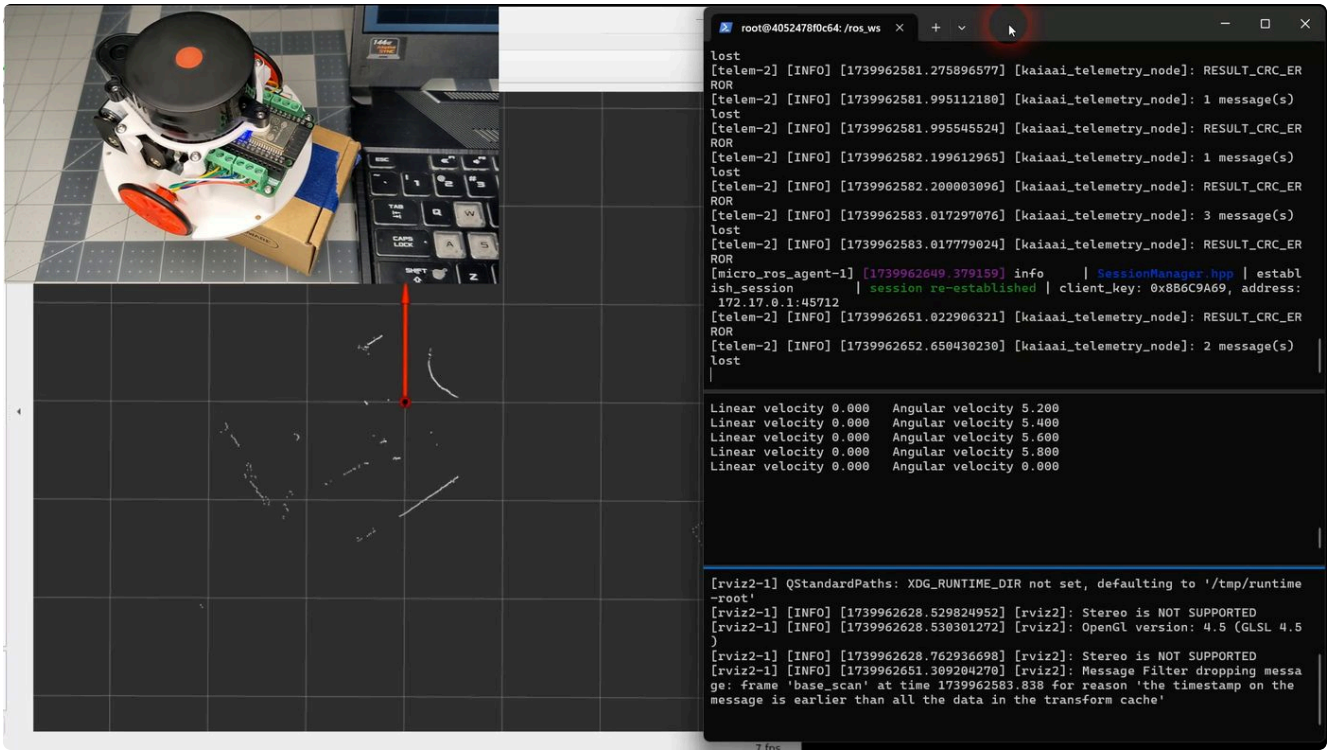
```
docker exec -it makerspet bash
ros2 launch kaiaai_bringup monitor_robot.launch.py
```



A third shell opens RViz.

An empty RViz window opens.

5. Turn the robot on. The white dots in RViz are what the LiDAR sees; the red arrow shows which way the robot faces.



Live LiDAR scan in RViz.

RVIZ DOESN'T OPEN?

`could not connect to display host.docker.internal:0.0` means your X server isn't running. Start it (display number 0) and rerun the command.

NOTE

Occasional `message(s) lost` and `RESULT_CRC_ERROR` lines are normal over WiFi. To keep them rare, give the robot a strong WiFi signal and keep the network quiet while the robot runs: for example, don't stream video on another PC or phone on the same WiFi.

Shut down

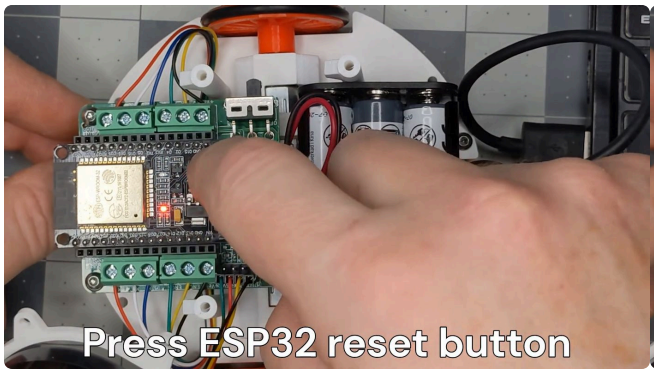
1. Turn the robot off.
2. Press **Ctrl+C** in each shell.
3. Type `exit` in each shell. Exiting the `docker run` shell stops the container, and `--rm` removes it so the name is free next time.

Next, you'll map your place and make the robot self-drive around obstacles.

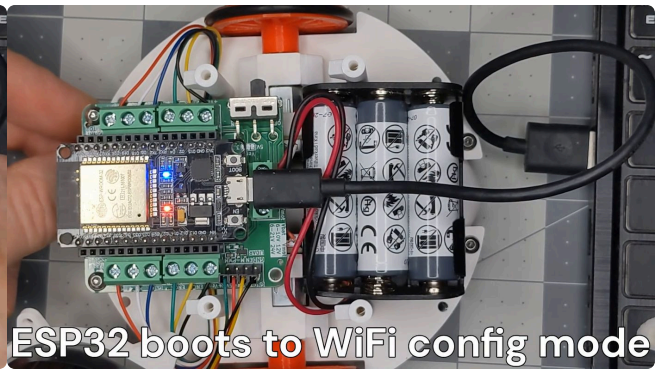
Reset the robot's WiFi settings

To change the WiFi network or your PC's IP address:

1. Power up the ESP32 (USB or battery).
2. Press **EN**, then within 1 second press and hold **BOOT**. The LED blinks very fast.
3. Keep holding until the LED turns solid on.
4. Release **BOOT**. The ESP32 reboots into AP mode; configure WiFi again as above.



Press EN, then hold BOOT.



Release BOOT: WiFi configuration mode.

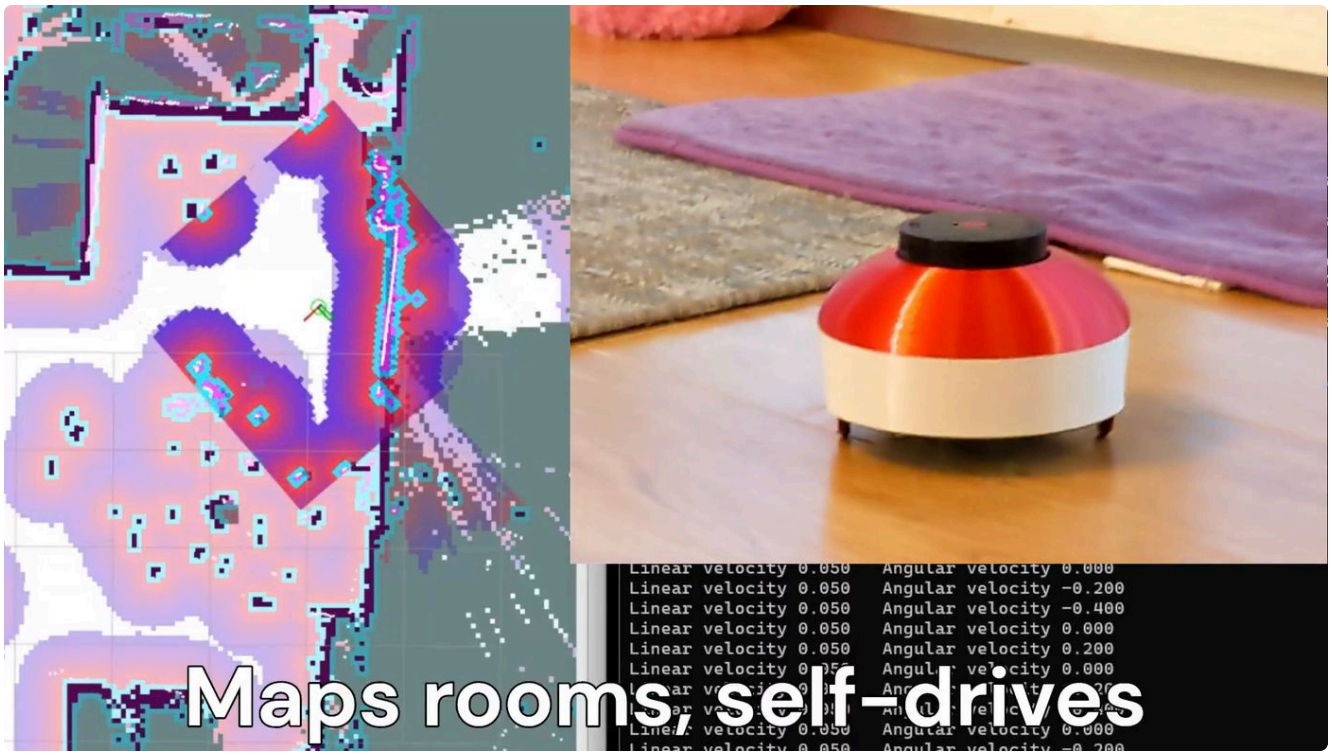
Full documentation and troubleshooting: makerspet.com/blog/bld-120mm-pack/ and Appendix B.

4. Map Your Place by Driving the Robot

Video: Map your room Step by Step using Arduino/ROS2 self-driving robot

In this chapter you drive the robot around your home from the PC keyboard while ROS2 builds a map from its LiDAR scans, then save the map to your PC. In the next chapter the robot uses this map to drive itself.

You need the robot assembled, flashed and connected (chapters 1–3), plus Docker Desktop, VcXsrv and PowerShell on your Windows PC. Mapping one or two rooms takes about 15–30 minutes.



The robot maps rooms with its LiDAR, then drives itself using that map.

Prepare your place

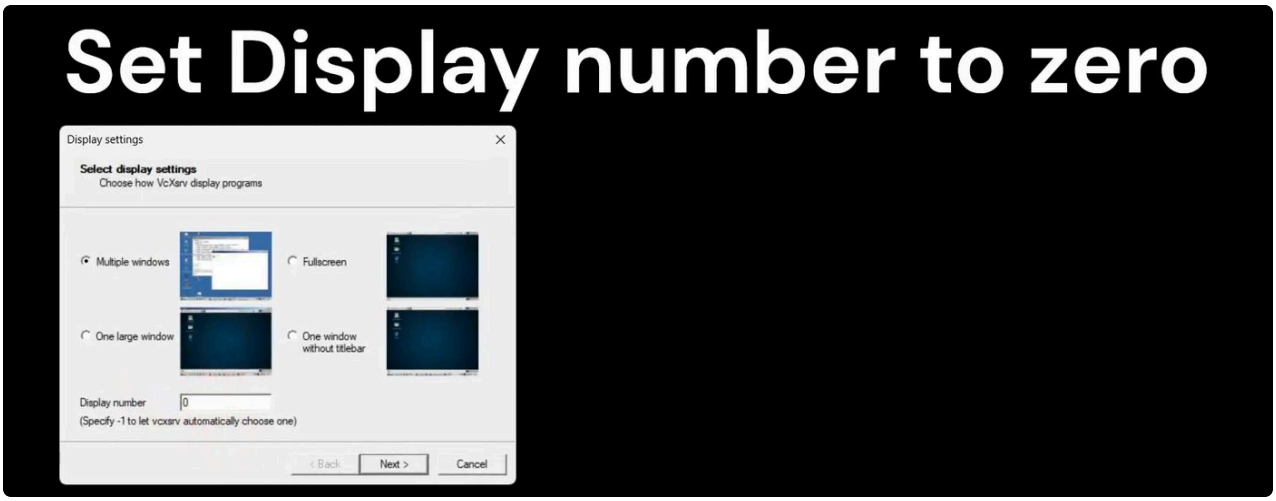
A LiDAR sees one thin horizontal slice at the height of its laser, so some things are invisible to it.

SINCE THE VIDEO WAS RECORDED

- Block off objects the LiDAR cannot detect, for example with cardboard: black (non-reflective), transparent, too low to reach the laser plane, very thin or sparse (fine mesh), lit brightly by near-infrared light (sunlight, incandescent bulbs), and mirrors or highly reflective objects.
- The robot can't drive over thick carpet or thresholds higher than about 5 mm (1/5″).
- Keep the scene static while mapping; for example, don't open or close doors.
- Declutter and leave plenty of space between obstacles.
- Block off angled ramps. A ramp tilts the laser plane and spoils the map.

Launch the robotics software

1. Make sure Docker Desktop is running.
2. Start the X server, which RViz needs to open its window: run XLaunch, keep **Multiple windows**, set **Display number** to `0`, click **Next** through the rest and **Finish**. Its icon appears in the system tray.

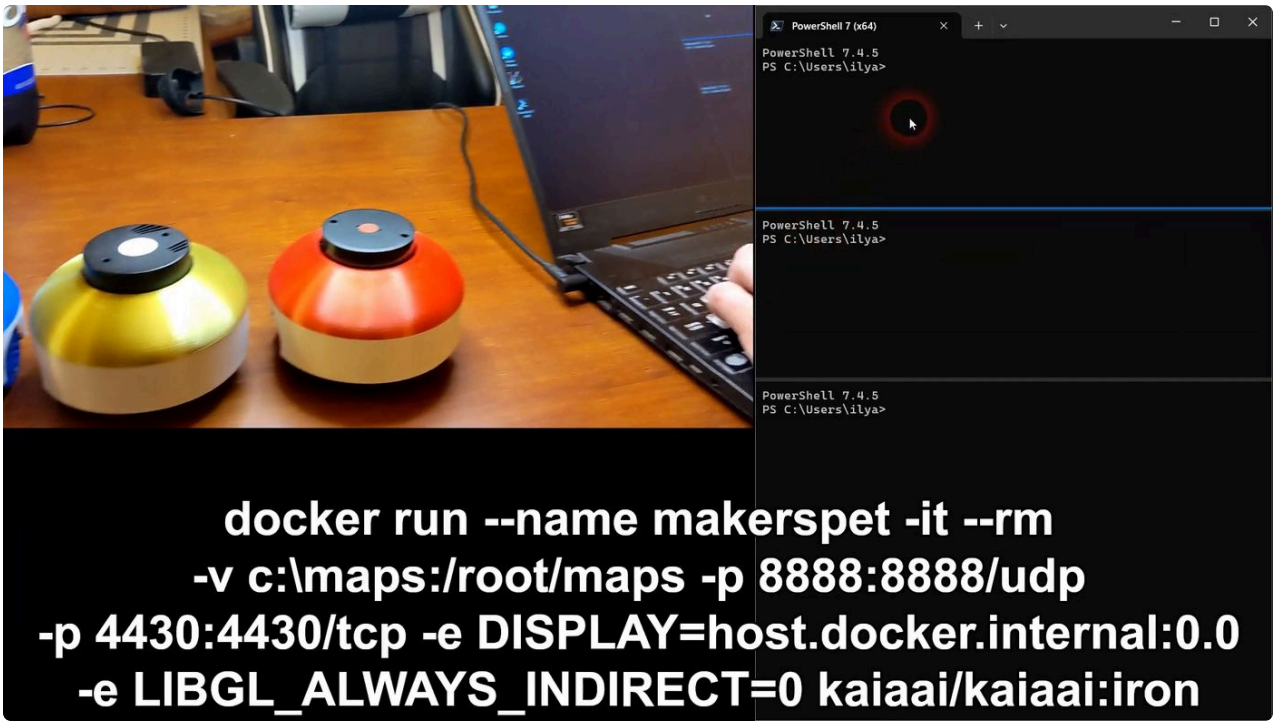


In XLaunch, set the Display number to zero.

3. Launch Windows PowerShell.
4. Click in the PowerShell window and press **Alt+Shift+-** to split it horizontally. Repeat until you have three panes: top, middle and bottom.
5. In the **top** window, launch the robotics software:

```
docker run --name makerspet -it --rm -v c:\maps:/root/maps -p 8888:8888/udp -p 4430:4430/t
cp -e DISPLAY=host.docker.internal:0.0 -e LIBGL_ALWAYS_INDIRECT=0 kaiaai/kaiaai:iron
```

Compared with Chapter 3, this adds `-v c:\maps:/root/maps`, which shares `C:\maps` on Windows with `/root/maps` (`~/maps`) in the container. Without it, `--rm` would delete your map when the container exits.

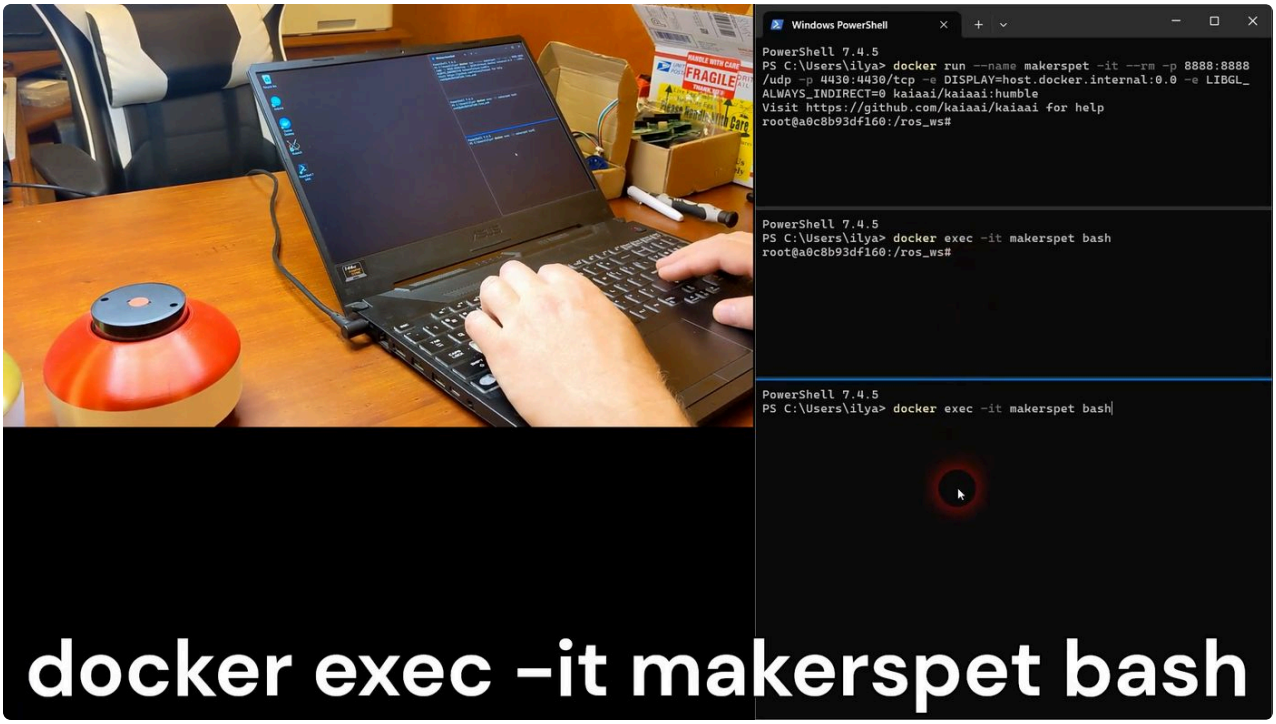


Start the Docker image with the maps folder shared.

6. In the **middle** and **bottom** windows, open shells in the same container:

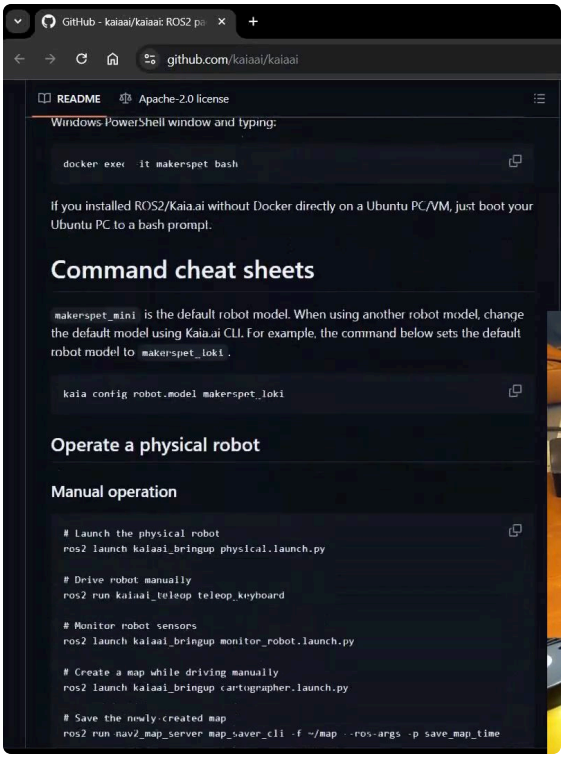
```
docker exec -it makerspet bash
```

Each shows a `root@...:/ros_ws#` prompt.



Open two more shells with docker exec.

7. Click `https://github.com/kaiaai/kaiaai` in the startup message and scroll to **Command cheat sheets** › **Operate a physical robot** to copy and paste commands.

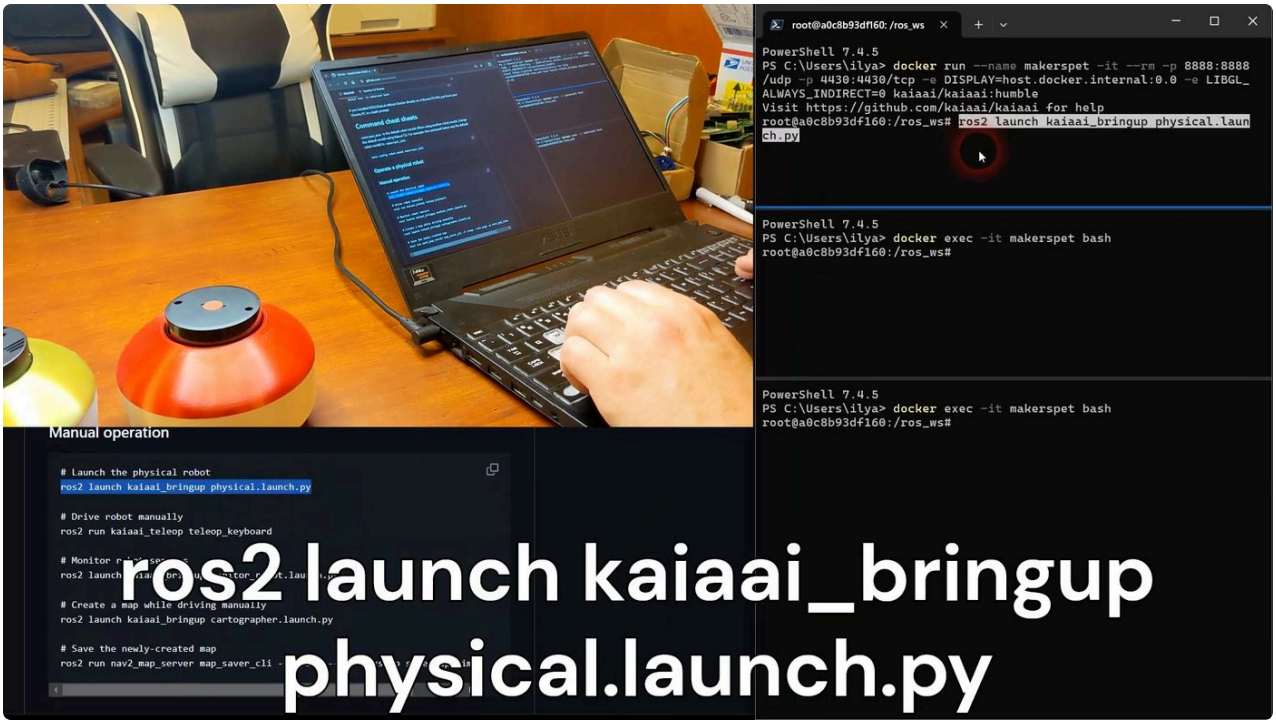


The cheat sheet for a physical robot.

Bring up the robot

1. In the **top** window, start ROS2 and the micro-ROS agent the robot connects to:

```
ros2 launch kaiaai_bringup physical.launch.py
```

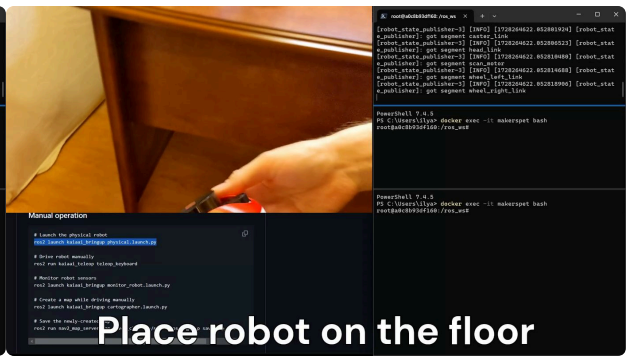


Launch ROS2 and micro-ROS in the top window.

2. Turn the robot on and place it on the floor.

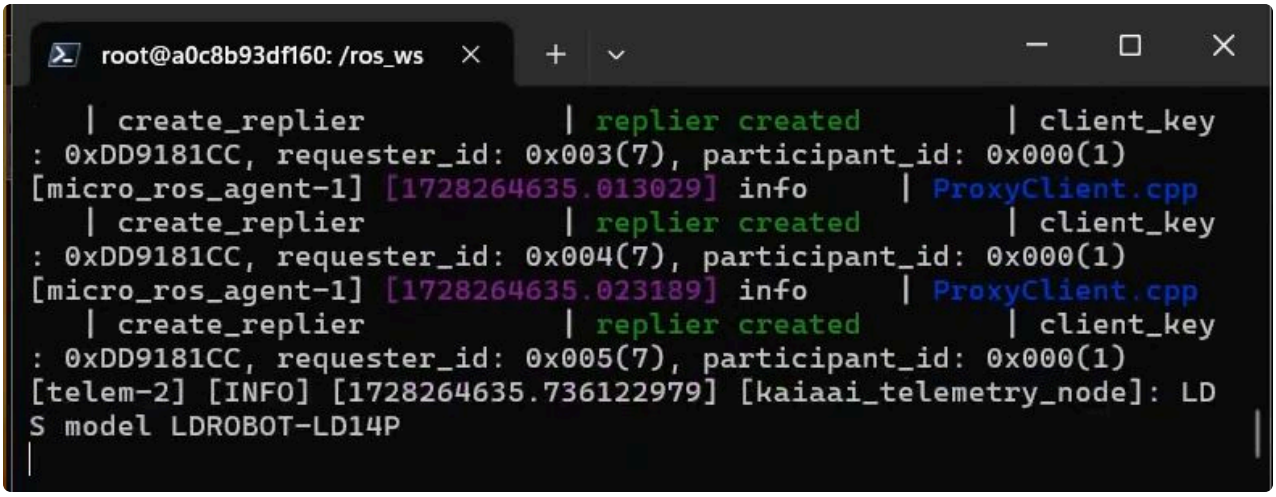


Turn the power switch on.



Place the robot on the floor.

3. Once the robot connects, the top window prints its messages, including the LiDAR model.



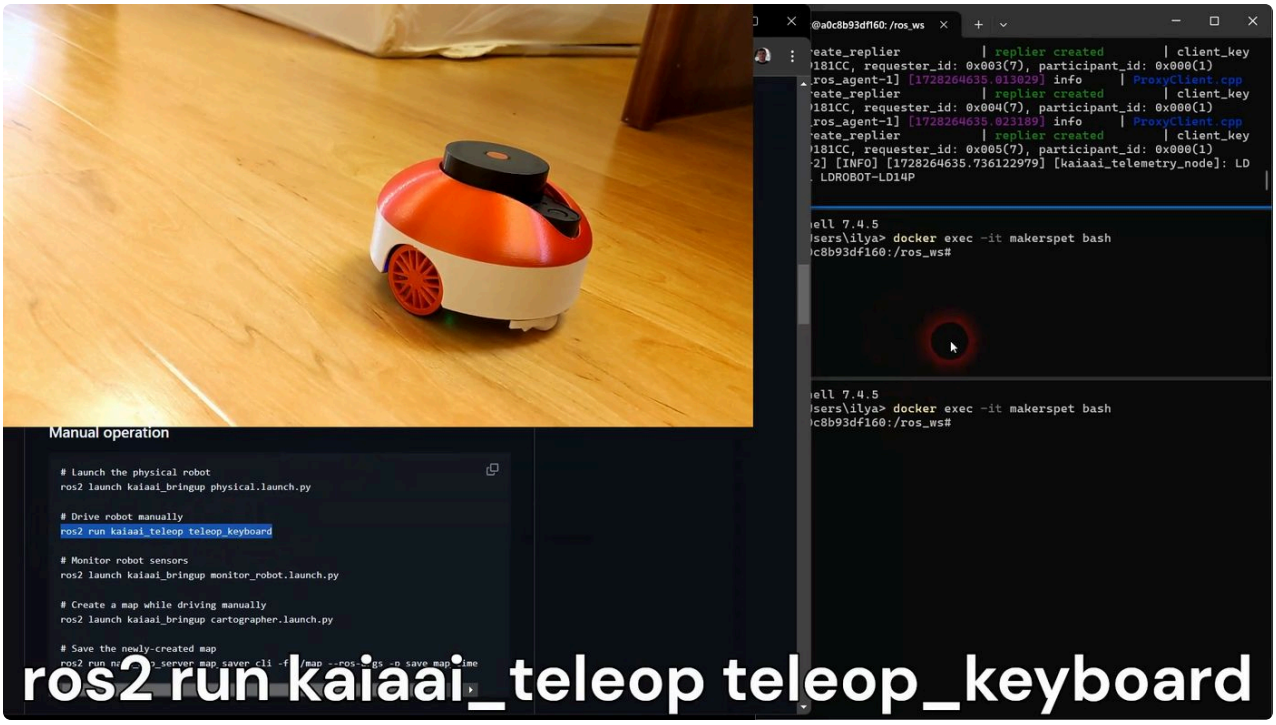
The robot has connected.

> **NOTE**
> Occasional `message(s) lost` and `RESULT_CRC_ERROR` lines are normal over WiFi. To keep them rare, give the robot a strong WiFi signal and keep the network quiet while the robot runs: for example, don't stream video on another PC or phone on the same WiFi.

Start teleoperation and mapping

1. In the **bottom** window, launch keyboard teleoperation:

```
ros2 run kaiaai_teleop teleop_keyboard
```



Launch teleoperation in the bottom window.

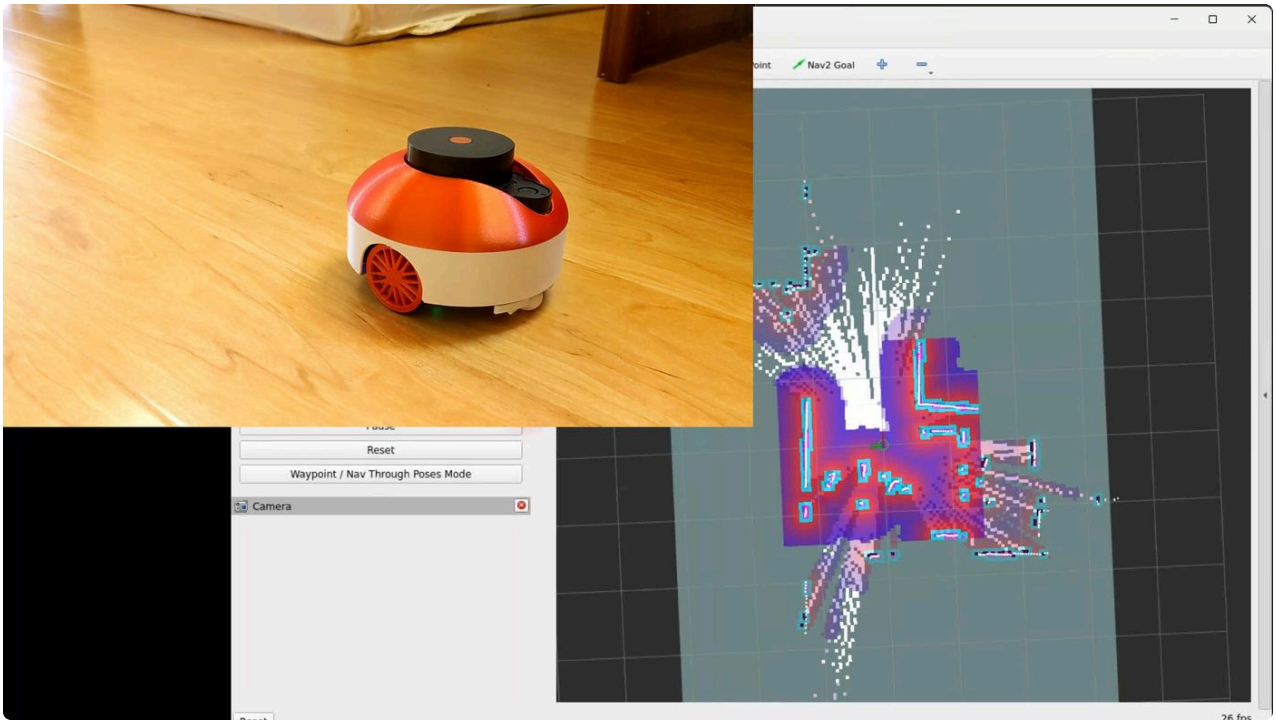
2. In the **middle** window, launch Nav2 with SLAM (simultaneous localization and mapping) and RViz:

```
ros2 launch kaiaai_bringup navigation.launch.py slam:=True
```



Launch mapping in the middle window.

3. RViz opens after a few seconds and shows the map starting to form.



The map starts to form around the robot.

> **NOTE**
> The command reference also lists `ros2 launch kaiaai_bringup cartographer.launch.py` under "Create a map while driving manually". The video uses `navigation.launch.py slam:=True`.

Drive around and map your place

1. Click in the **bottom** window so it receives your key presses.
2. Drive the robot:

Key	Action
`w` / `x`	increase / decrease linear (forward) velocity
`a` / `d`	increase / decrease angular (turning) velocity
`s`	keep straight
`CAPS`	large step
`Space`	force stop
`CTRL-C`	quit

Keys change speed in steps; the robot keeps moving until you change it or press `Space`.

```
PowerShell 7.4.5
PS C:\Users\ilya> docker exec -it makerspet bash
root@a0c8b93df160:/ros_ws# ros2 run kaiaai_teleop teleop_keyboard
YAML file name : /ros_ws/install/makerspet_mini/share/makerspet_min
i/config/teleop_keyboard.yaml
Max linear velocity 0.200        Max angular velocity 3.860
Control Kaia.ai-compatible Robot
---------------------------------
Moving around:
      w
 a    s    d
      x
w/x   : increase/decrease linear  velocity
a/d   : increase/decrease angular velocity
s     : keep straight
CAPS  : large step
Space : force stop
CTRL-C to quit
```

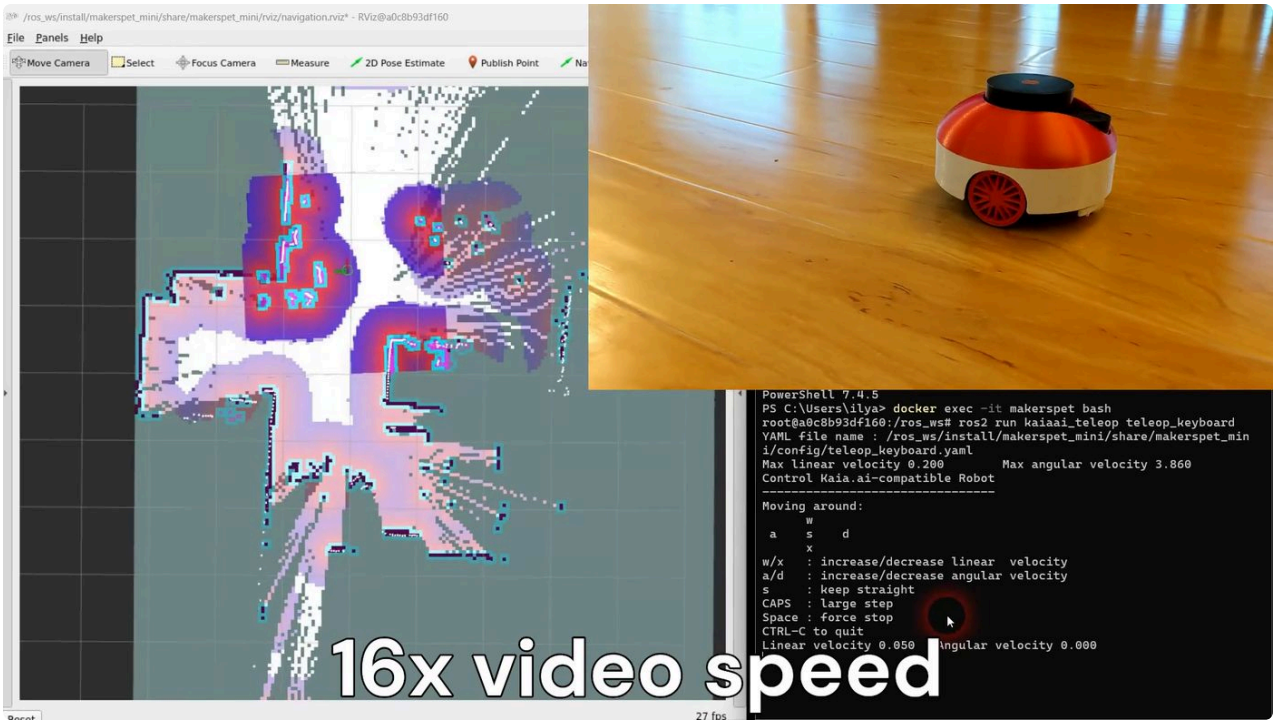
The teleoperation keys.

3. Drive as slowly as possible. The video uses 0.05 m/s and 0.2 rad/s, one or two key presses from a standstill.



Drive slowly for a better map.

4. Visit every room, following walls and circling furniture, until the map has no gaps.



The map grows as the robot explores (16× speed).

SINCE THE VIDEO WAS RECORDED

Map skewed or distorted? Drive and turn slowly; fast motion skews SLAM badly. Check the sensors with `ros2 launch kaiaai_bringup monitor_robot.launch.py`, and start with a simple room (one central obstacle).

Map distorts or the robot drifts off it? The wheel speed probably doesn't match the command, usually due to a wrong encoder PPR (pulses per revolution) in `config.yaml`. Read it with `ros2 param get /pet motor.encoder.ppr`. To measure it, count encoder ticks over one wheel revolution and divide by 4 (the firmware multiplies PPR by 4 for quadrature edges). Set the correct PPR and a reachable Max RPM, then re-upload the sketch data.

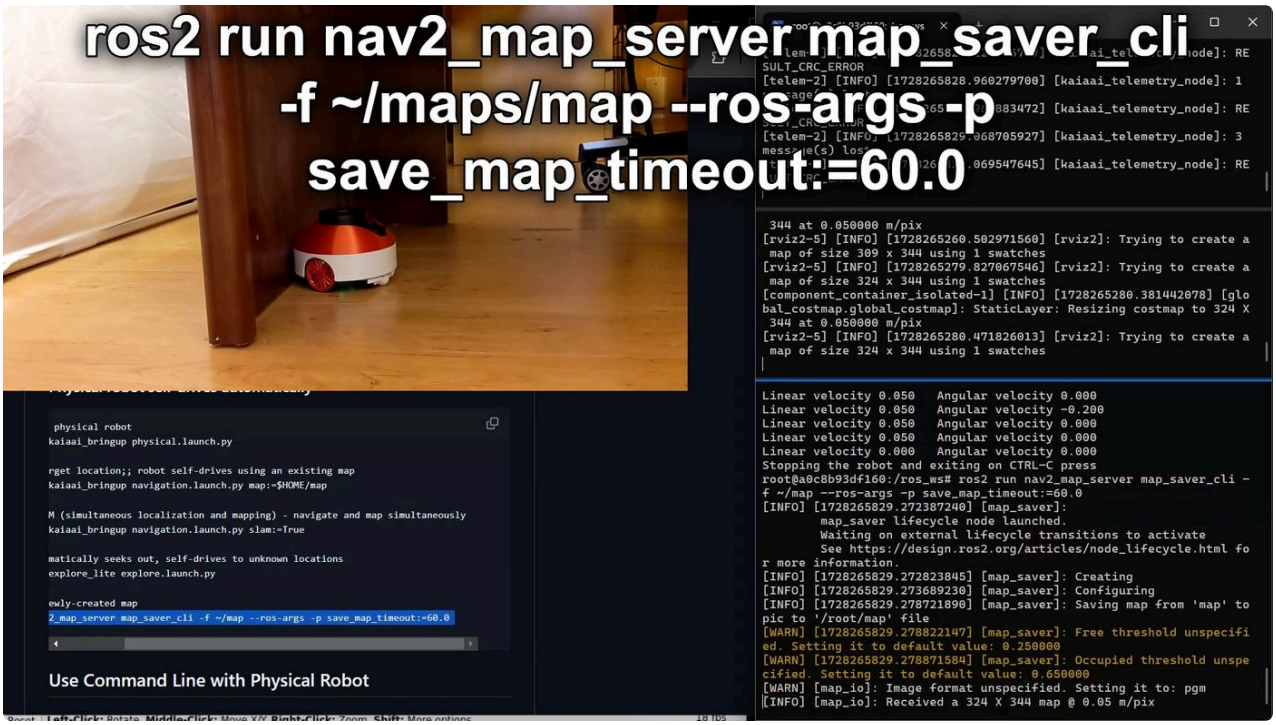
Scan slides sideways when driving straight? If rotation looks right and odometry is calibrated, the LiDAR is mounted rotated relative to its URDF frame. Add the yaw offset to the laser joint's rpy; for a LiDAR mounted 180° around, add 3.14 rad.

Save your map and shut down

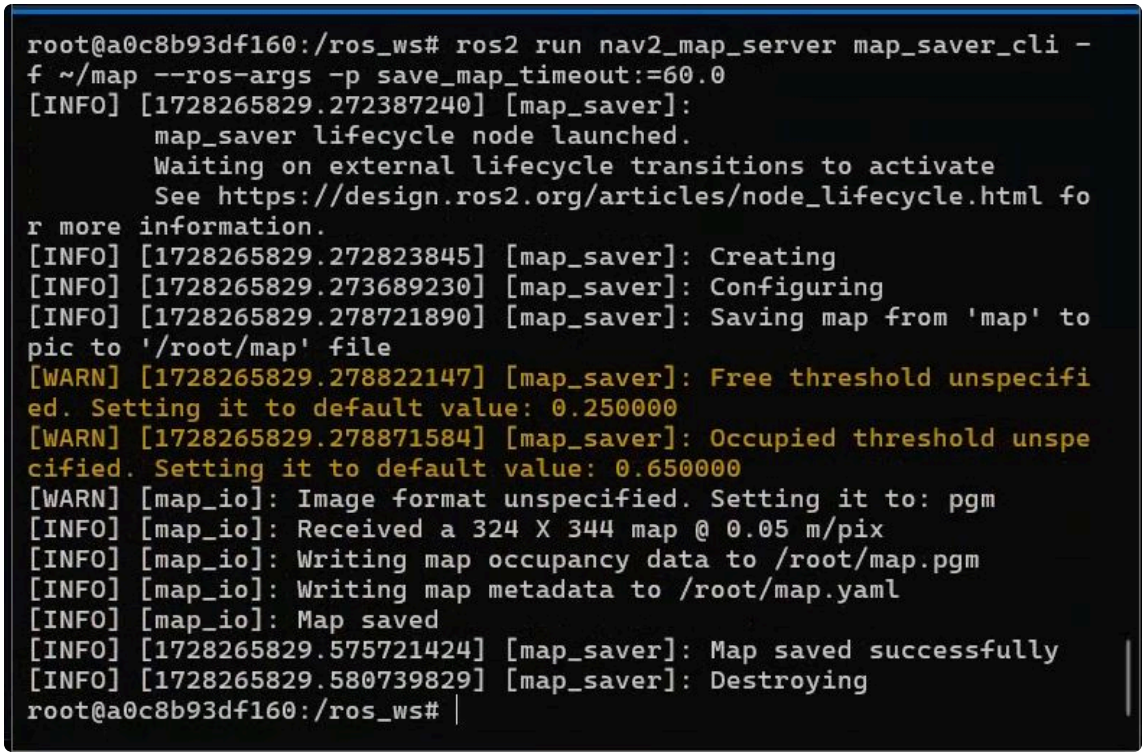
1. Click in the **bottom** window and press **Ctrl+C** to stop teleoperation.
2. In the same window, save the map:

```
ros2 run nav2_map_server map_saver_cli -f ~/maps/map --ros-args -p save_map_timeout:=60.0
```

This writes `map.pgm` (the image) and `map.yaml` (resolution and origin) to `C:\maps`. Wait for `Map saved successfully`.

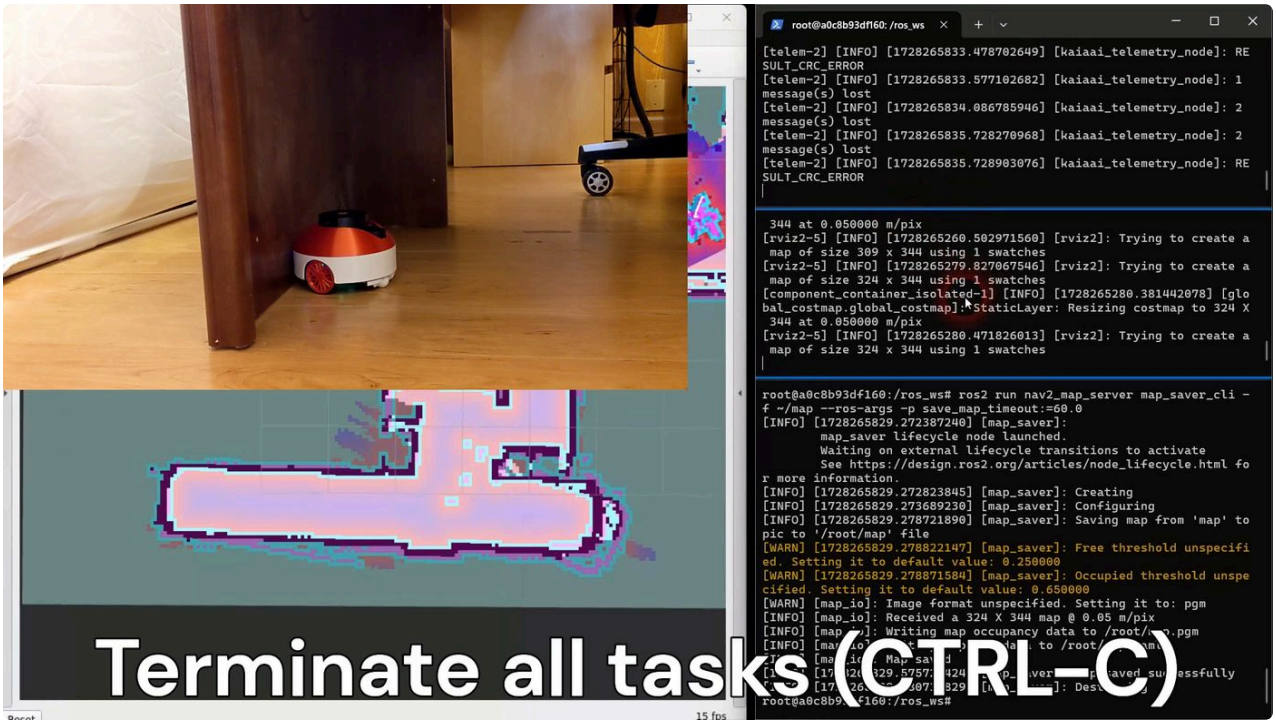


Save the map.



Map saved successfully.

3. Press **Ctrl+C** in the middle and top windows and wait for the processes to finish.



Terminate all tasks with Ctrl+C.

4. Type `exit` in each shell and close PowerShell.
5. Turn off the robot.

SINCE THE VIDEO WAS RECORDED

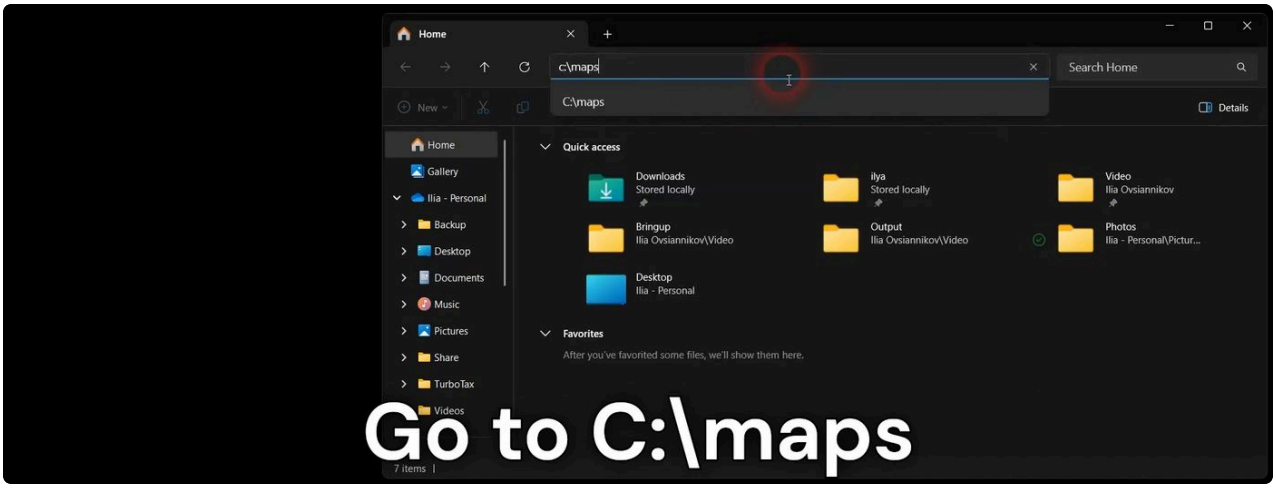
`--rm` discards changes made inside the container (your map is safe in `C:\maps`). To keep other changes, such as edited config files, commit the running container from another PowerShell window *before* exiting it:

```
docker container commit makerspet kaiaai/kaiaai:iron
```

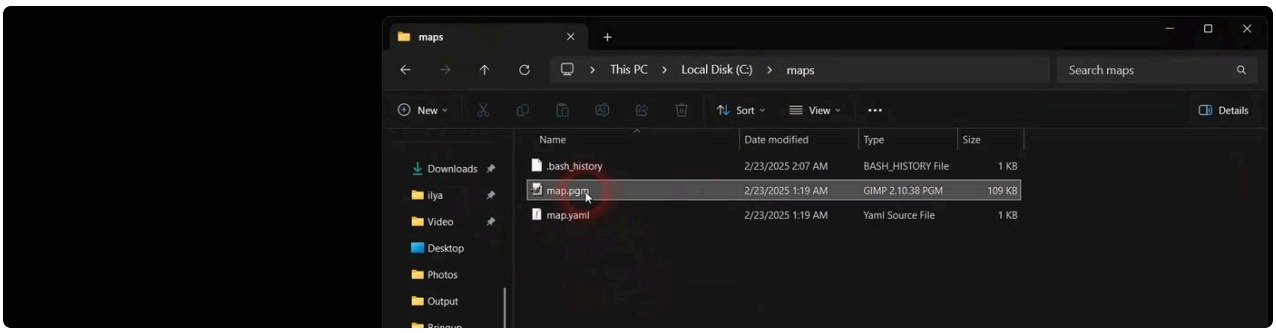
Don't commit or publish an image that contains secrets such as your WiFi credentials.

View your map

1. Open File Explorer and type `C:\maps` into the address bar.

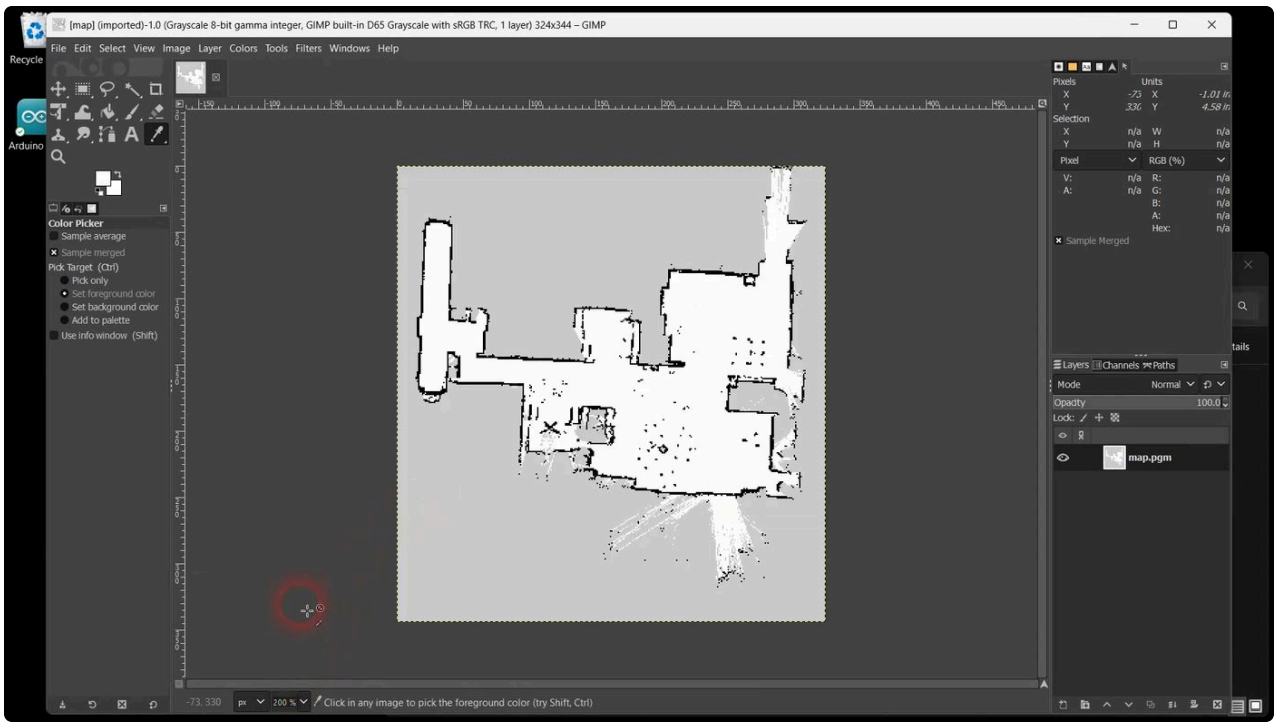


Go to C:\maps in File Explorer.



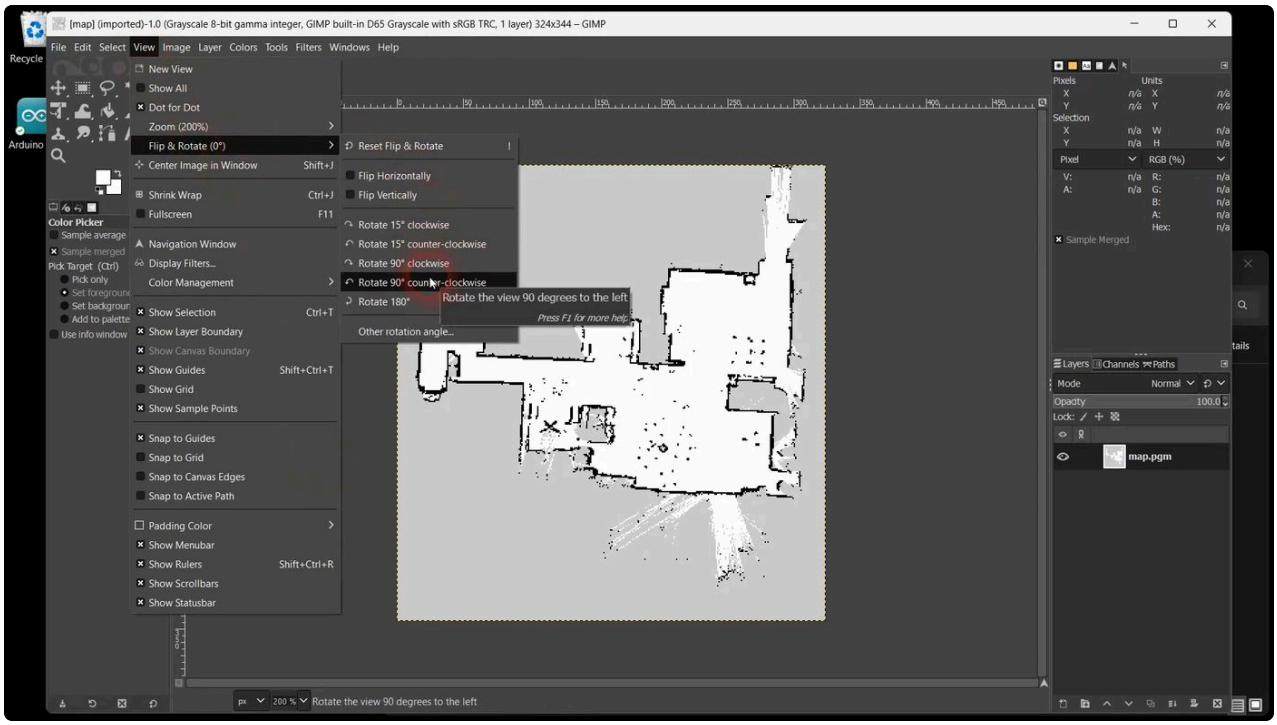
map.pgm and map.yaml.

2. Open `map.pgm` in a PGM-capable viewer: GIMP for Windows (https://www.gimp.org/downloads/), as in the video, or the lighter XnView. White is free space, black is obstacles, gray is unexplored.



The map in GIMP.

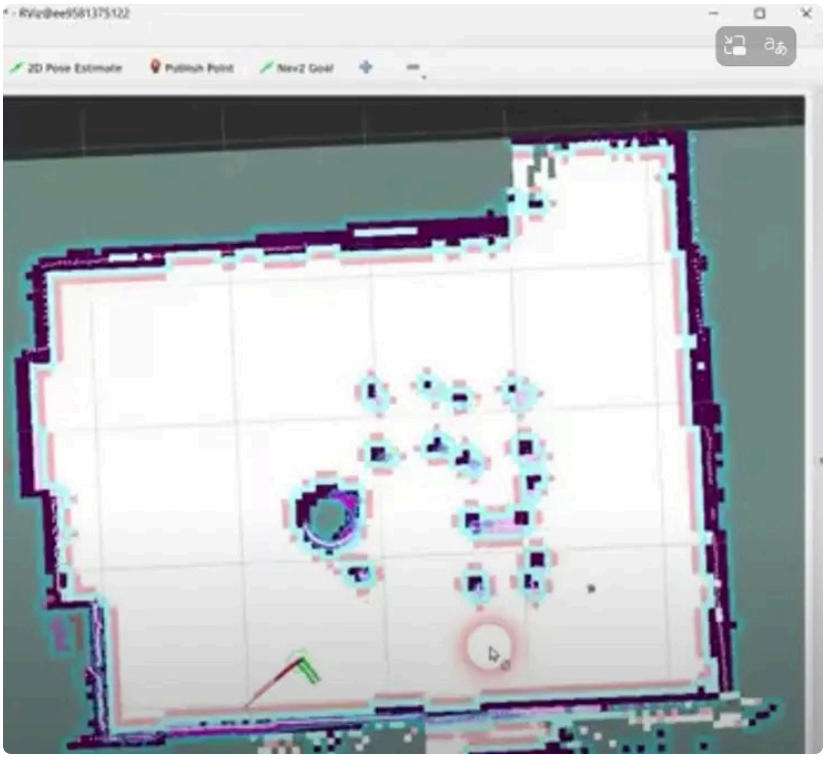
3. If the map appears rotated, use **View › Flip & Rotate › Rotate 90° counter-clockwise** (or whichever fits) in GIMP. This rotates the view, not the file.



Rotate the view in GIMP.

> **TIP**
> Don't edit and re-save `map.pgm`. The robot uses it, with `map.yaml`, in the next chapter.

A clean map has straight walls, closed outlines and few stray specks. If yours is much messier, declutter, drive slower and map again.



A clean map.

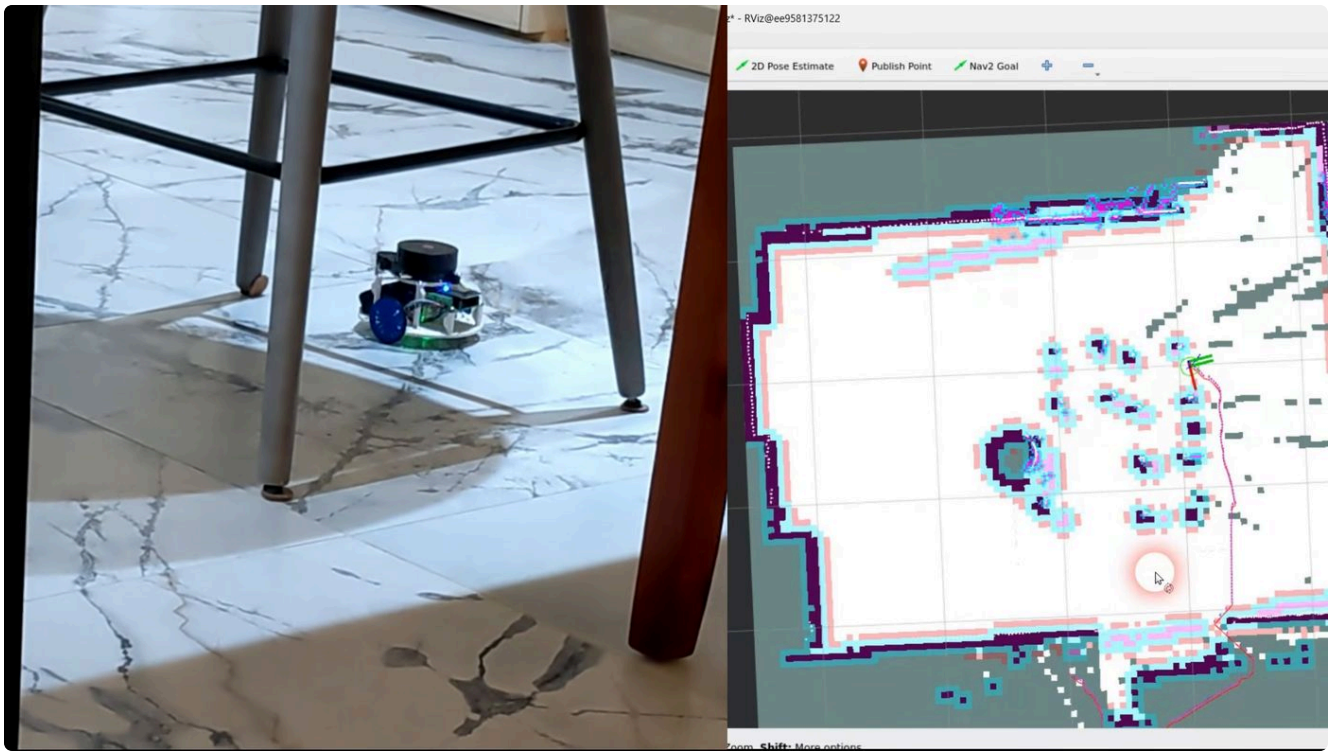
Next: let the robot drive itself using this map.

5. Let the Robot Explore and Self-Drive

Video: Arduino/ROS2 robot self-drives - operation instructions

Now your robot drives itself. First it explores your place and builds a map. Then you save the map and click any spot in RViz; the robot plans a path, drives there and steers around obstacles.

Your robot should be assembled, brought up and connected to WiFi and your PC. Chapter 4 helps: it uses the same Docker image, terminals and RViz. Allow about 30 minutes, mostly watching the robot explore.



The robot self-drives under a chair; RViz shows the map and planned path.

Prepare your place

The LiDAR sees obstacles only in one horizontal plane, at sensor height; anything lower is invisible. The robot also gets stuck on thick carpet and high thresholds.

1. Remove low objects, or block them with cardboard so the LiDAR can see them.
2. Remove or block off thick carpet and high thresholds.



Block low obstacles with cardboard, or remove them.



Remove thick carpet.

SINCE THE VIDEO WAS RECORDED

Also prepare the following:

- Block (for example, with cardboard) objects the LiDAR cannot detect: black (non-reflective), transparent, too low for the laser plane, too small (such as fine sparse mesh), mirrors and highly reflective objects, and objects under bright near-infrared light (sunlight, incandescent bulbs).
- The robot cannot drive over thick carpet or thresholds higher than about 5 mm (1/5″).
- Keep the scene static while mapping; for example, don't open or close doors.
- Declutter and leave plenty of space between obstacles.
- Block off angled ramps: they tilt the LiDAR's laser plane and mess up the map.

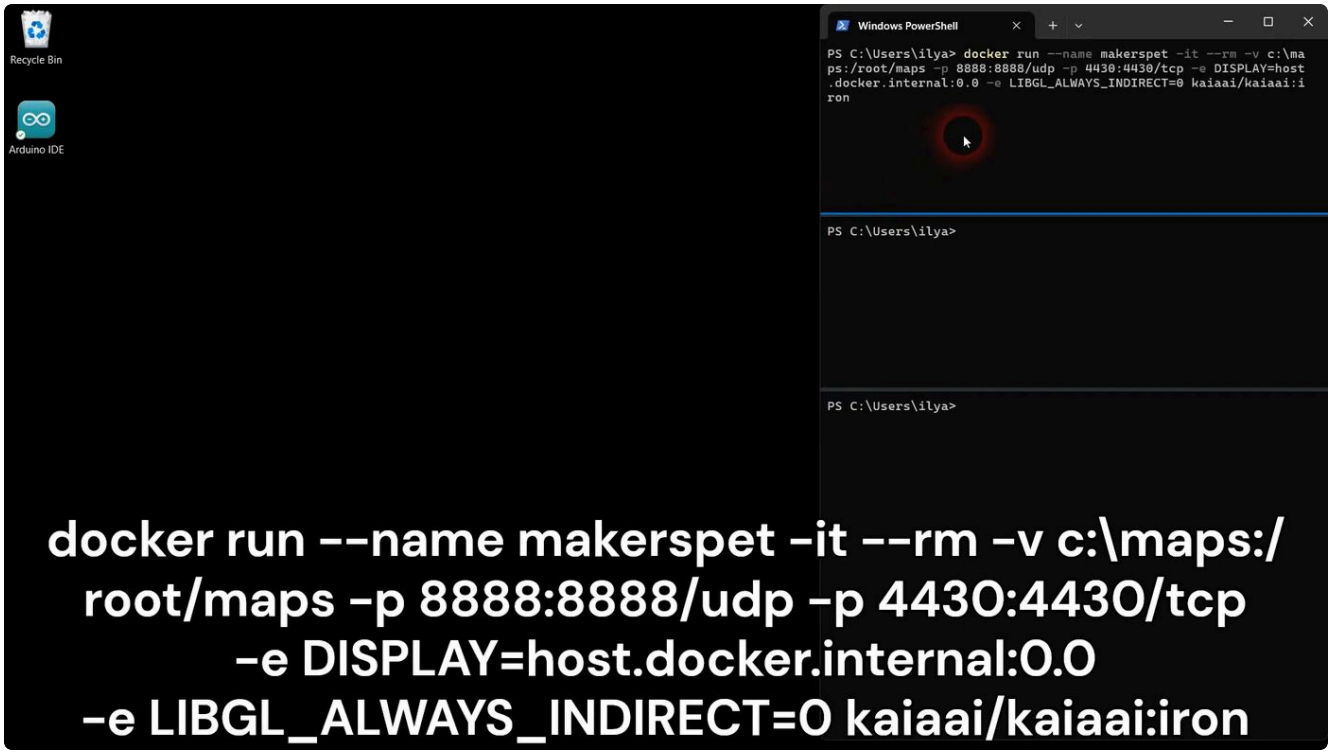
Launch the robotics software

You will use three command windows: one for the Docker container and the robot link, one for mapping and navigation, and one for exploration and saving the map.

1. Open Windows PowerShell, click in its window and press **Alt+Shift+-** to split it horizontally. Repeat until you have three panes: top, middle and bottom.
2. In the top window, launch the Docker image:

```
docker run --name makerspet -it --rm -v c:\maps:/root/maps -p 8888:8888/udp -p 4430:4430/t
cp -e DISPLAY=host.docker.internal:0.0 -e LIBGL_ALWAYS_INDIRECT=0 kaiaai/kaiaai:iron
```

`-v c:\maps:/root/maps` shares `c:\maps` on your PC with the container, so your saved map survives when the container stops.



Launch the Docker image in the top window.

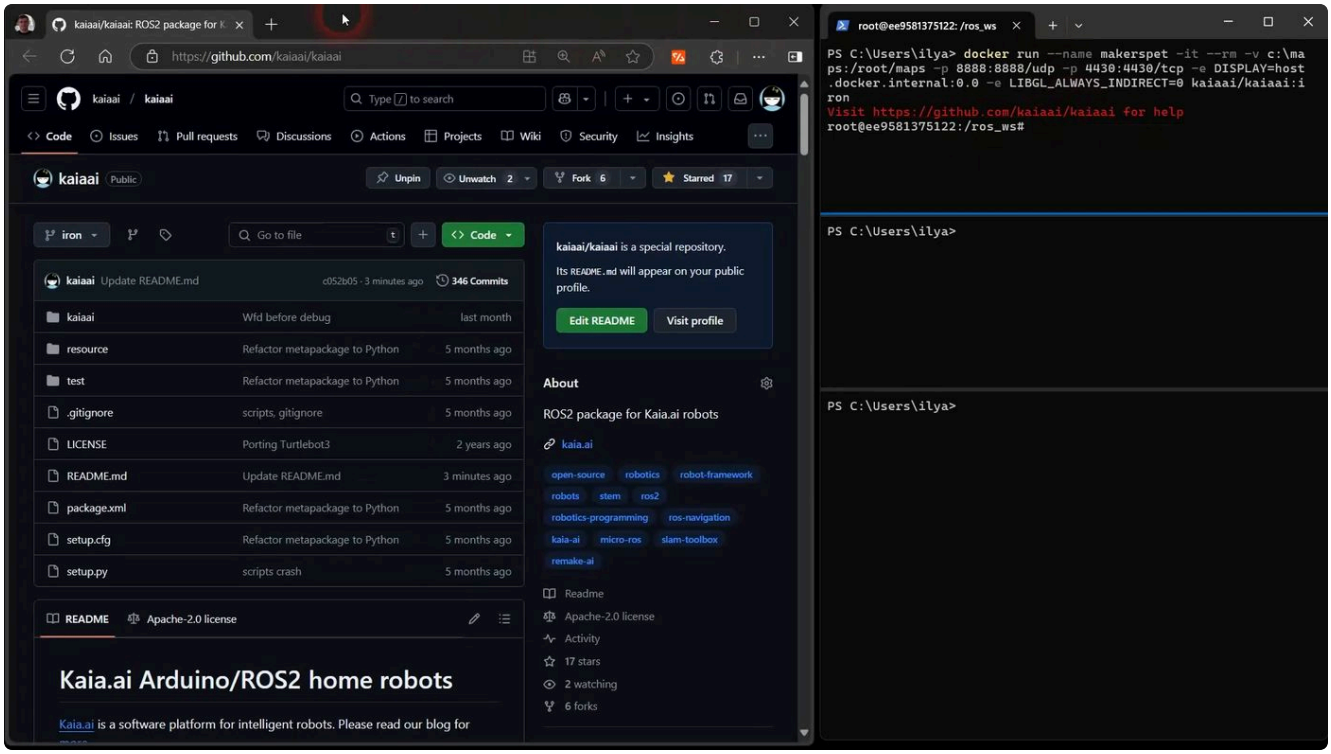
TIP

Start the X server first: launch **XLaunch** from your desktop with display number `0` (Chapter 2). Without it, RViz can't open its window.

SINCE THE VIDEO WAS RECORDED

A ROS 2 Jazzy image, `kaiaai/kaiaai:jazzy`, is also available next to `kaiaai/kaiaai:iron`. To use it, change the tag at the end of the command.

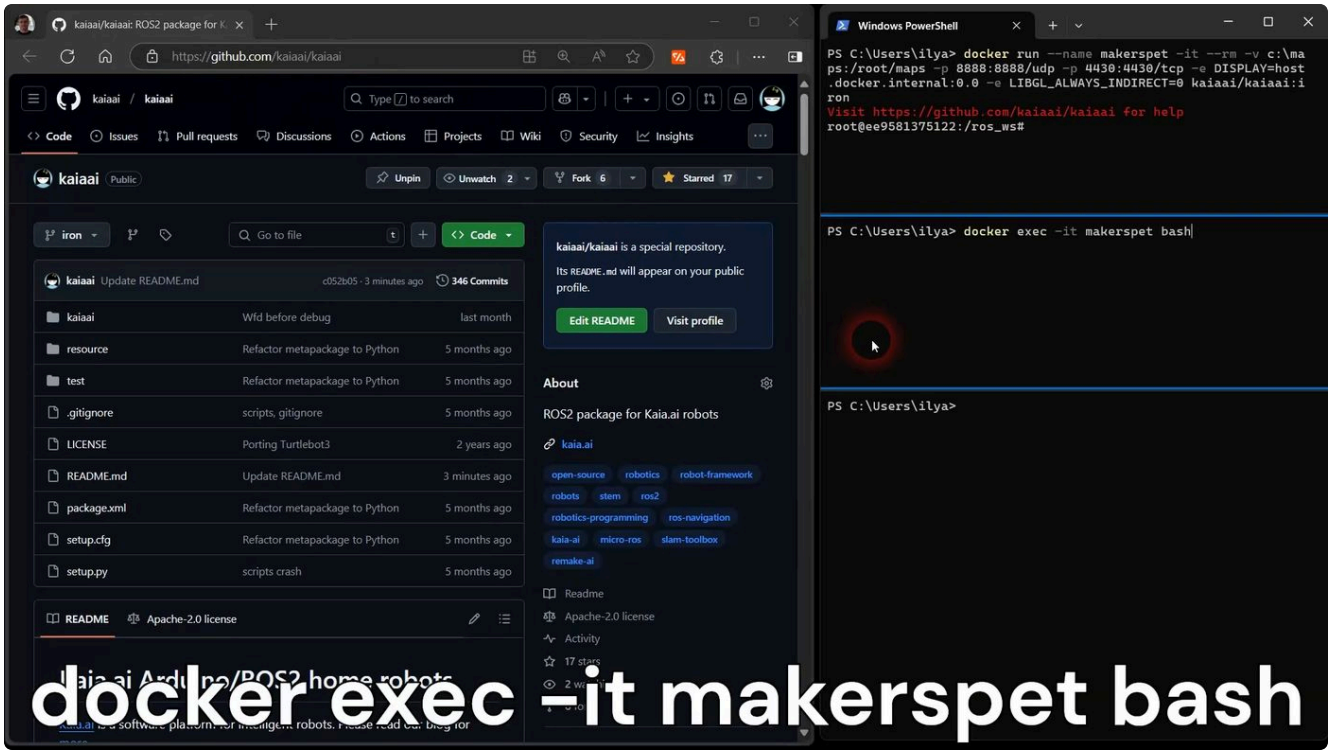
3. Ctrl-click the help link `https://github.com/kaiaai/kaiaai` that the container prints. Every command in this chapter is under **Command cheat sheets** there, ready to copy and paste.



Ctrl-click the help link to open the command reference.

4. In each of the other two windows, open a Bash shell inside the container:

```
docker exec -it makerspet bash
```

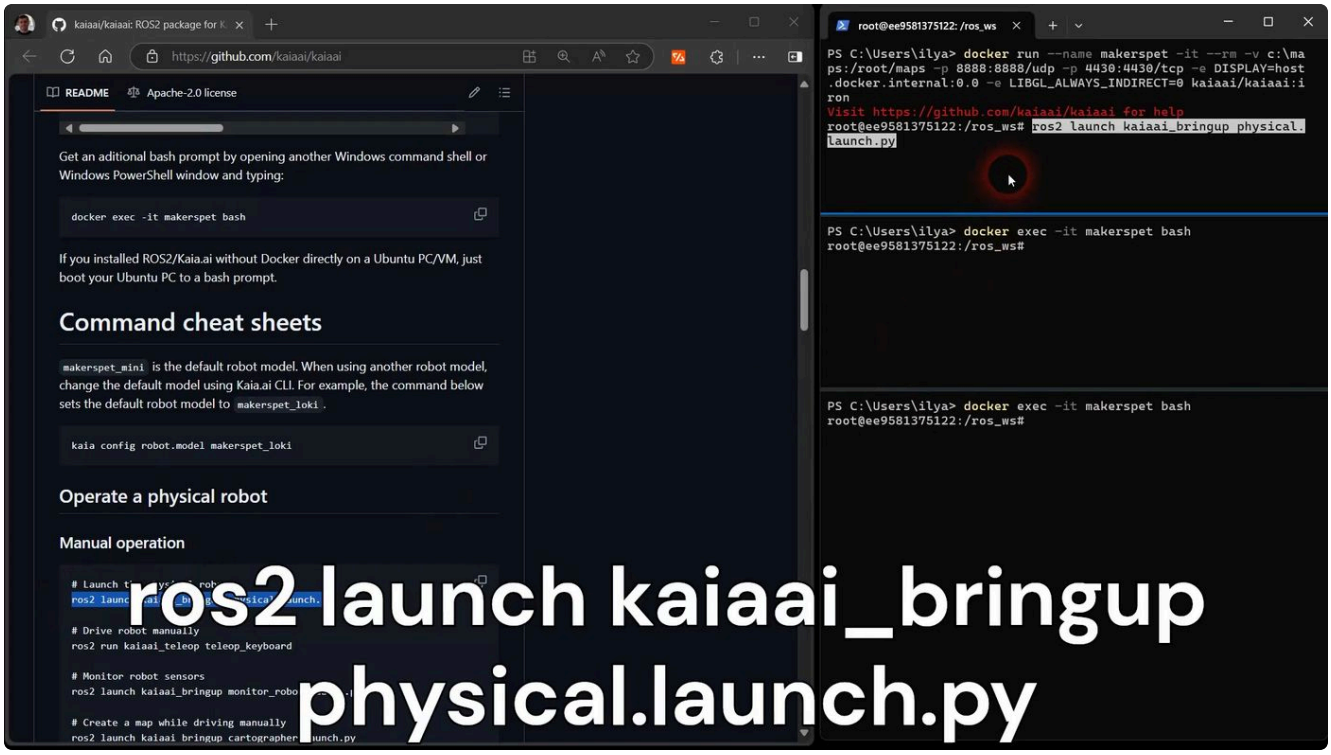


Open a Bash shell in the middle and bottom windows.

Connect to your robot

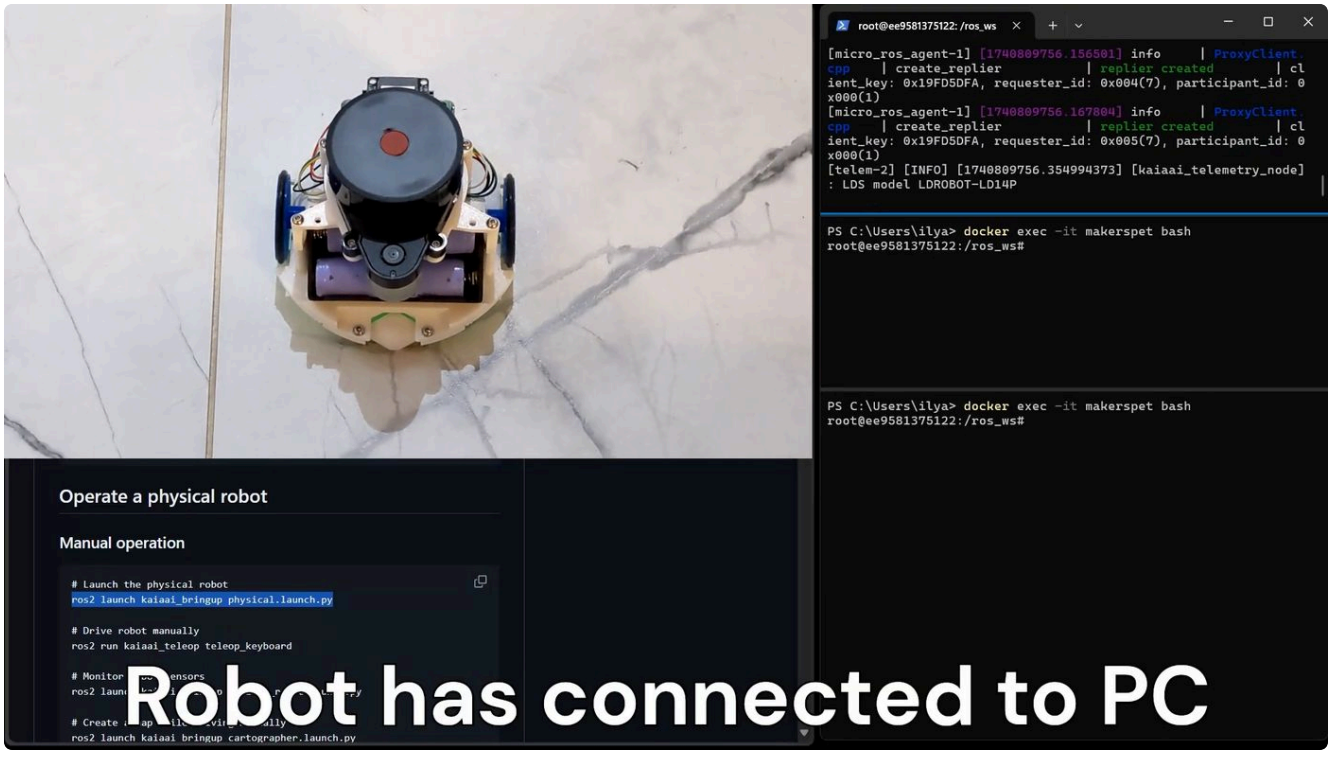
1. In the top window, launch communication with your robot:

```
ros2 launch kaiaai_bringup physical.launch.py
```



Launch robot communication in the top window.

2. Turn your robot on and place it on the floor. Once it connects to WiFi and your PC, the top window prints its messages.

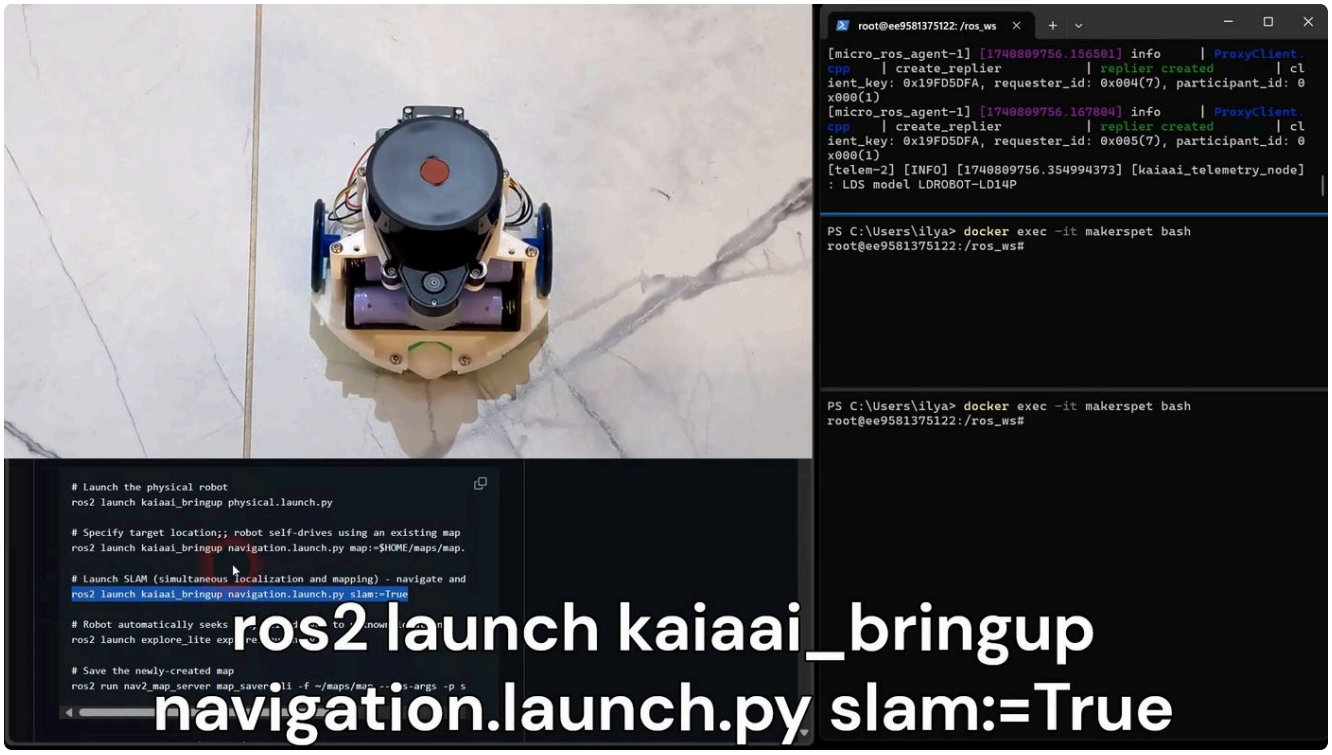


The robot is connected: the top window shows its messages.

Let the robot explore and create a map

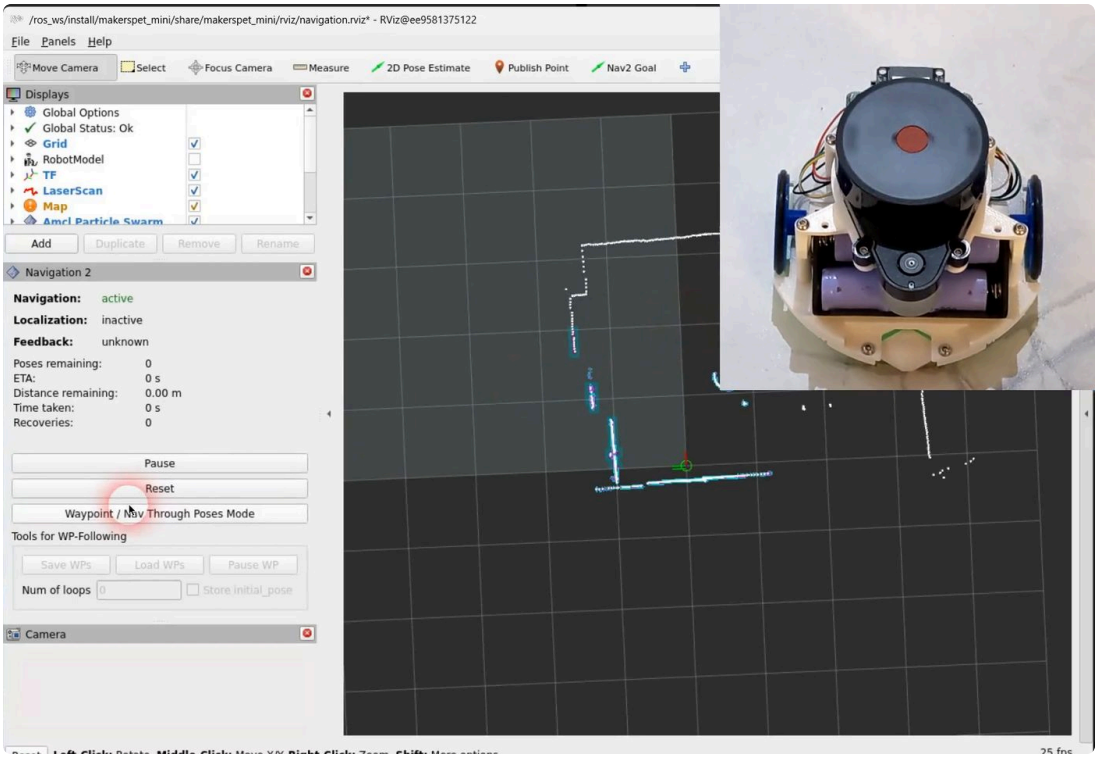
1. In the middle window, launch mapping, navigation and visualization. `slam:=True` builds a new map while navigating (SLAM: simultaneous localization and mapping):

```
ros2 launch kaiaai_bringup navigation.launch.py slam:=True
```



Launch mapping and navigation in the middle window.

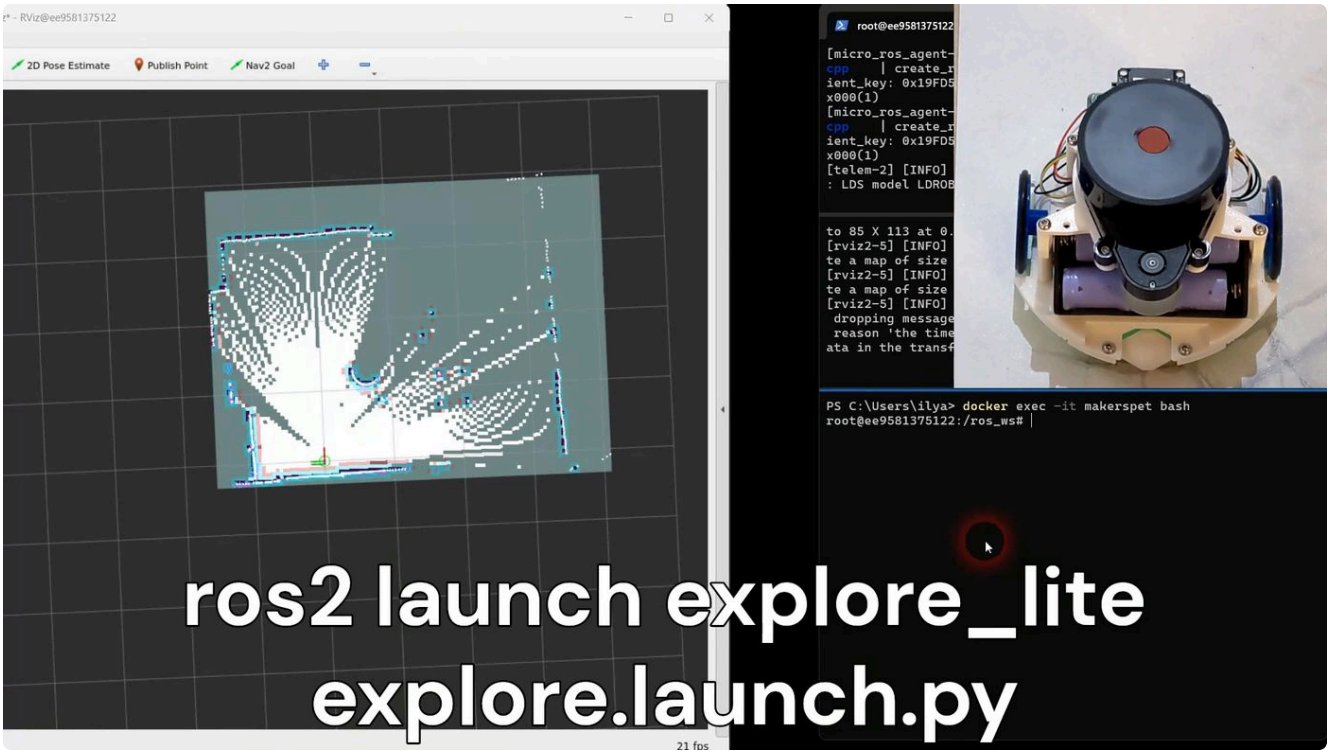
2. RViz opens with the start of the map and the **Navigation 2** panel on the left.



RViz with the start of the map and the Navigation 2 panel.

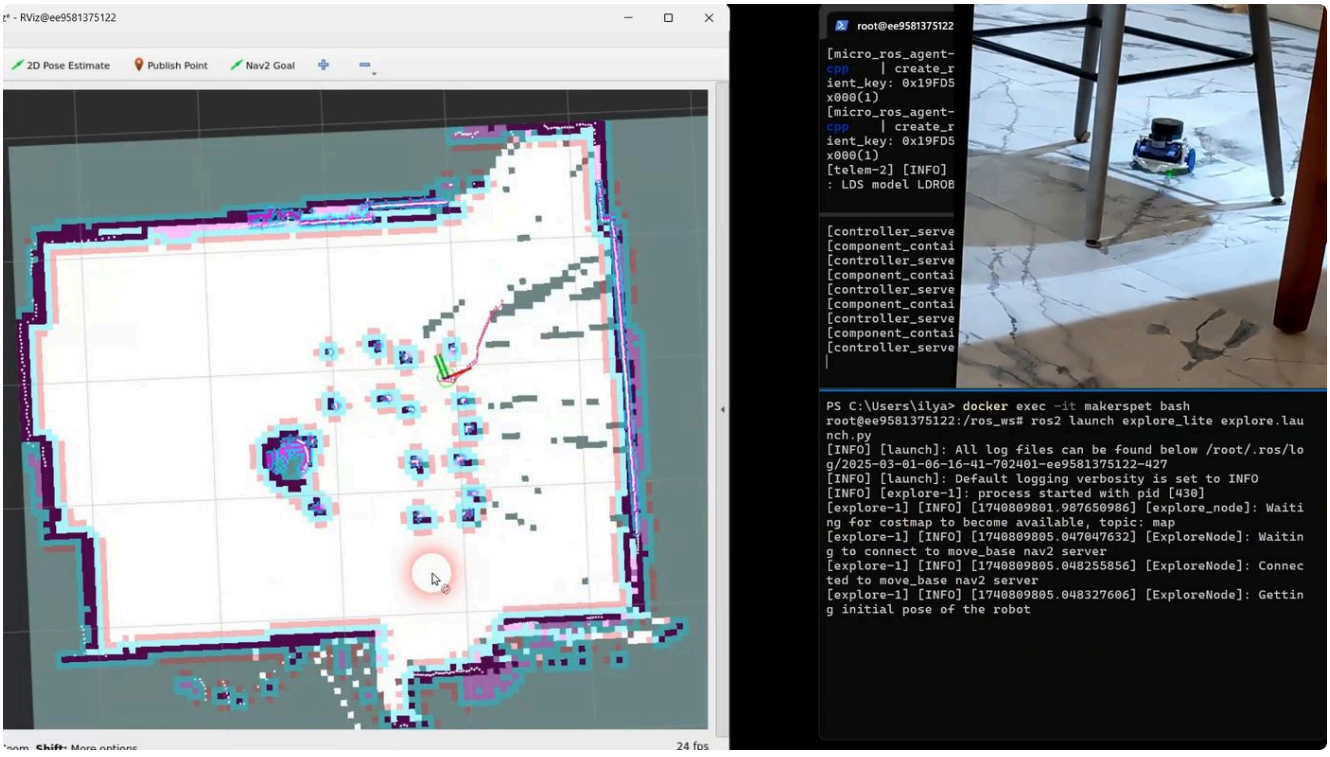
3. In the bottom window, launch exploration:

```
ros2 launch explore_lite explore.launch.py
```



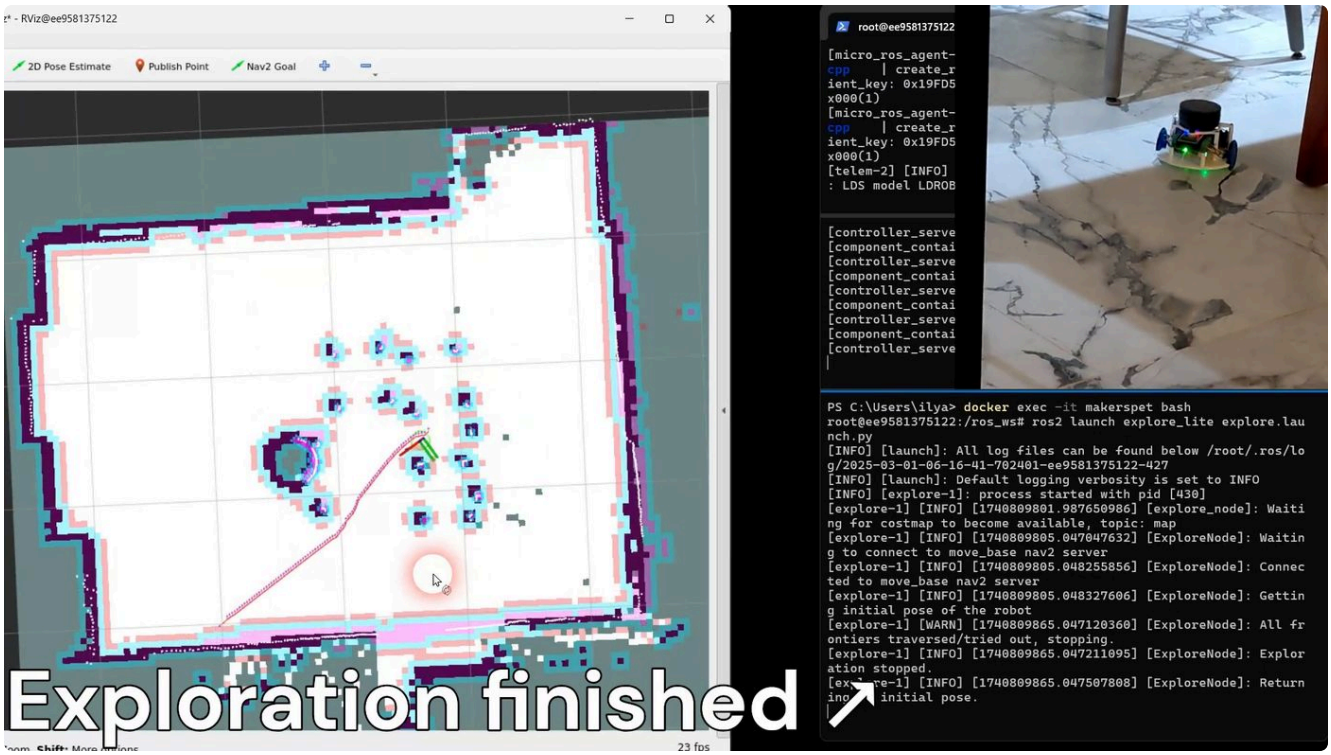
Launch exploration in the bottom window.

4. Watch your robot explore. The software keeps sending it to *frontiers*, the edges between mapped and unknown areas, until nothing is left to explore.



The map grows as the robot explores.

5. When no frontiers are left, the robot returns to its start and the bottom window reports `All frontiers traversed/tried out, stopping.`



Exploration finished: the map is complete.

Save the map

1. Press Ctrl-C in the bottom window to stop exploration.
2. In the same window, save the map:

```
ros2 run nav2_map_server map_saver_cli -f ~/maps/map --ros-args -p save_map_timeout:=60.0
```

3. Look for `Map saved successfully`. The map is two files, `/root/maps/map.pgm` (the image) and `/root/maps/map.yaml` (its settings), also in `c:\maps` on your PC.

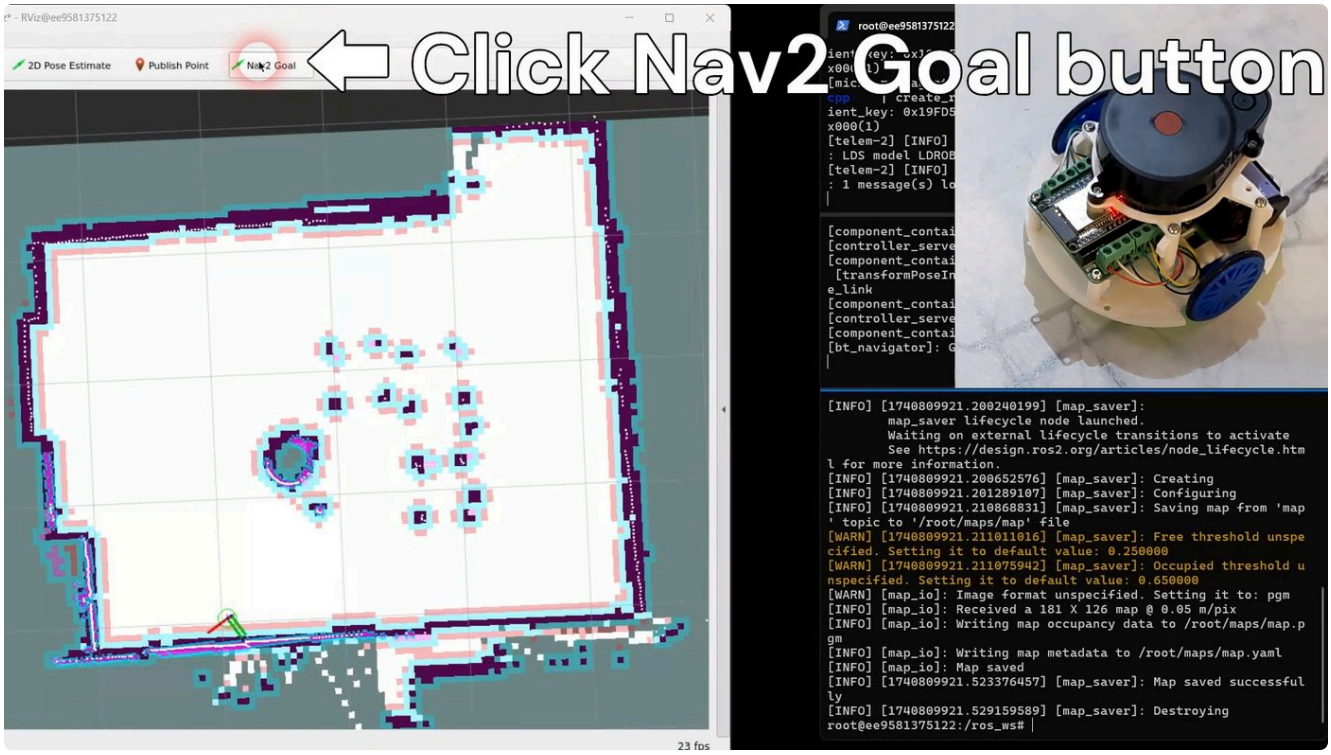
```
[INFO] [1740809921.200240199] [map_saver]:
        map_saver lifecycle node launched.
        Waiting on external lifecycle transitions to activate
        See https://design.ros2.org/articles/node_lifecycle.htm
l for more information.
[INFO] [1740809921.200652576] [map_saver]: Creating
[INFO] [1740809921.201289107] [map_saver]: Configuring
[INFO] [1740809921.210868831] [map_saver]: Saving map from 'map
' topic to '/root/maps/map' file
[WARN] [1740809921.211011016] [map_saver]: Free threshold unspe
cified. Setting it to default value: 0.250000
[WARN] [1740809921.211075942] [map_saver]: Occupied threshold u
nspecified. Setting it to default value: 0.650000
[WARN] [map_io]: Image format unspecified. Setting it to: pgm
[INFO] [map_io]: Received a 181 X 126 map @ 0.05 m/pix
[INFO] [map_io]: Writing map occupancy data to /root/maps/map.p
gm
[INFO] [map_io]: Writing map metadata to /root/maps/map.yaml
[INFO] [map_io]: Map saved
[INFO] [1740809921.523376457] [map_saver]: Map saved successful
ly
[INFO] [1740809921.529159589] [map_saver]: Destroying
root@ee9581375122:/ros_ws#
```

The map saver reports success.

Send the robot to a destination

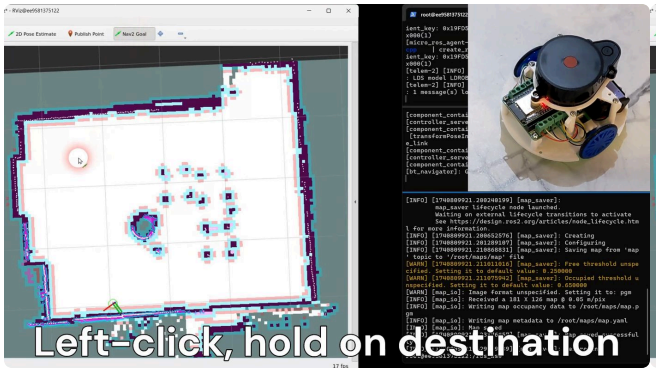
Navigation is still running with your new map, so you can send the robot places right away.

1. Click **Nav2 Goal** in the RViz toolbar.

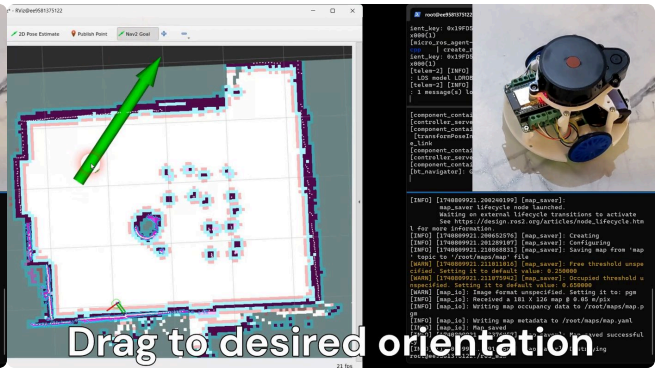


Click Nav2 Goal.

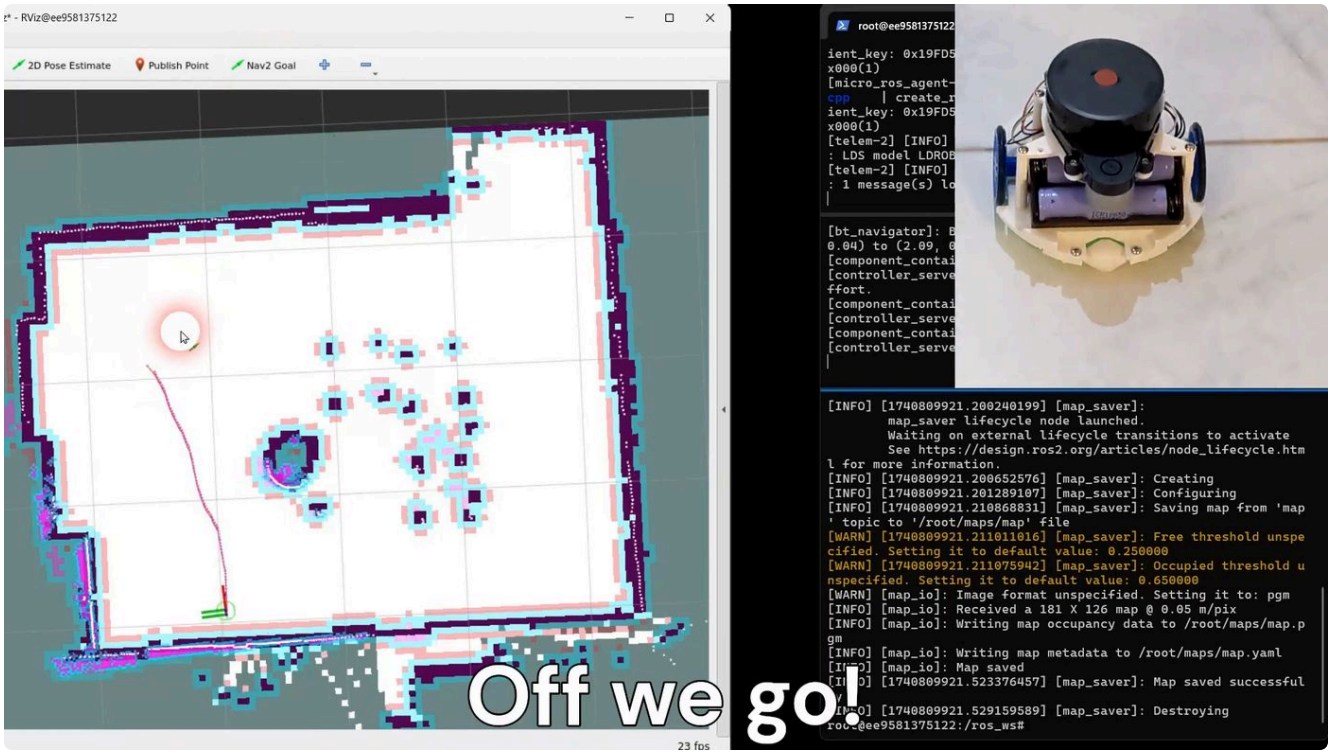
2. Click and hold on the map where you want the robot to go.
3. Drag to set the direction it should face on arrival (the green arrow).
4. Release and watch your robot self-drive.



Click and hold on the destination.



Drag to set the orientation, then release.



The robot follows the planned path (the thin line) to the goal.

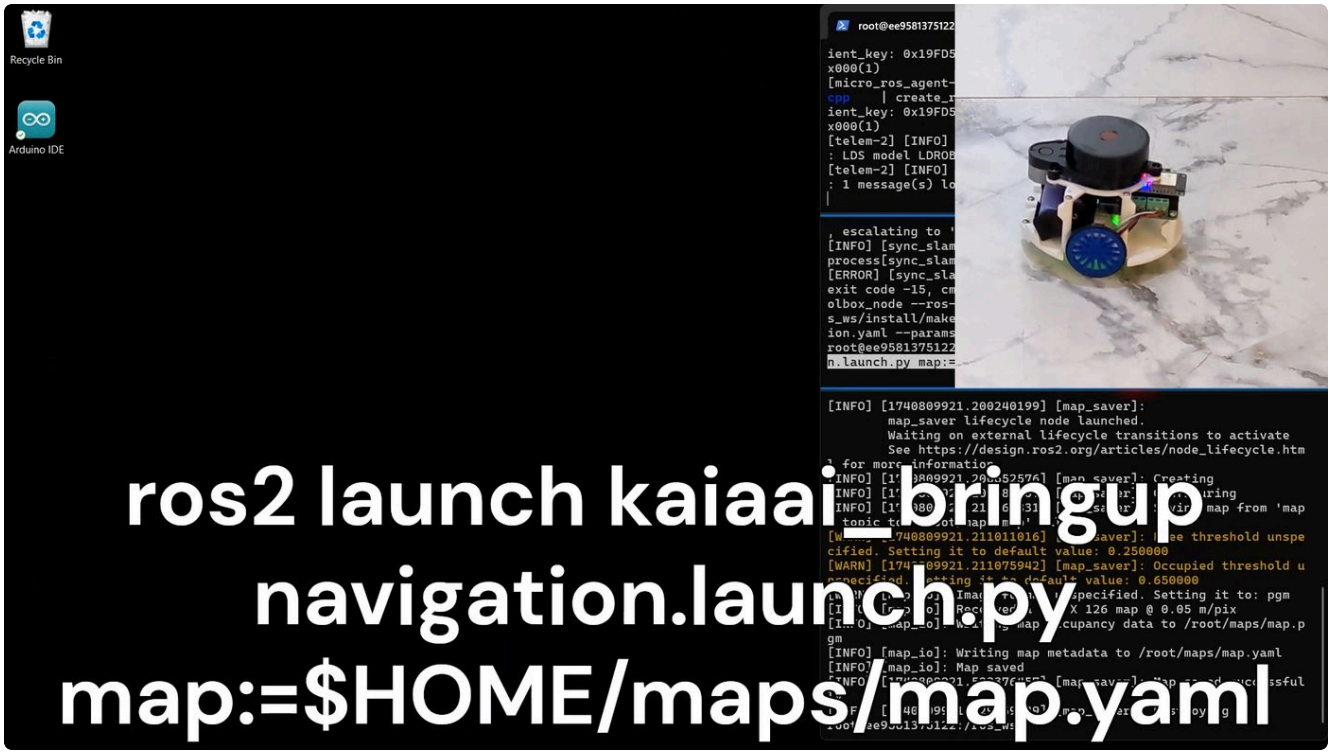
5. Try a few more destinations.

Self-drive using the saved map

Next time, skip exploring and load the saved map.

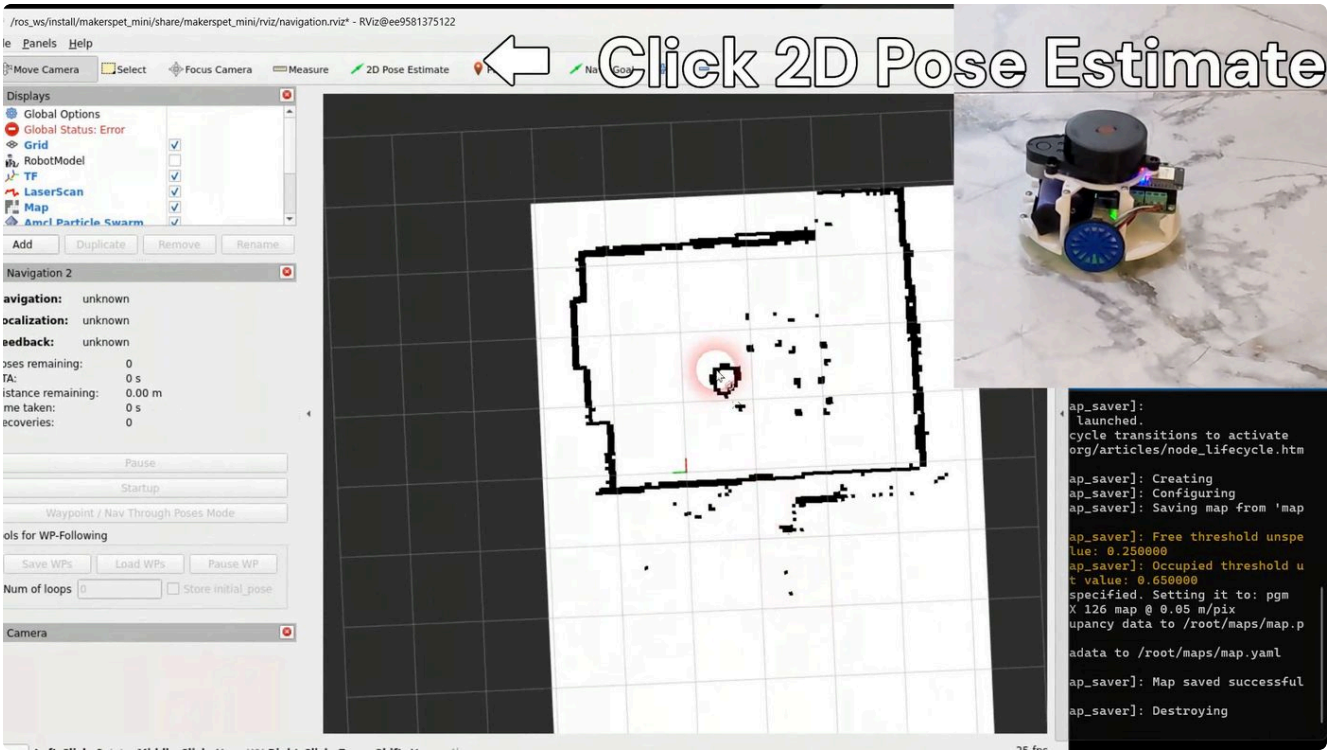
1. Press Ctrl-C in the middle window to stop mapping and navigation.
2. In the middle window, launch navigation with your saved map:

```
ros2 launch kaiaai_bringup navigation.launch.py map:=$HOME/maps/map.yaml
```



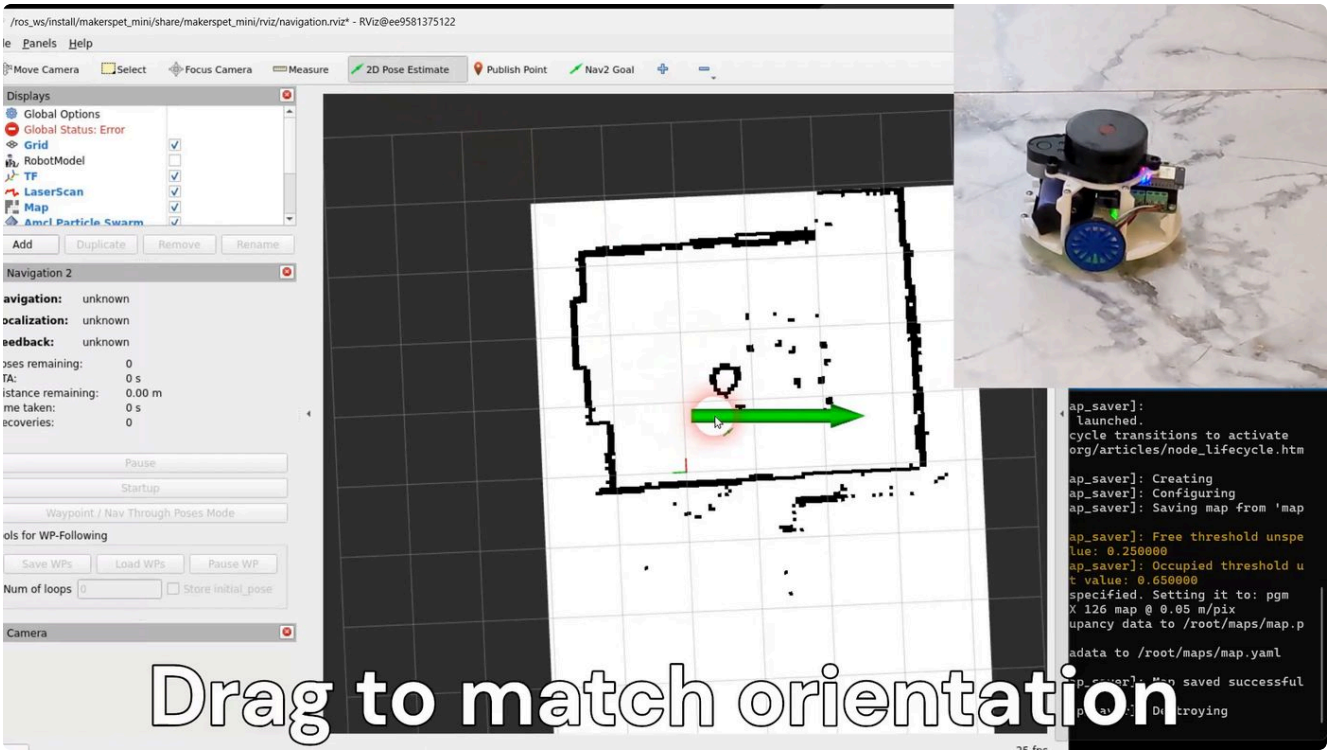
Launch navigation with the saved map.

3. RViz shows the saved map. The robot doesn't know where it is yet, so the **Navigation 2** panel shows `unknown`.
4. Click **2D Pose Estimate** in the toolbar.



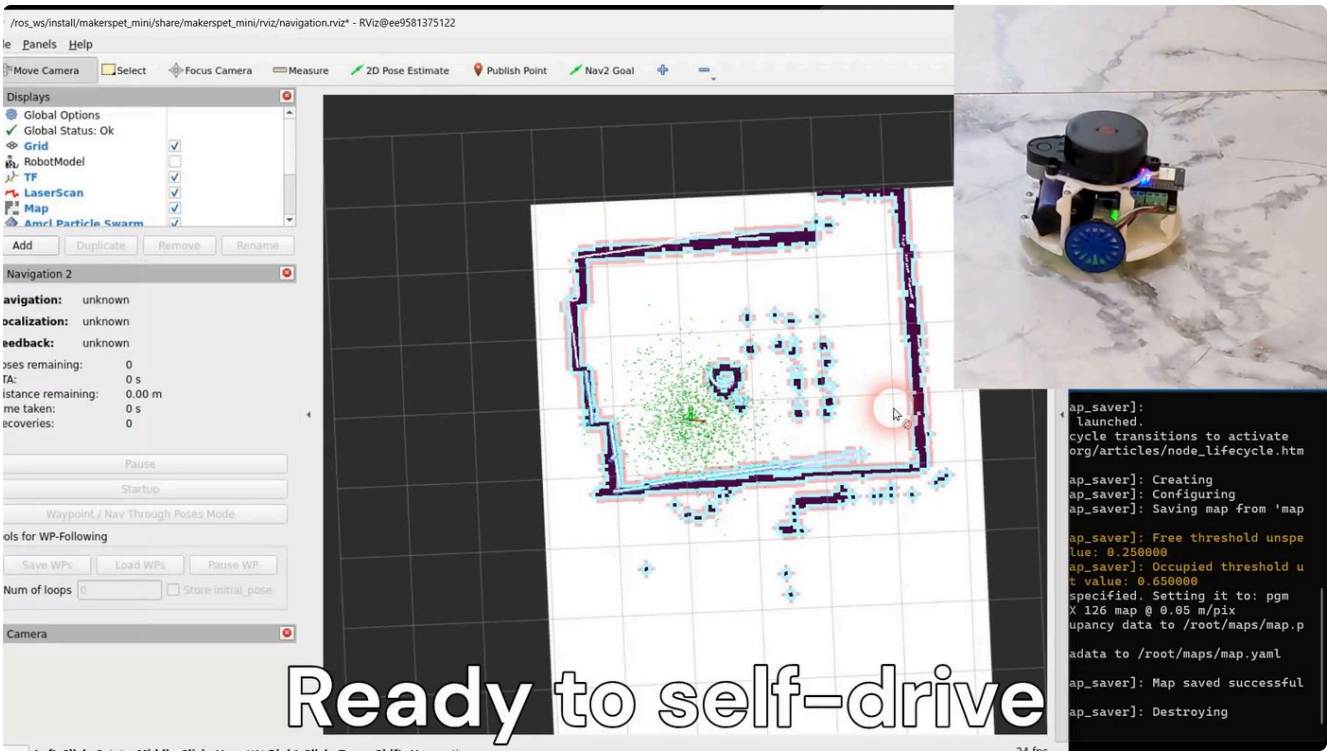
Click 2D Pose Estimate.

5. Click and hold at the robot's *current* location on the map, drag in the direction it faces, and release.



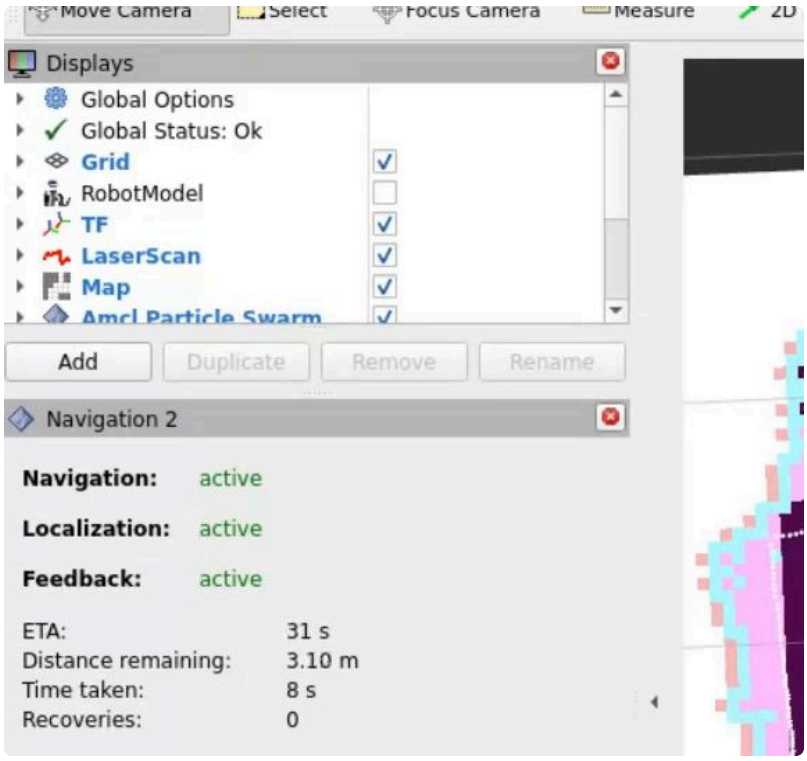
Click at the robot's location and drag to match its orientation.

6. A cloud of green arrows appears: the robot's guesses of its position. It tightens as the robot drives and matches LiDAR readings to the map. The robot is ready.



The green particle cloud shows the estimated position.

7. Set a destination with **Nav2 Goal**, as before.
8. The **Navigation 2** panel shows whether navigation and localization are `active`, the ETA, remaining distance, time taken and number of recoveries.



The Navigation 2 panel shows navigation status.

Shut down

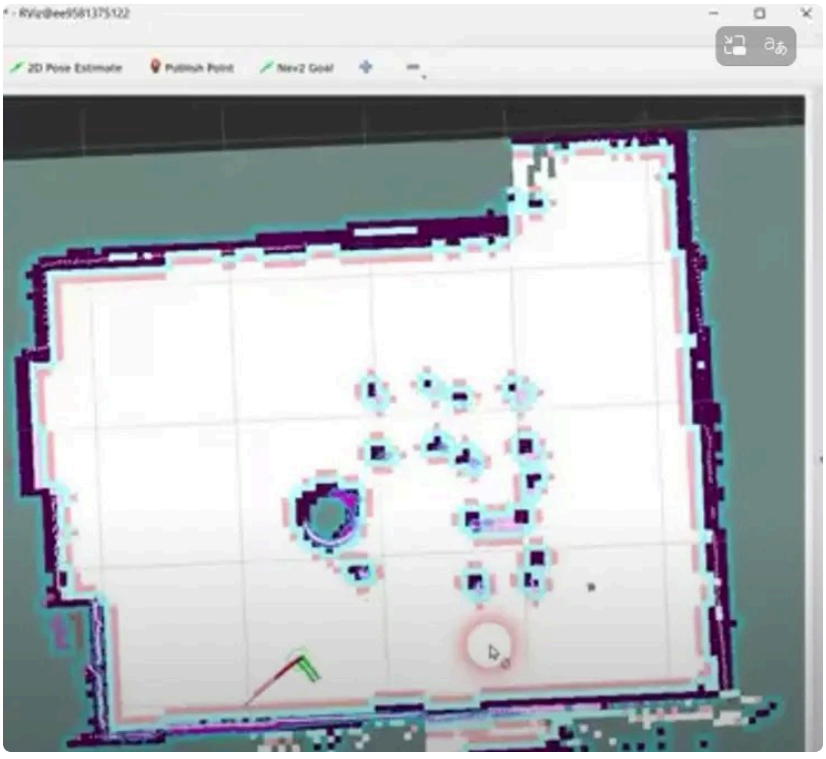
1. Press Ctrl-C in each window.
2. Type `exit` in the top window to stop the Docker container. It was started with `--rm`, so it is removed; your map is safe in `c:\maps`.
3. Close PowerShell.

If navigation fails: "Goal Failed" errors

Goal Failed means the robot found no safe path to your goal. Usually obstacles are too close together. The robot wants plenty of space around itself: if the path passes too close to an obstacle, even without touching it, the robot refuses to drive.

To fix it:

- Make a clean map. If it is skewed or messy, map again, driving slowly.
- Remove obstacles.
- Leave plenty of space between the remaining ones.



A nice, clean map.

> **WARNING**
> Navigation also fails if your PC is too slow. Watch for errors like `Transform data too old when converting from odom to map`: processing odometry took too long. Close other heavy programs, or use a faster PC.

Tuning navigation

To tune mapping and navigation, edit `navigation.yaml`. Inside the container it is here:

```
/ros_ws/src/makerspet_mini/config/navigation.yaml
```

Tuning takes experimentation; search for "ROS2 Nav2 tuning" for tutorials. The settings you will most likely change:

- **Speed.** The Nav2 controller, not teleop, sets autonomous speed. In `controller_server`, under `FollowPath`:
- `max_vel_x`: top forward speed in m/s (default 0.22). Raise it gradually, for example to 0.30, and set `max_speed_xy` to the same value.
- `max_vel_theta`: top turning speed in rad/s (default 1.0).
- `acc_lim_x` and `acc_lim_theta`: optionally reach target speed quicker.
- **Inflation radius.** How far the robot keeps from obstacles. Larger is safer but causes more "Goal Failed" errors in tight spaces.

> **WARNING**
> Keep `max_vel_x` comfortably below 0.4 m/s, the drivetrain's mechanical limit (200 RPM motor, derated 90%). Higher speed also gives the planner less time to react, so raise values in small steps and test each change.

Restart the navigation launch (Ctrl-C, then launch again) to apply changes. The `--rm` container loses your edits when it stops; to keep them, commit it to your image from a Windows command window before you exit:

```
docker container commit makerspet kaiaai/kaiaai:iron
# or, if you run the Jazzy image:
docker container commit makerspet kaiaai/kaiaai:jazzy
```

> **TIP**
> Found better values than the `navigation.yaml` defaults? Please share them in the Maker's Pet support forum.

RUNNING MORE THAN ONE ROBOT

Start one Docker container per robot, each with a unique `--name` and host UDP port, and point each robot's WiFi configuration at its port. For example (other options as above):

```
docker run --name makerspet1 ... -p 8888:8888/udp ... kaiaai/kaiaai:iron
docker run --name makerspet2 ... -p 8889:8888/udp ... kaiaai/kaiaai:iron
```

Then open a shell with `docker exec -it makerspet1 bash` or `docker exec -it makerspet2 bash`.

6. Practice in Simulation (No Robot Needed)

Video: Simulate Arduino/ROS2 LiDAR robot using ROS2 Gazebo

Companion article: Tutorial: Map, Navigate ROS2 Robot in Simulation

You can do this chapter any time after installing the PC software, even before your kit arrives. A simulated Maker's Pet Mini robot drives around a simulated living room in Gazebo, a 3D robot simulator. It "sees" with a simulated LiDAR and runs the same ROS2 mapping (SLAM) and navigation (Nav2) software as the real robot. You will:

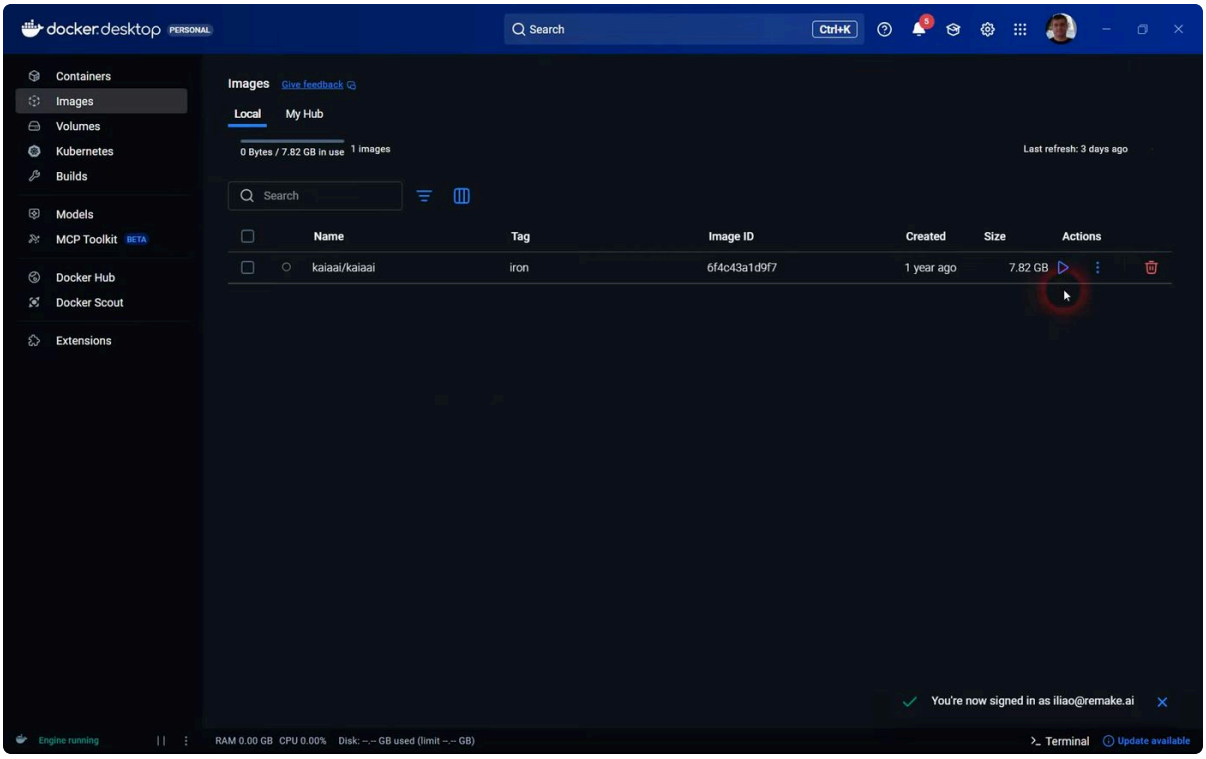
1. start the Docker container and the Gazebo simulation,
2. drive the robot from the keyboard,
3. launch SLAM and Nav2 and watch the map build up in RViz,
4. send the robot to goals on the map until the room is explored,
5. save the map to your PC.

It takes about 15 minutes once the software is installed. The video is recorded on Windows; the Ubuntu commands from the companion article are included too.

> **WHAT YOU NEED FIRST**
>
> - **Windows PC:** WSL2, Docker Desktop, the VcXsrv X server (XLaunch), Windows PowerShell and the `kaiaai/kaiaai:iron` Docker image, installed as in Chapter 2.
> - **Ubuntu PC:** Docker Engine, Terminator and the `kaiaai/kaiaai:iron` image, as in Chapter 8.
> - Either way, skip the Arduino and firmware steps.
> - Optional: XnView (or any `.pgm` viewer) to look at the saved map.

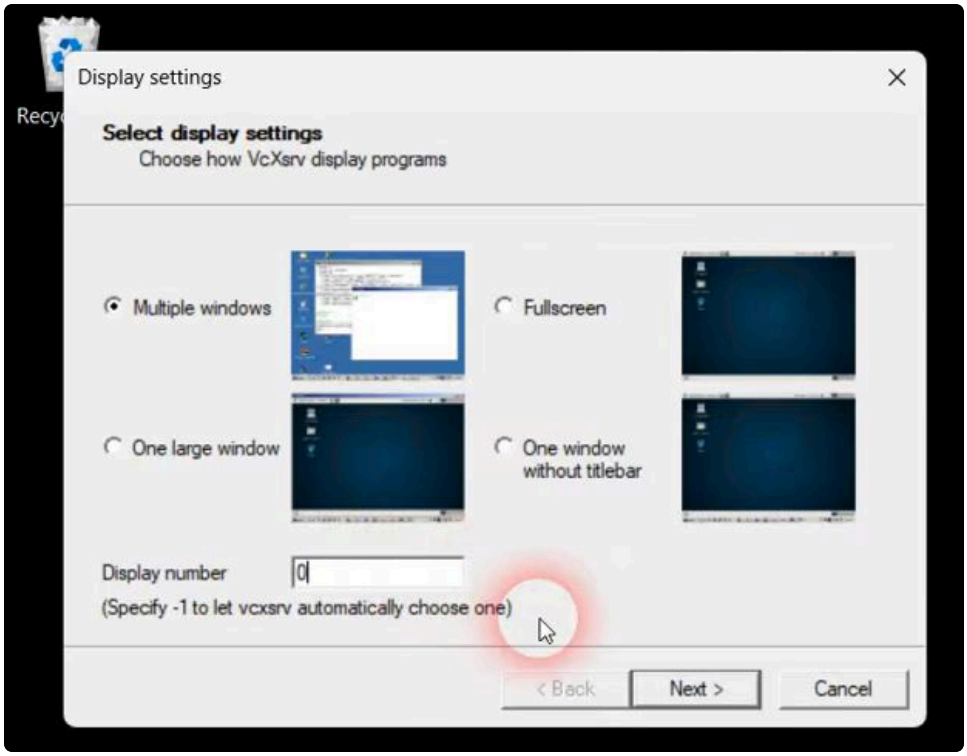
Start Docker, the X server and PowerShell (Windows)

1. Start **Docker Desktop** and wait until it finishes "Starting the Docker Engine...".
2. On the **Images** tab, check that `kaiaai/kaiaai` with the tag `iron` is listed. If not, pull it as in Chapter 2.



Docker Desktop is running with the kaiaai/kaiaai:iron image.

3. Start **XLaunch**. Keep **Multiple windows**, set **Display number** to `0` (not the default `-1`) and finish the wizard as in Chapter 2. Gazebo and RViz windows show up on this X server.



In XLaunch, set the display number to 0.

4. Start **Windows PowerShell**.

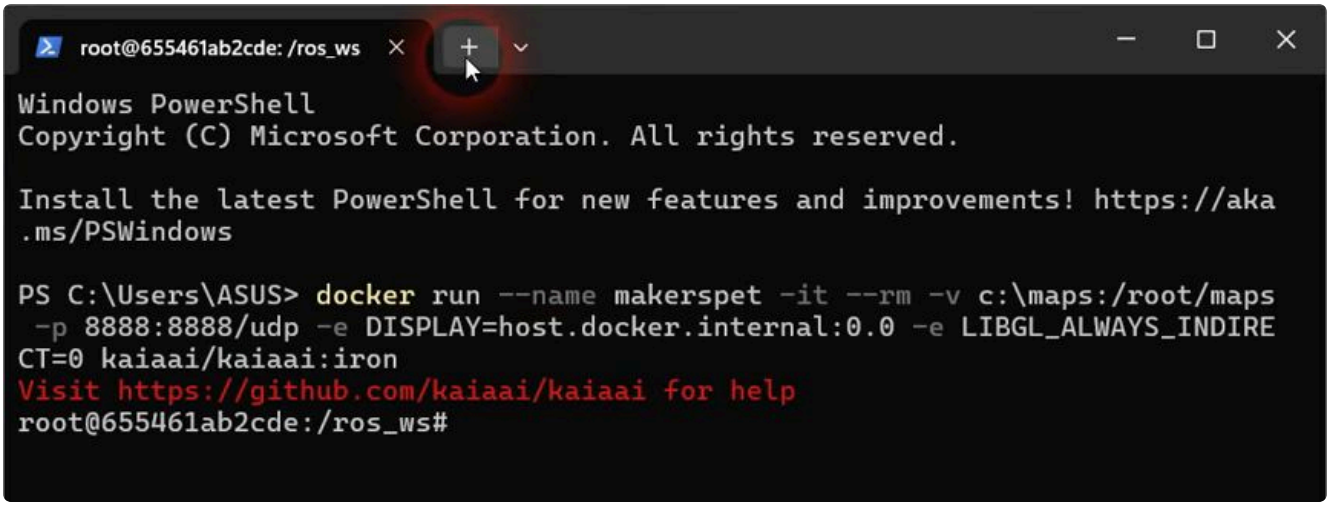
> **TIP**
> The `docker run` command below shares `C:\maps` with the container as `/root/maps`, so maps you save in Docker land on your PC.

Launch the Docker container

1. In PowerShell, start the container (copy the command from the companion article to avoid typos):

```
docker run --name makerspet -it --rm -v c:\maps:/root/maps -p 8888:8888/udp -e DISPLAY=hos
t.docker.internal:0.0 -e LIBGL_ALWAYS_INDIRECT=0 kaiaai/kaiaai:iron
```

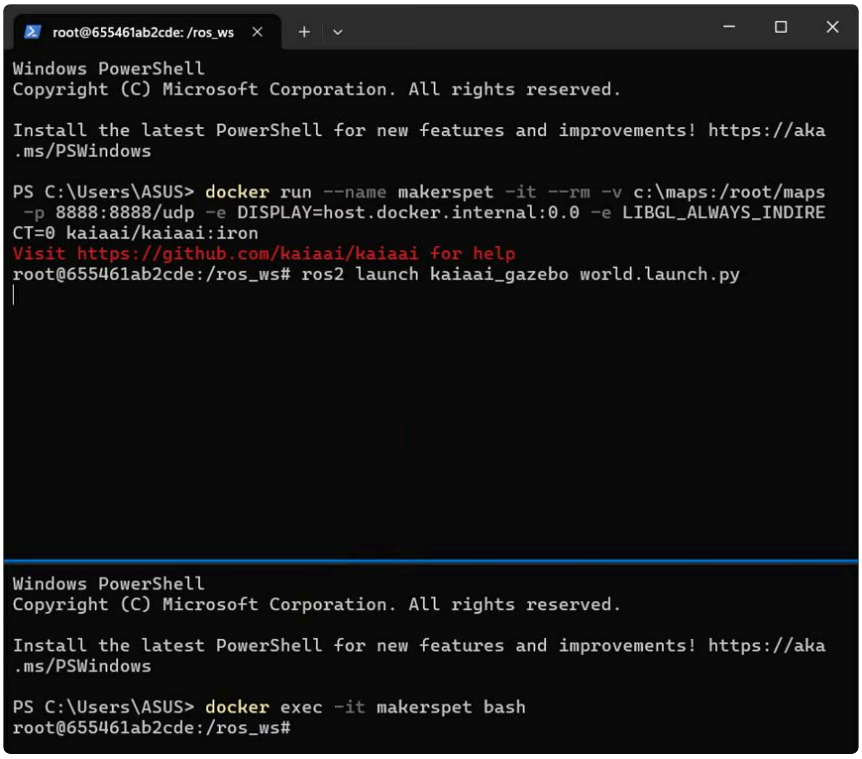
The prompt changes to something like `root@655461ab2cde:/ros_ws#`. `--name makerspet` lets you open more terminals in the same container later.



The container is running; the prompt starts with root@.

2. Open two more terminals. I split the window into panes (in Windows Terminal, press **Alt+Shift+-**); tabs or windows work too.
3. In each new one, enter the running container:

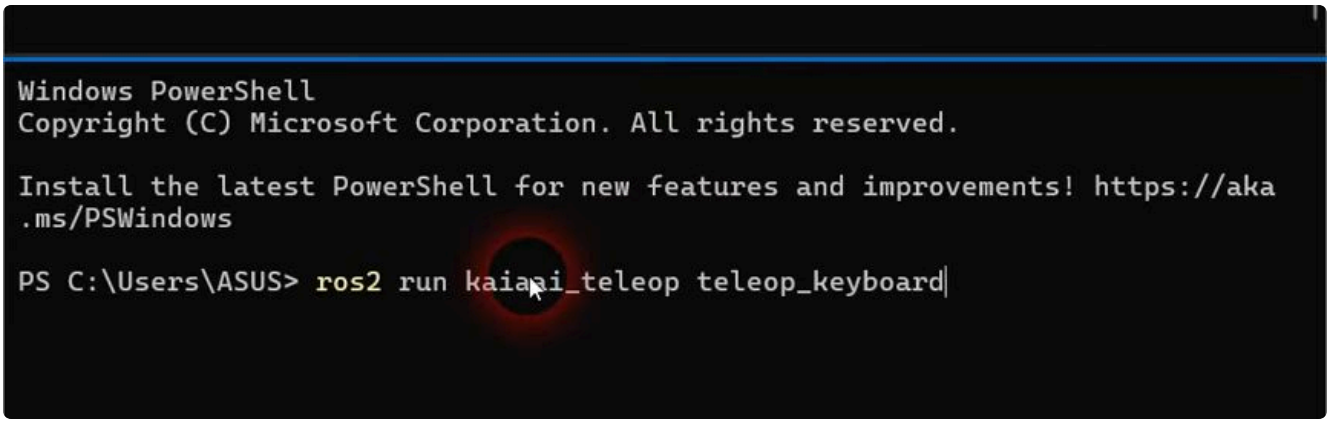
```
docker exec -it makerspet bash
```



A second pane opened in the same container with docker exec.

WARNING

A new pane or tab starts in plain PowerShell (`PS C:\Users\...>`), where ROS2 commands don't work. Run `docker exec -it makerspet bash` first and check that the prompt starts with `root@` . I made this mistake in the video.



Oops: this ros2 command is typed into PowerShell. Run docker exec first.

On an Ubuntu PC

Use these commands from the companion article instead (split Terminator into three terminals):

```
sudo docker run --name makerspet -it --rm -v ~/maps:/root/maps -p 8888:8888/udp -e DISPLAY -e QT_X11_NO_MITSHM=1 --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" --volume="${XAUTHORITY}:/root/.Xauthority" kaiaai/kaiaai:iron
```

```
sudo docker exec -it makerspet bash
```

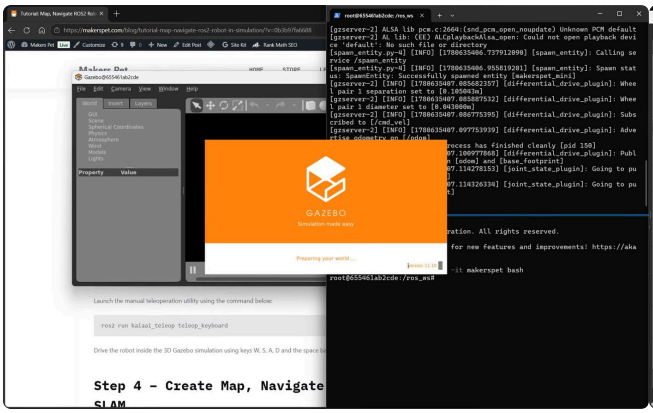
Maps are saved to `~/maps`.

Launch the Gazebo simulation

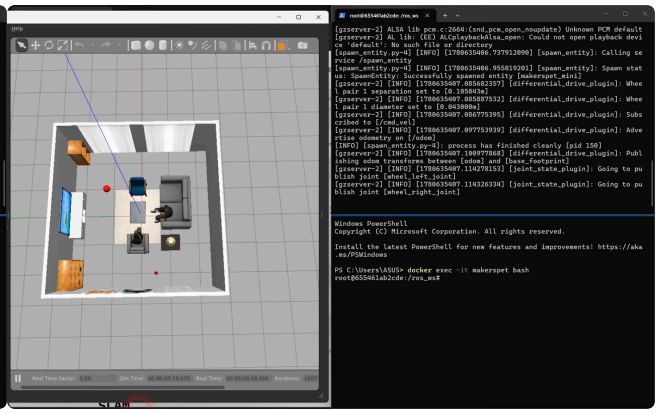
1. In the first container terminal, start the simulated world:

   ```
   ros2 launch kaiaai_gazebo world.launch.py
   ```

2. Wait for the Gazebo window; the first start takes a while. The terminal prints `Successfully spawned entity [makerspet_mini]`.



Gazebo is starting up.



The simulated living room; the robot is the small red disc near the bottom.

> **NOTE**
> Ignore the ALSA / `Could not open playback device` errors: there is no sound card in Docker.

> **MOVING AROUND IN GAZEBO**
> Mouse wheel zooms, dragging pans. The big red ball is just an object; the robot is the small red disc with the black LiDAR on top.

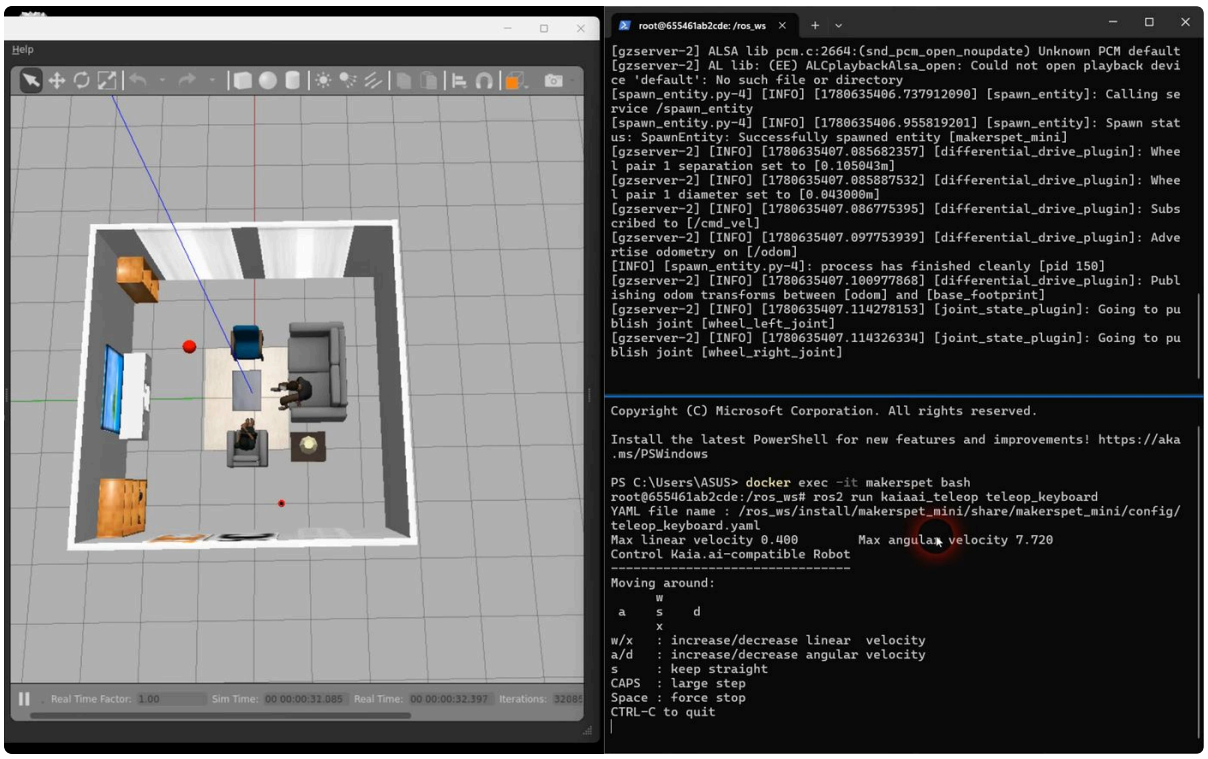
Drive the robot by hand

1. In the second container terminal, start keyboard teleop:

```
ros2 run kaiaai_teleop teleop_keyboard
```

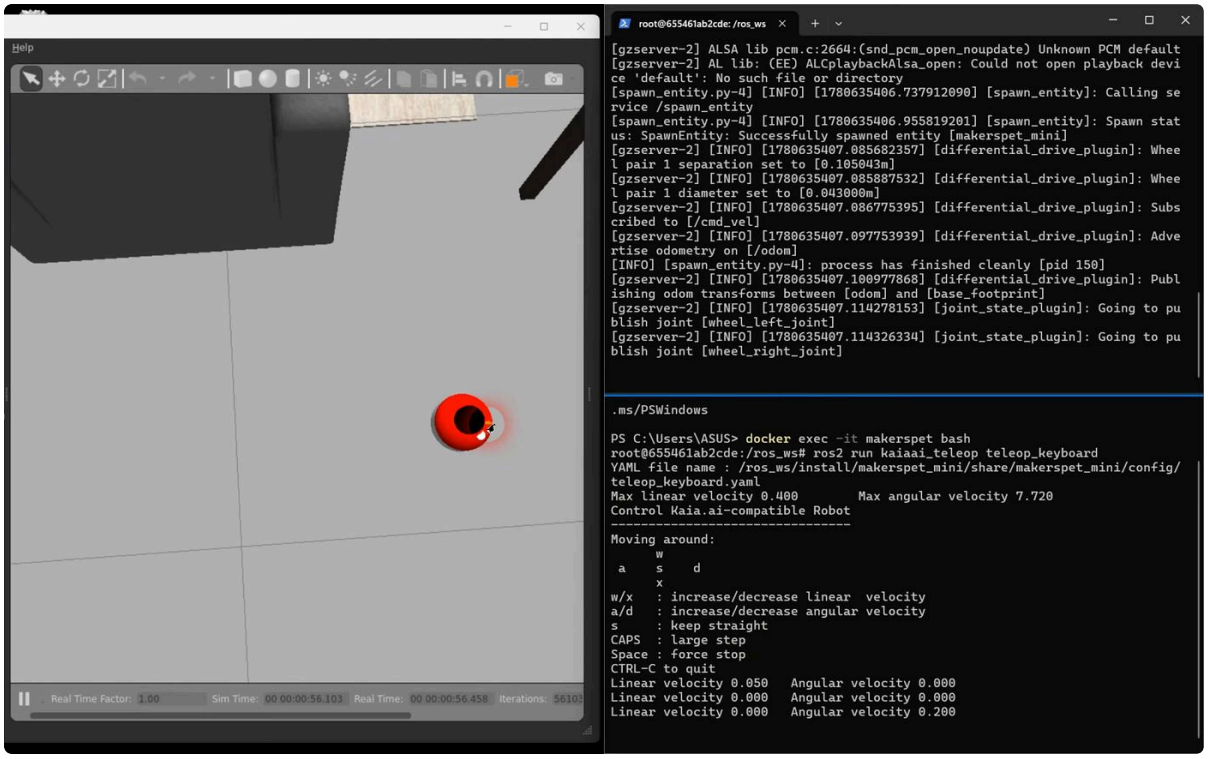
2. It lists its keys:

- `w` / `x` : increase / decrease linear (forward) speed
- `a` / `d` : increase / decrease angular (turning) speed
- `s` : keep straight (stop turning)
- `Space` : force stop
- `CAPS` : large steps
- `Ctrl-C` : quit



teleop_keyboard next to Gazebo; key help at the bottom right.

3. Click into the teleop terminal and drive. Each key press changes the speed a little. The robot keeps that speed until you change it; press **Space** to stop.



Driving the robot; each key press prints a new velocity.

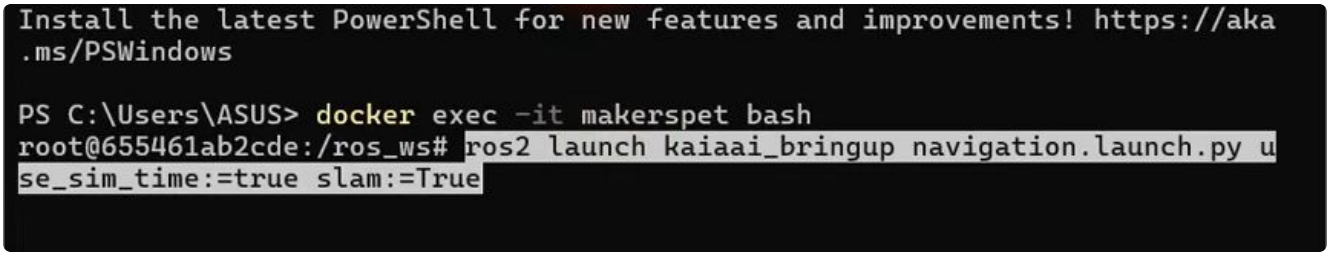
Launch SLAM and Nav2

Leave Gazebo and teleop running.

1. In the third container terminal, run:

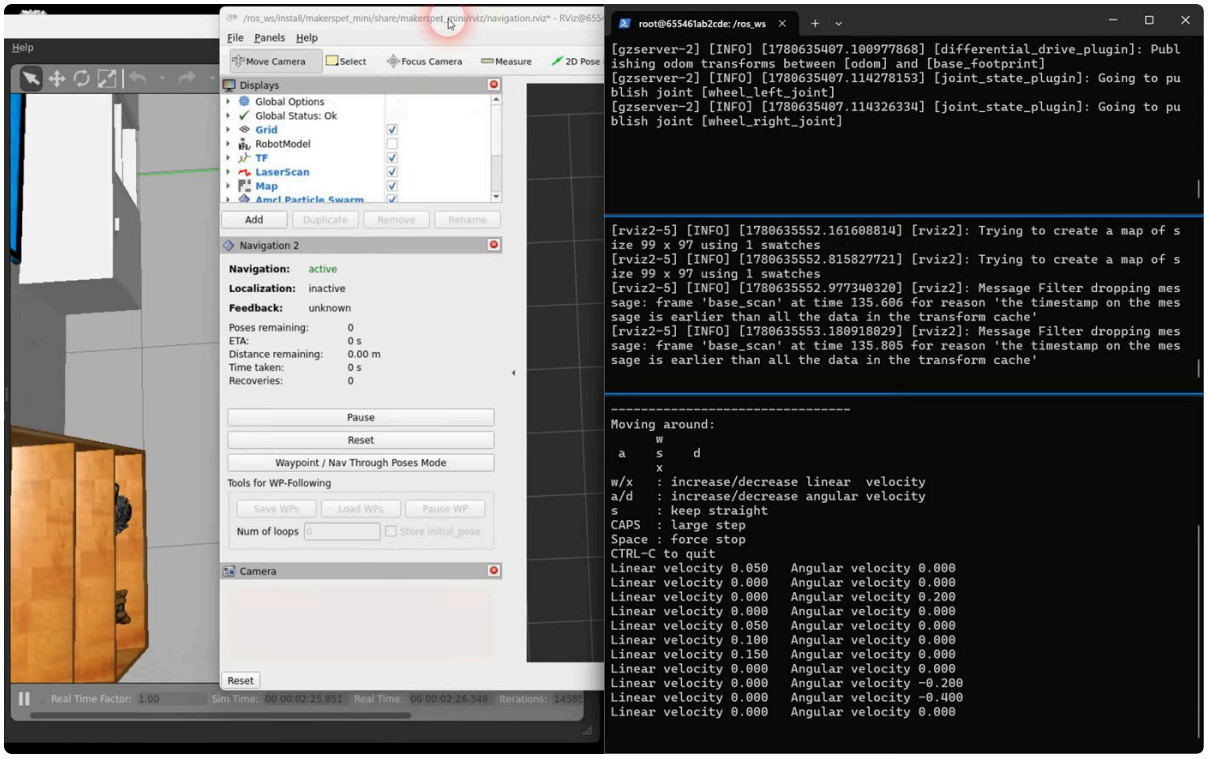
```
ros2 launch kaiaai_bringup navigation.launch.py use_sim_time:=true slam:=True
```

`use_sim_time:=true` uses the simulator's clock; `slam:=True` builds a new map instead of loading a saved one.



The navigation launch command.

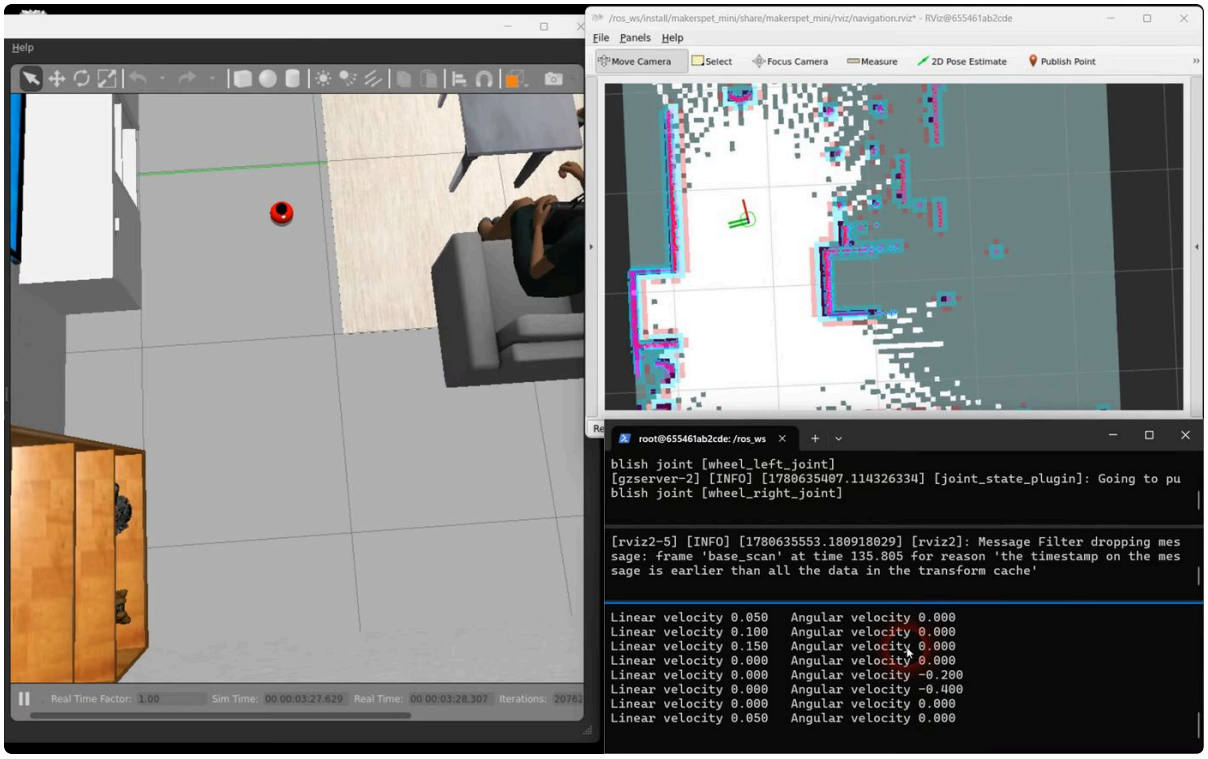
2. RViz opens. With SLAM, Localization shows `inactive`; that's expected. On the map, white is free floor, dark outlines are walls and furniture, gray-green is unexplored.



RViz with the Displays and Navigation 2 panels.

> **ARRANGING THE WINDOWS**
> I hide the RViz side panels with the small arrow on the map's left edge and put Gazebo left, RViz right, terminals below. If RViz is too narrow, the **Nav2 Goal** button hides behind a » on the toolbar; widen the window.

3. Drive around a little with teleop and watch the map grow.



Driving with teleop while SLAM builds the map.

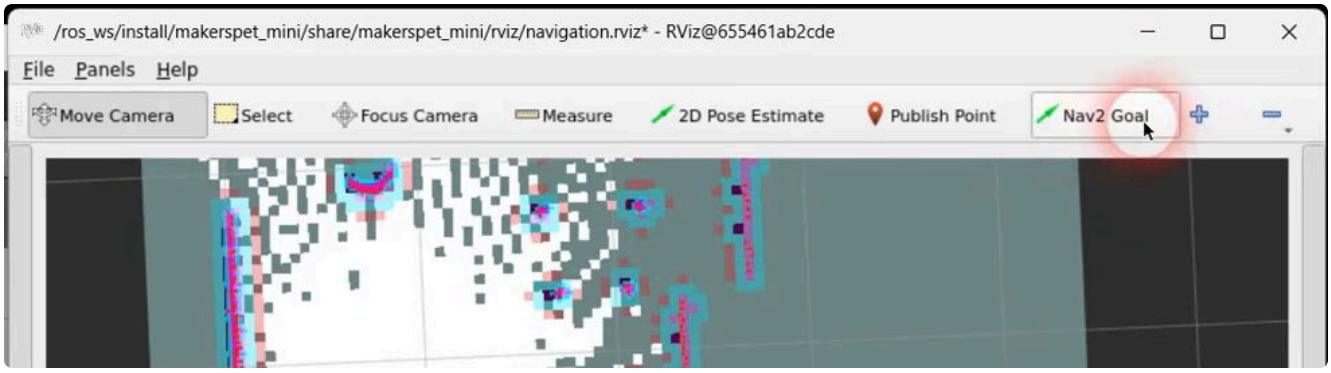
4. Before letting the robot drive itself, stop teleop with **Ctrl-C**, or teleop and Nav2 will fight over speed commands.

```
Linear velocity 0.150   Angular velocity 0.000
Linear velocity 0.000   Angular velocity 0.000
Linear velocity 0.000   Angular velocity -0.200
Linear velocity 0.000   Angular velocity -0.400
Linear velocity 0.000   Angular velocity 0.000
Linear velocity 0.050   Angular velocity 0.000
Linear velocity 0.000   Angular velocity 0.000
Stopping the robot and exiting on CTRL-C press
root@655461ab2cde:/ros_ws#
```

Teleop stopped with Ctrl-C.

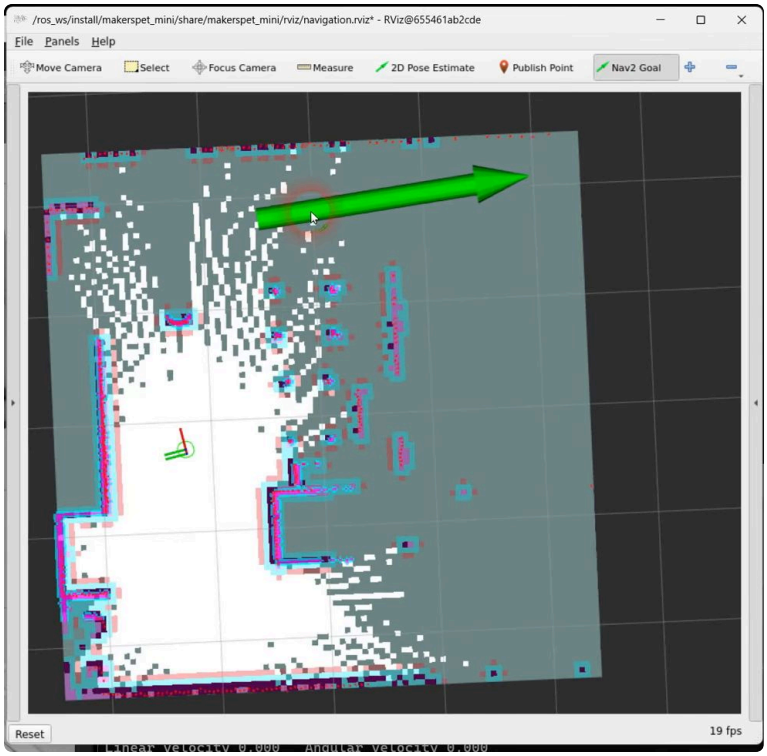
Send the robot to goals

1. Click **Nav2 Goal** on the RViz toolbar.



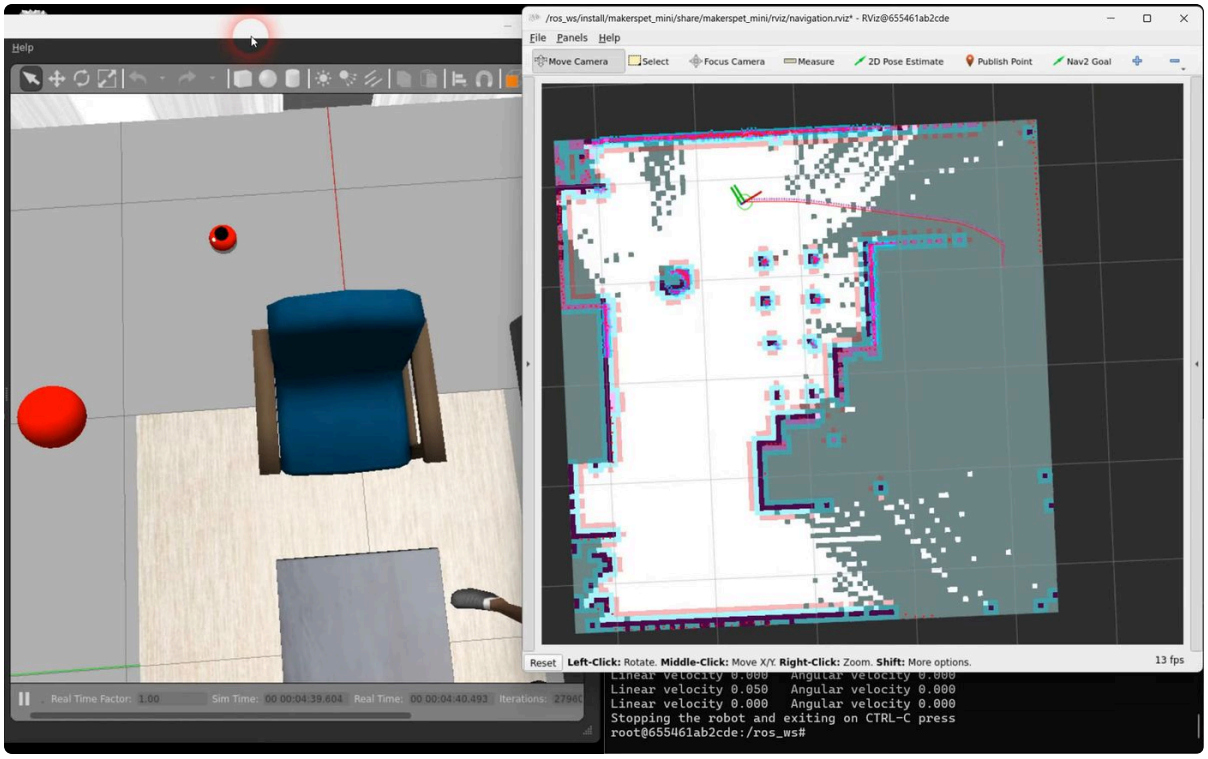
The Nav2 Goal button.

2. Left-click and hold where you want the robot to go.
3. Drag toward the direction it should face on arrival. A green arrow shows the goal.
4. Release.



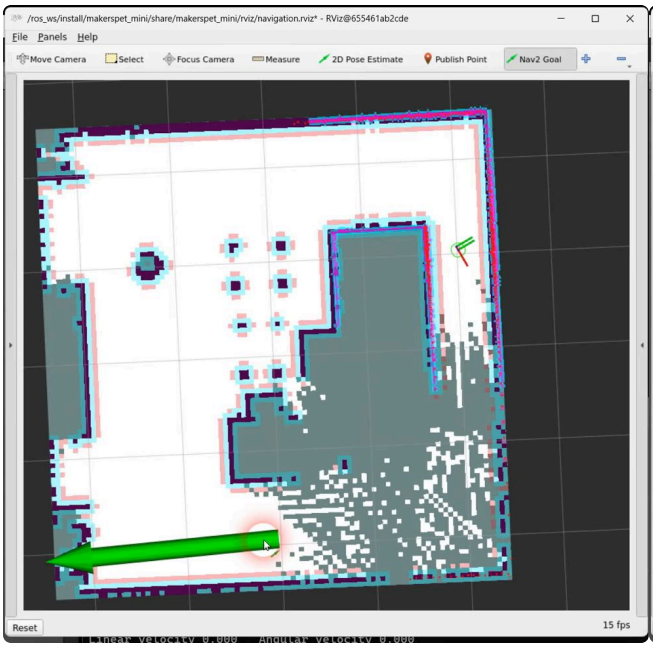
Click, hold and drag; the green arrow is the goal.

5. Nav2 plans a path (thin pink line) and the robot follows it. On arrival the navigation terminal prints `Reached the goal!` / `Goal succeeded`.

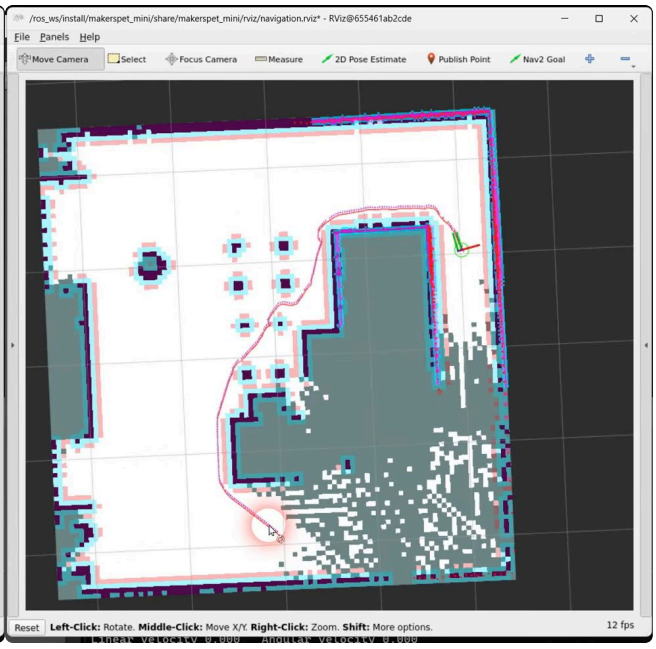


Nav2 plans a path and the robot follows it.

6. Keep setting goals, preferably at the edge of the explored (white) area, so the LiDAR sees new parts of the room.



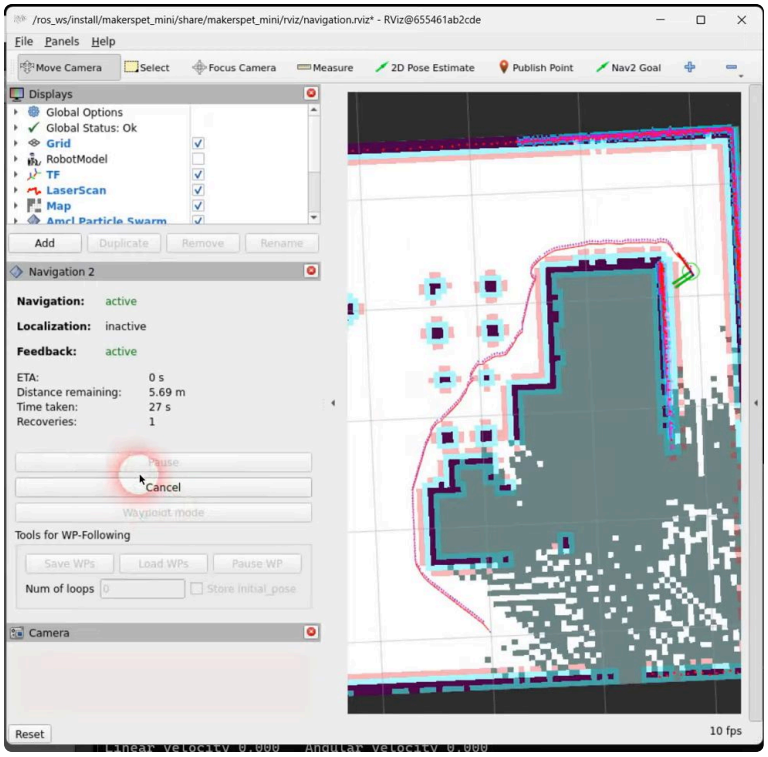
A long goal into an unexplored corner.



Nav2 plans around the furniture.

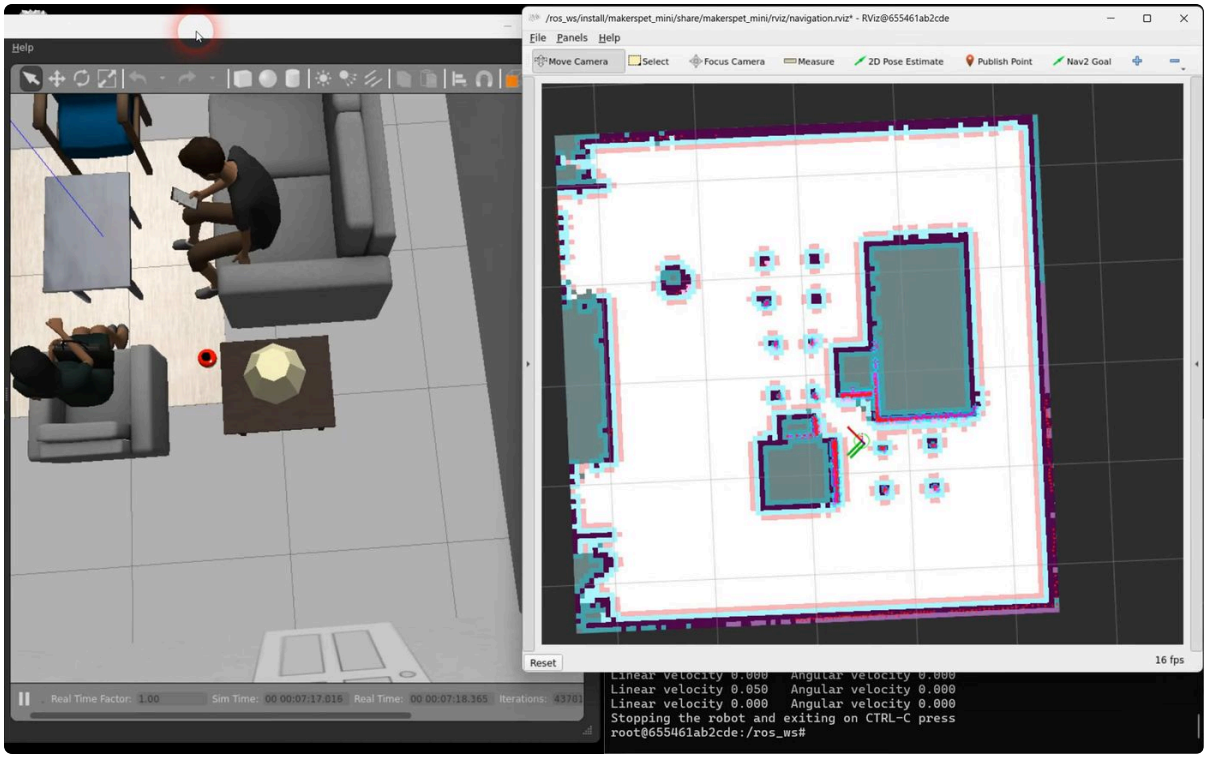
IF THE ROBOT GETS STUCK

If the **Recoveries** counter in the Navigation 2 panel climbs, show the side panel, click **Cancel** and give a closer goal.



Canceling a goal.

7. Repeat until the walls form a closed outline with no gray-green left inside.



The finished map.

> **NOTE**
> You can also map the whole room by hand with `teleop_keyboard`; SLAM works either way.

Save the map

1. In a free container terminal, run:

```
ros2 run nav2_map_server map_saver_cli -f ~/maps/map --ros-args -p save_map_timeout:=60.0
```

```
Linear velocity 0.000   Angular velocity -0.200
Linear velocity 0.000   Angular velocity -0.400
Linear velocity 0.000   Angular velocity 0.000
Linear velocity 0.050   Angular velocity 0.000
Linear velocity 0.000   Angular velocity 0.000
Stopping the robot and exiting on CTRL-C press
root@655461ab2cde:/ros_ws# ros2 run nav2_map_server map_saver_cli -f ~/maps/
map --ros-args -p save_map_timeout:=60.0
```

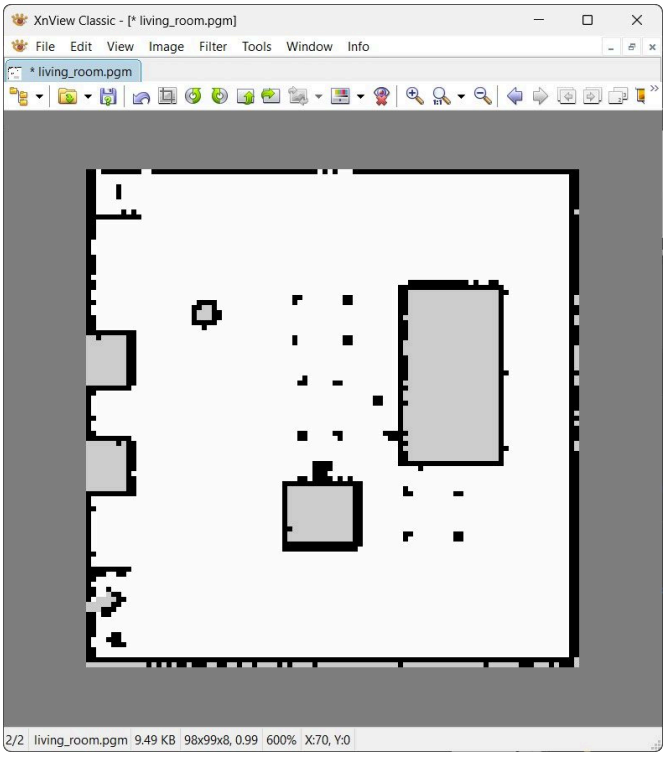
The map saver command.

2. Wait for `Map saved successfully`. `map.pgm` (the map image) and `map.yaml` (size, resolution, origin) go to `/root/maps`, which is `C:\maps` on Windows or `~/maps` on Ubuntu.

```
etting it to default value: 0.650000
[WARN] [map_io]: Image format unspecified. Setting it to: pgm
[INFO] [map_io]: Received a 100 X 97 map @ 0.05 m/pix
[INFO] [map_io]: Writing map occupancy data to /root/maps/map.pgm
[INFO] [map_io]: Writing map metadata to /root/maps/map.yaml
[INFO] [map_io]: Map saved
[INFO] [1780635867.767172691] [map_saver]: Map saved successfully
[INFO] [1780635867.768260831] [map_saver]: Destroying
root@655461ab2cde:/ros_ws#
```

Map saved: /root/maps/map.pgm and map.yaml.

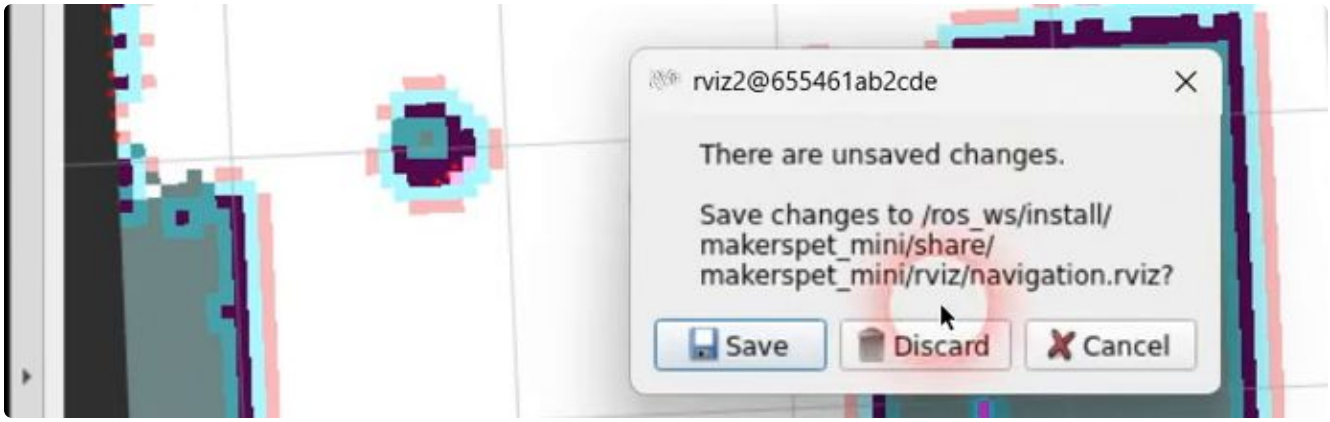
3. Open `map.pgm` in XnView to view it.



The saved map in XnView.

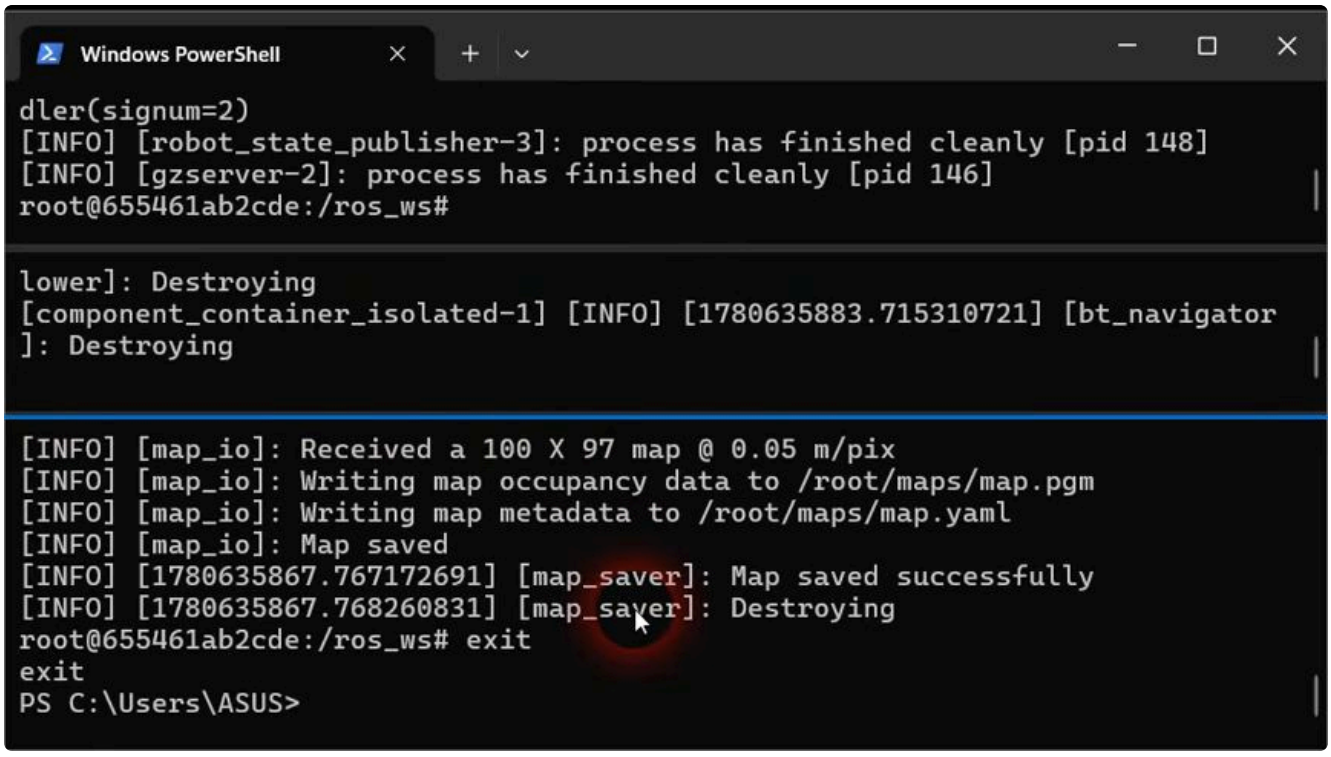
Shut everything down

1. Close RViz. When asked to save `navigation.rviz`, click **Discard** unless you meant to change the layout.



Closing RViz: click Discard.

2. Press **Ctrl-C** in the navigation and Gazebo terminals and wait for them to exit.
3. Type `exit` in each container terminal. Leaving the `docker run` one stops the container, and `--rm` deletes it. Your map is safe in `C:\maps` (or `~/maps`) on your PC.



Leaving the container with exit.

7. Program Your Robot

In this chapter you write and run two short Python programs:

- **Face the closest object.** The robot turns in place toward anything within 25 cm. It's small enough to try on your desk.
- **Follow me.** The robot follows the closest object in front of it, staying 40 cm away. Walk slowly and it follows your feet.

You try each program in the simulation first (Chapter 6), then on the real robot. No robotics theory is needed: you copy the code, run it and watch the robot. A short explanation follows each program.

How a robot program talks to the robot

The robot software is built on ROS 2, where programs exchange messages on named *topics*. Your programs use just two:

- `/scan`: the LiDAR's distance readings, one full circle about five times a second.
- `/cmd_vel`: drive commands, a forward speed and a turning speed. The keyboard teleop app from Chapter 3 sends its commands here too.

Each program reads `/scan`, decides how to move, and sends a command to `/cmd_vel`. The same program works in the simulation and on the real robot.

Set up a folder for your code

The robot software runs in a Docker container, and everything you save inside it is lost when the container exits. Keep your programs on your PC instead and share the folder with the container.

1. Create the folder `C:\robot_code`.
2. Start the container for the simulation as in Chapter 6, with one more `-v` option for the code folder:

   ```
   docker run --name makerspet -it --rm -v c:\maps:/root/maps -v c:\robot_code:/root/code -p
   8888:8888/udp -e DISPLAY=host.docker.internal:0.0 -e LIBGL_ALWAYS_INDIRECT=0 kaiaai/kaiaa
   i:iron
   ```

   Inside the container, `C:\robot_code` appears as `/root/code`.

3. Download both programs into it:

```
cd /root/code
curl -O https://raw.githubusercontent.com/makerspet/ebook/main/self-driving/v1/book/code/face_closest.py
curl -O https://raw.githubusercontent.com/makerspet/ebook/main/self-driving/v1/book/code/follow_me.py
```

   You can also type them in from this chapter: save each one in `C:\robot_code` with any text editor, for example Notepad, choosing **All files** as the file type.

> **UBUNTU PC**
> Use the Ubuntu `docker run` command from Chapter 8 with `-v ~/robot_code:/root/code` added, and keep your programs in `~/robot_code`.

Program 1: face the closest object

Try it in the simulation

1. In the first container terminal, start the simulated world:

```
ros2 launch kaiaai_gazebo world.launch.py
```

2. In a second container terminal, place the red ball 30 cm to the robot's right:

```
ros2 run gazebo_ros spawn_entity.py -entity ball -database red_ball_10in -x -2.0 -y -0.8
```

   The robot starts at `x = -2.0`, `y = -0.5`, facing along `x`. Always place objects so they don't overlap the robot; a ball spawned on top of it knocks the robot away.

3. Run the program:

```
cd /root/code
python3 face_closest.py
```

4. Watch the robot turn right until it faces the ball. The terminal shows what the robot sees, with `0` meaning straight ahead:

```
closest: 0.17 m at  -1 degrees
```

5. In the Gazebo window, press **T** (translate mode), drag the ball to another spot near the robot, and watch it turn again. Anything further than 25 cm away is ignored.

6. Press **Ctrl+C** to stop the program. It stops the wheels before it exits.

The program

```
"""Turn in place to face the closest object within 25 cm.

Run inside the robot software's container:  python3 face_closest.py
Stop with Ctrl+C.
"""
import math

import rclpy
from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data
from rclpy.signals import SignalHandlerOptions
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan

MAX_DISTANCE = 0.25  # meters: ignore anything further away
TURN_SPEED = 1.0     # radians per second, at most
ON_TARGET = math.radians(5)  # close enough: stop turning


def robot_angle(scan, i):
    """Angle of scan point i as seen by the robot: 0 = straight ahead, + = left, - = right.

    The robot's LiDAR is mounted upside down, so its angles run the other way around,
    starting from the back. 180 degrees minus the scan angle fixes both.
    """
    a = math.pi - (scan.angle_min + i * scan.angle_increment)
    return math.atan2(math.sin(a), math.cos(a))  # wrap to -180..180 degrees


class FaceClosest(Node):
    def __init__(self):
        super().__init__('face_closest')
        self.cmd = self.create_publisher(Twist, 'cmd_vel', 10)
        self.create_subscription(LaserScan, 'scan', self.on_scan, qos_profile_sensor_data)
```

```
    def on_scan(self, scan):
        # Find the closest valid point within MAX_DISTANCE.
        closest, angle = MAX_DISTANCE, None
        for i, r in enumerate(scan.ranges):
            if scan.range_min < r < closest:
                closest, angle = r, robot_angle(scan, i)
        twist = Twist()
        if angle is not None:
            print(f'closest: {closest:.2f} m at {math.degrees(angle):4.0f} degrees')
            if abs(angle) > ON_TARGET:
                # Turn toward it: left for positive angles, right for negative ones.
                twist.angular.z = max(-TURN_SPEED, min(TURN_SPEED, 2.0 * angle))
        self.cmd.publish(twist)  # an empty Twist means "stop"


def main():
    rclpy.init(signal_handler_options=SignalHandlerOptions.NO)  # we handle Ctrl+C ourselves
    node = FaceClosest()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    node.cmd.publish(Twist())  # stop the wheels before exiting
    node.destroy_node()
    rclpy.try_shutdown()


if __name__ == '__main__':
    main()
```

How it works

- `on_scan` runs every time a new LiDAR scan arrives on `/scan`.
- The loop goes through all the distance readings and keeps the closest one under 25 cm.
- `robot_angle` turns the reading's position in the scan into an angle around the robot: `0` straight ahead, positive to the left, negative to the right.
- If the object is more than 5 degrees off, the robot turns toward it. The turning speed grows with the angle, so it slows down as it lines up. `twist.angular.z` is the turning speed.
- Publishing an empty `Twist` to `/cmd_vel` stops the robot.

Try it on the robot

1. Bring up the robot as in Chapter 3, starting the container with the code folder added:

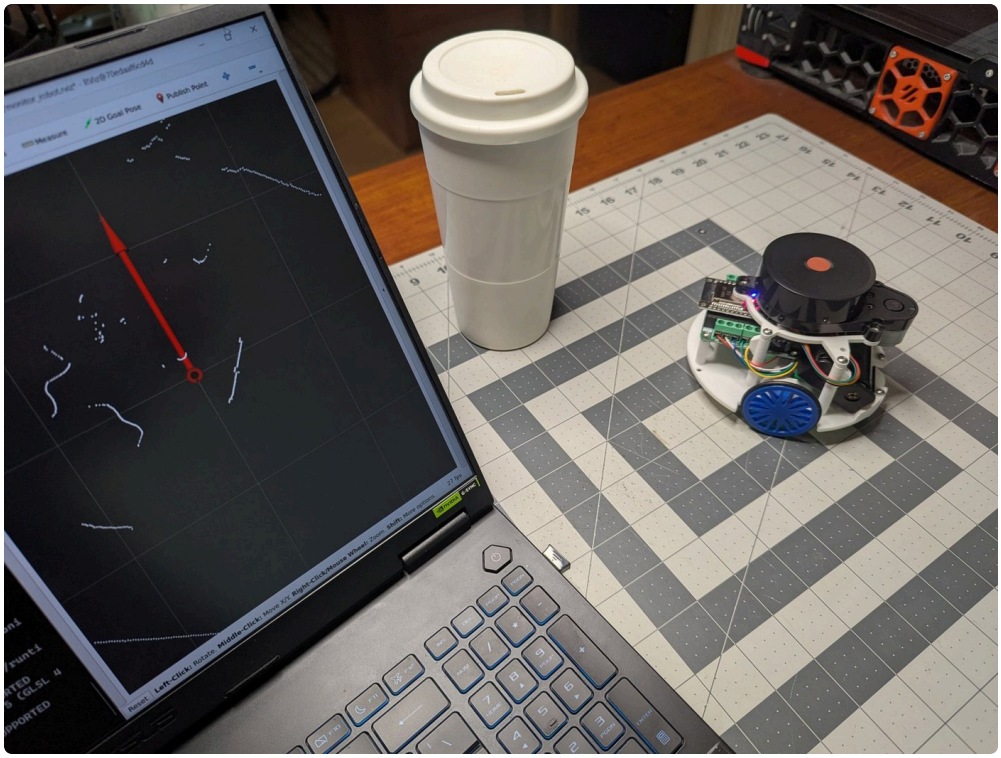
```
docker run --name makerspet -it --rm -v c:\maps:/root/maps -v c:\robot_code:/root/code -p
8888:8888/udp -p 4430:4430/tcp -e DISPLAY=host.docker.internal:0.0 -e LIBGL_ALWAYS_INDIREC
T=0 kaiaai/kaiaai:iron
```

Then run `ros2 launch kaiaai_bringup physical.launch.py` and turn the robot on.

2. Place the robot on your desk, away from walls and other objects.
3. In a second container terminal, run `python3 /root/code/face_closest.py`.
4. Hold your hand or a small object, such as a cup, about 15–20 cm from the robot and move it around. The robot turns to face it. With a cup placed to the robot's front left, the output looks like this:

```
closest: 0.14 m at   52 degrees
closest: 0.14 m at   52 degrees
...
closest: 0.14 m at   -2 degrees
```

The cup starts 52 degrees to the left; once the robot faces it, the angle drops to about 0 and the robot stops.



The robot has turned to face the cup. The laptop shows the live LiDAR scan in RViz.

> **WHICH WAY IS FRONT?**
> Before relying on the robot's moves, check the printout: hold an object in front of the robot and it should read about `0 degrees`; to its left, about `90`; to its right, about `-90`.

Program 2: follow me

Try it in the simulation

1. Start a fresh simulation (`Ctrl+C` and relaunch `world.launch.py`) so the robot is back at its starting spot, facing along `x`.
2. Place the ball 70 cm in front of the robot:

```
ros2 run gazebo_ros spawn_entity.py -entity ball -database red_ball_10in -x -1.3 -y -0.5
```

3. Run the program:

```
python3 /root/code/follow_me.py
```

4. The robot drives toward the ball and stops about 40 cm from it:

```
following: 0.40 m at    3 degrees
```

5. In Gazebo, drag the ball (**T**) a little further away or to the side. The robot follows. Drag it closer and the robot backs up.
6. Press **Ctrl+C** to stop.

The program

```
"""Follow the closest object in front of the robot, staying 40 cm away.

Run inside the robot software's container:  python3 follow_me.py
Stop with Ctrl+C.
"""
import math

import rclpy
from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data
from rclpy.signals import SignalHandlerOptions
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan

KEEP_DISTANCE = 0.40   # meters: how far behind to stay
SEE_DISTANCE = 1.00    # meters: ignore anything further away
SEE_ANGLE = math.radians(30)  # look 30 degrees to each side of straight ahead
MAX_SPEED = 0.15       # meters per second
MAX_TURN = 1.0         # radians per second


def robot_angle(scan, i):
    """Angle of scan point i as seen by the robot: 0 = straight ahead, + = left, - = right.

    The robot's LiDAR is mounted upside down, so its angles run the other way around,
    starting from the back. 180 degrees minus the scan angle fixes both.
    """
    a = math.pi - (scan.angle_min + i * scan.angle_increment)
    return math.atan2(math.sin(a), math.cos(a))  # wrap to -180..180 degrees


class FollowMe(Node):
    def __init__(self):
        super().__init__('follow_me')
        self.cmd = self.create_publisher(Twist, 'cmd_vel', 10)
        self.create_subscription(LaserScan, 'scan', self.on_scan, qos_profile_sensor_data)
```

```
    def on_scan(self, scan):
        # Find the closest point in the cone in front of the robot.
        closest, angle = SEE_DISTANCE, None
        for i, r in enumerate(scan.ranges):
            a = robot_angle(scan, i)
            if abs(a) < SEE_ANGLE and scan.range_min < r < closest:
                closest, angle = r, a
        twist = Twist()
        if angle is not None:
            print(f'following: {closest:.2f} m at {math.degrees(angle):4.0f} degrees')
            # Drive to close the gap (back up if too close) and steer toward the target.
            twist.linear.x = max(-MAX_SPEED, min(MAX_SPEED, 0.8 * (closest - KEEP_DISTANCE)))
            twist.angular.z = max(-MAX_TURN, min(MAX_TURN, 2.0 * angle))
        self.cmd.publish(twist)  # nothing in sight: stop


def main():
    rclpy.init(signal_handler_options=SignalHandlerOptions.NO)  # we handle Ctrl+C ourselves
    node = FollowMe()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    node.cmd.publish(Twist())  # stop the wheels before exiting
    node.destroy_node()
    rclpy.try_shutdown()


if __name__ == '__main__':
    main()
```

How it works

- The robot only looks at a 60-degree cone straight ahead (`SEE_ANGLE`, 30 degrees to each side) and up to 1 m away (`SEE_DISTANCE`). Whatever is closest in that cone is what it follows.
- The forward speed depends on the gap: too far, drive forward; too close, back up; at 40 cm (`KEEP_DISTANCE`), stand still. `twist.linear.x` is the forward speed.
- The turning speed steers toward the object, as in Program 1.
- If nothing is in the cone, the robot stops.

Try it on the robot

1. Bring up the robot as in Program 1 and put it on the floor in an open space.

2. Stand about 60 cm in front of it and run `python3 /root/code/follow_me.py`.
3. Step back slowly. The robot follows your feet. Step toward it and it backs up. Here's the output with my foot in front of the robot, moving it around:

```
following: 0.66 m at   25 degrees
following: 0.55 m at   -2 degrees
following: 0.57 m at   30 degrees
following: 0.37 m at   -4 degrees
following: 0.29 m at    0 degrees
```



The robot following my foot.

4. Press **Ctrl+C** to stop.

> **WARNING**
> Keep the area clear, start slowly and keep a hand near **Ctrl+C**. The robot follows whatever is closest in front of it, which may be a chair leg rather than you.

Make it yours

Every setting is a named value at the top of each program. Change one, save the file and run the program again:

- **Face the closest object:** set `MAX_DISTANCE = 0.5` to react to objects up to 50 cm away.

- **Follow me:** set `KEEP_DISTANCE = 0.6` to stay further back, or `SEE_ANGLE = math.radians(15)` for a narrower view that's less likely to pick up a chair leg.
- **Speed:** lower `TURN_SPEED`, `MAX_SPEED` or `MAX_TURN` for gentler moves.

Keep `MAX_SPEED` well below the robot's top speed of about 0.4 m/s.

Let an AI assistant drive your robot

An AI coding assistant that can run commands on your PC, such as Claude Code, can control the robot through the same container: look at the LiDAR scan, run your programs and send drive commands, all from plain-English requests. That's how the programs in this chapter were tested on a real robot.

1. Install an AI coding assistant that can run terminal commands on your PC.
2. Bring up the robot as in Chapter 3, so the `makerspet` container is running and the robot is connected.
3. Give the assistant the facts it needs and your safety rules in your first message:

```
My robot's software runs in a Docker container named makerspet, and the robot is connecte
d.
Run commands in it with: docker exec makerspet bash -lc "<command>"
The LiDAR scan is on /scan; drive commands go to /cmd_vel (geometry_msgs/Twist).
Scan angles are mirrored: the angle around the robot is 180 degrees minus the scan angle.
Rules: look before you move, keep speed under 0.15 m/s, limit every motion to 15 seconds,
and always finish by publishing a zero Twist to /cmd_vel.
First, tell me what the robot sees within 50 cm.
```

4. Then ask for what you want, for example:
    - "I placed a cup next to the robot. Where is it?"
    - "Copy face_closest.py into the container and run it for 15 seconds."
    - "I put my foot in front of the robot. Make it follow my foot for 10 seconds."

> **WARNING**
> AI assistants make mistakes. Review each command before you approve it, keep the robot on the floor and away from edges, and keep a hand near the power switch.

8. Using an Ubuntu PC Instead

Video: Arduino/ROS2 self driving robot - Ubuntu software setup and robot bring up

This chapter repeats Chapters 2 and 3 on a PC running **Ubuntu 22.04**. The differences: you give your user access to serial ports, install two Python pieces the ESP32 uploader needs, use Docker Engine instead of Docker Desktop, and share your X11 display with the container instead of running an X server. Steps that match Windows are kept short and point back to Chapter 3.

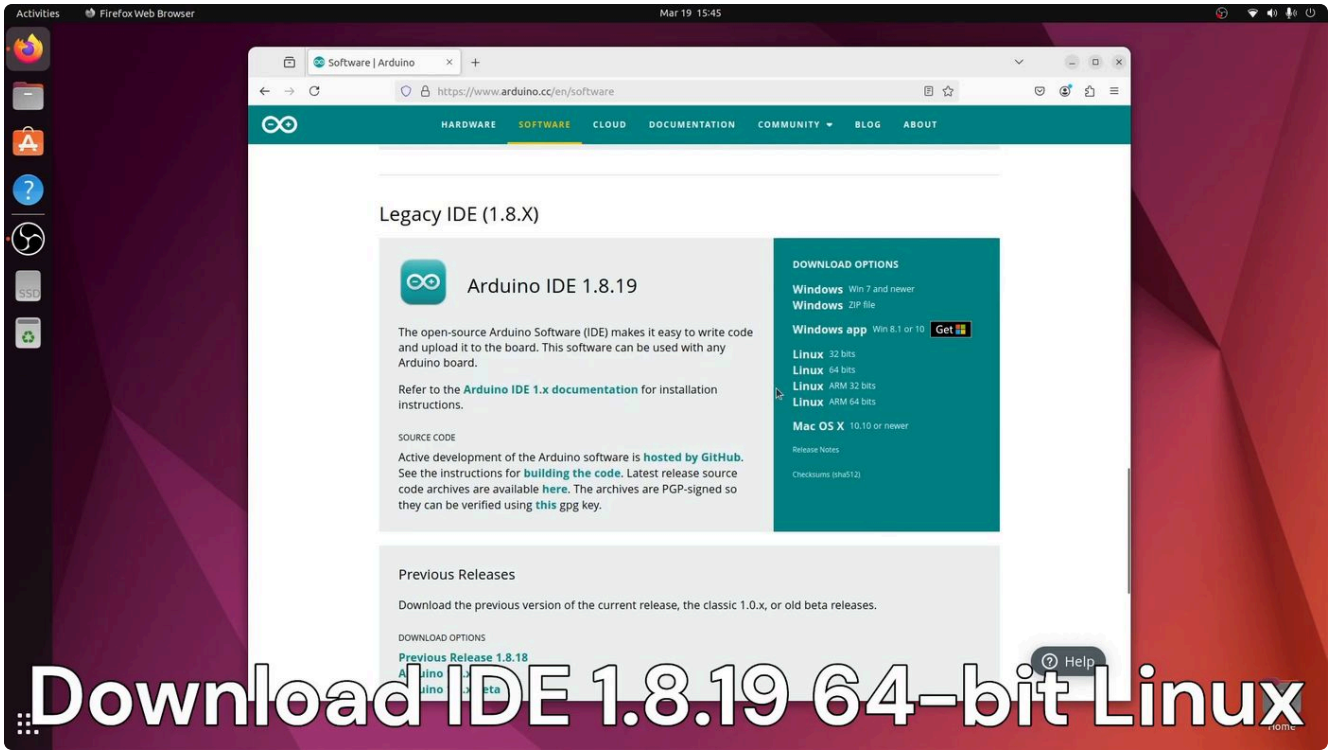
Allow about an hour, plus download time for the Docker image. You need the assembled robot, a USB cable for the ESP32 and a 2.4 GHz WiFi network.



Software install and bring-up on Ubuntu 22.04.

Install the Arduino IDE 1.8.19

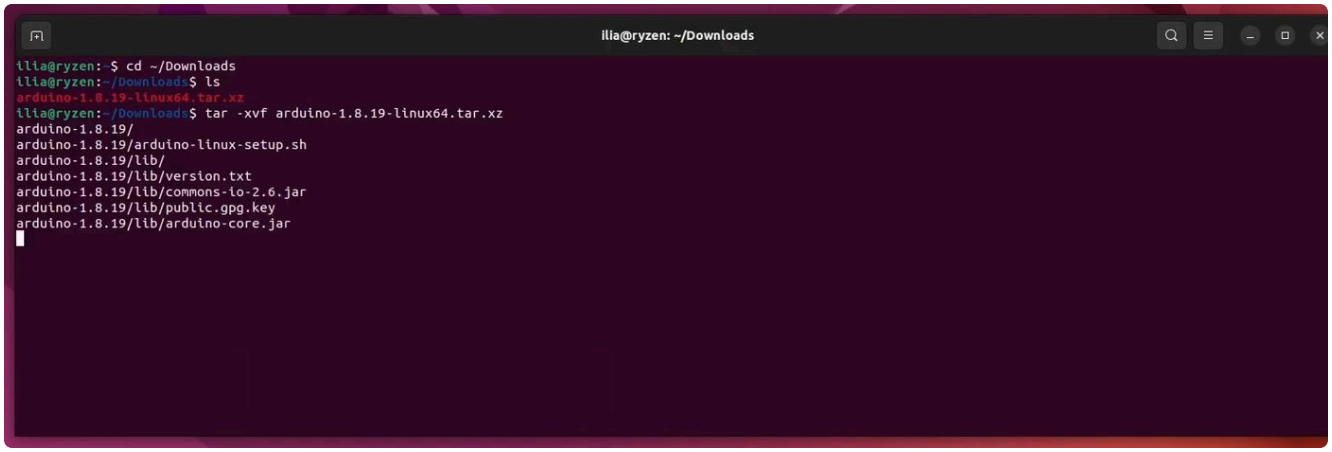
1. On arduino.cc, click **Software**, scroll to **Legacy IDE (1.8.X)** and download **Arduino IDE 1.8.19** for **Linux 64 bits**. (**Just download** skips the donation.)



Download the Legacy IDE 1.8.19 for 64-bit Linux.

2. Open a terminal: click the **Apps** grid, type `terminal`, right-click the Terminal icon and choose **Add to Favorites**.
3. Unpack the archive (press **Tab** to complete file names):

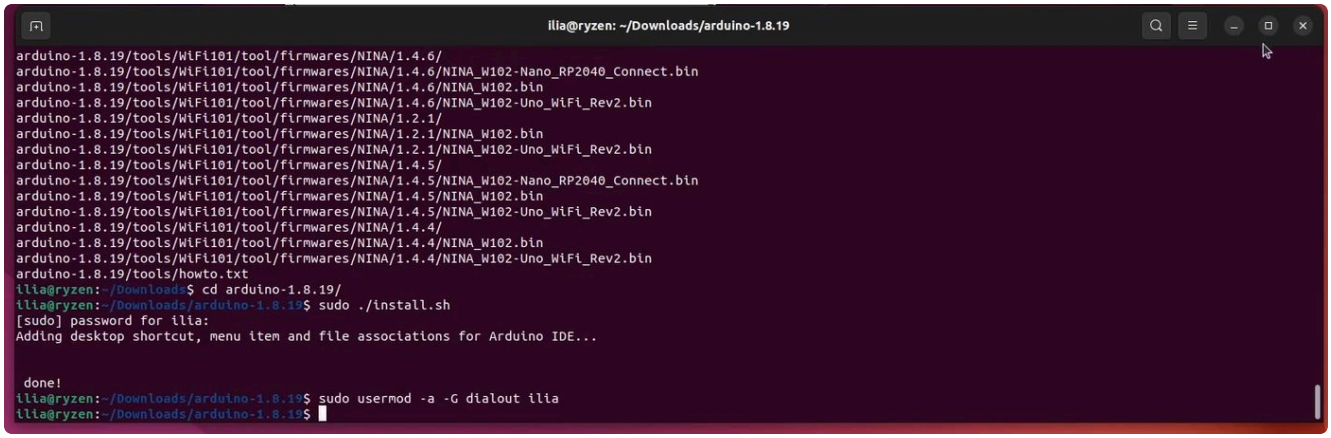
```
cd ~/Downloads
ls
tar -xvf arduino-1.8.19-linux64.tar.xz
```



Unpacking the Arduino archive.

4. Run the install script. It asks for your password because it installs system-wide:

```
cd arduino-1.8.19/
sudo ./install.sh
```



The install script finishes with "done!".

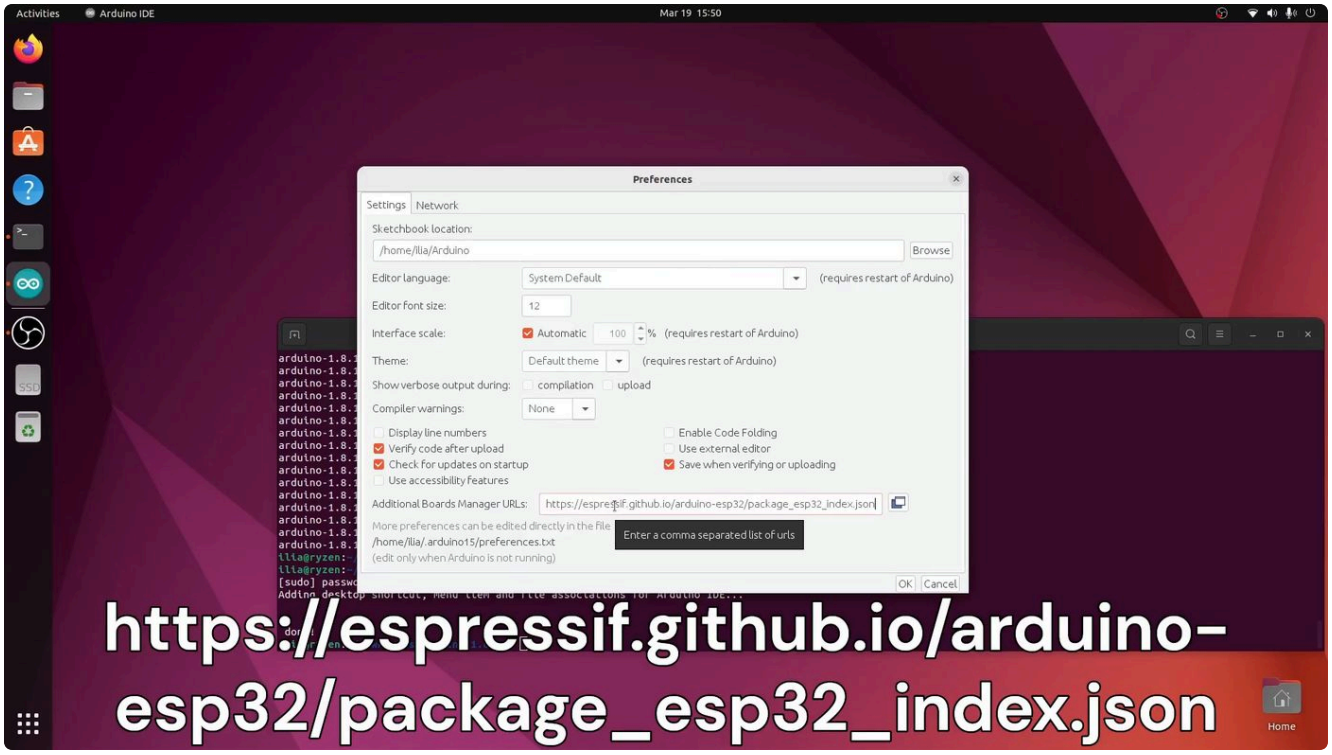
5. In **Apps**, add **Arduino IDE** to Favorites and launch it.

Add the ESP32 toolchain

Same as Chapter 3.

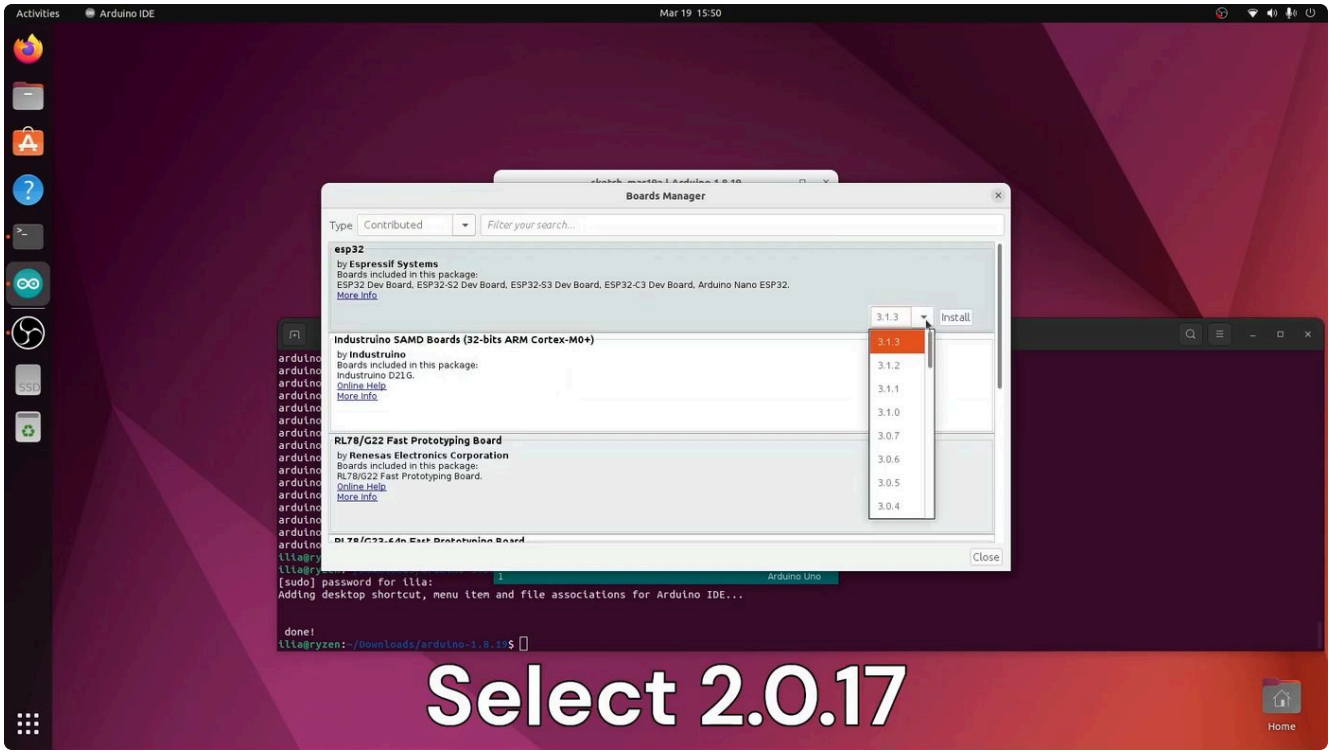
1. In **File → Preferences**, paste this into **Additional Boards Manager URLs**:

```
https://espressif.github.io/arduino-esp32/package_esp32_index.json
```



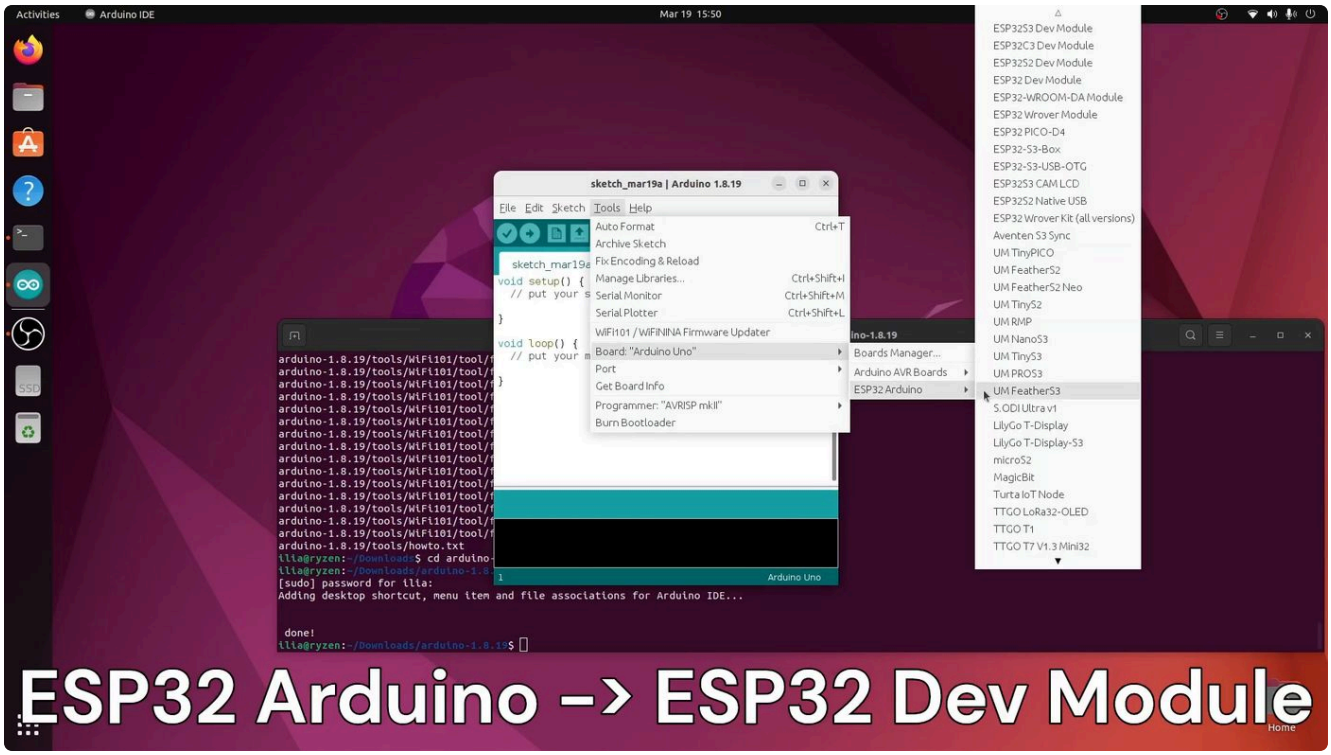
Adding the Espressif boards URL.

2. In **Tools → Board → Boards Manager**, set **Type** to **Contributed**, find **esp32 by Espressif Systems**, select **2.0.17** and click **Install.**



Install esp32 version 2.0.17.

3. Select **Tools → Board → ESP32 Arduino → ESP32 Dev Module.**



Choose ESP32 Dev Module.

> **SINCE THE VIDEO WAS RECORDED**
>
> Stay on ESP32 board package **2.0.17**. In 2.0.15 HardwareSerial is broken: the firmware compiles but the LiDAR returns no data. 3.x removed `ledcAttachPin` / `ledcSetup`, which the firmware needs.

Give yourself access to serial ports and install pyserial

On Ubuntu, only members of the `dialout` group can open serial ports, and the ESP32 uploader (`esptool.py`) needs two Python fixes.

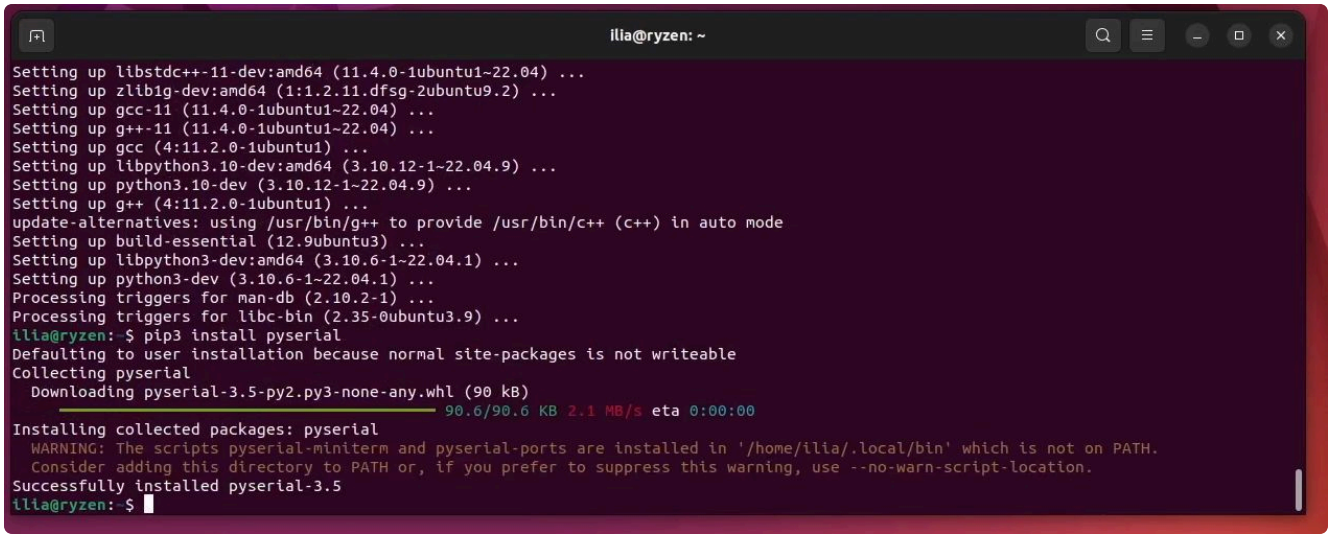
1. Add your user to `dialout` (in the video, `YOUR_USERNAME` is `ilia`):

```
sudo usermod -a -G dialout YOUR_USERNAME
```

2. **Reboot your PC** so the group membership takes effect.

3. Install pip and the Python serial library:

```
sudo apt install -y python3-pip
pip3 install pyserial
```



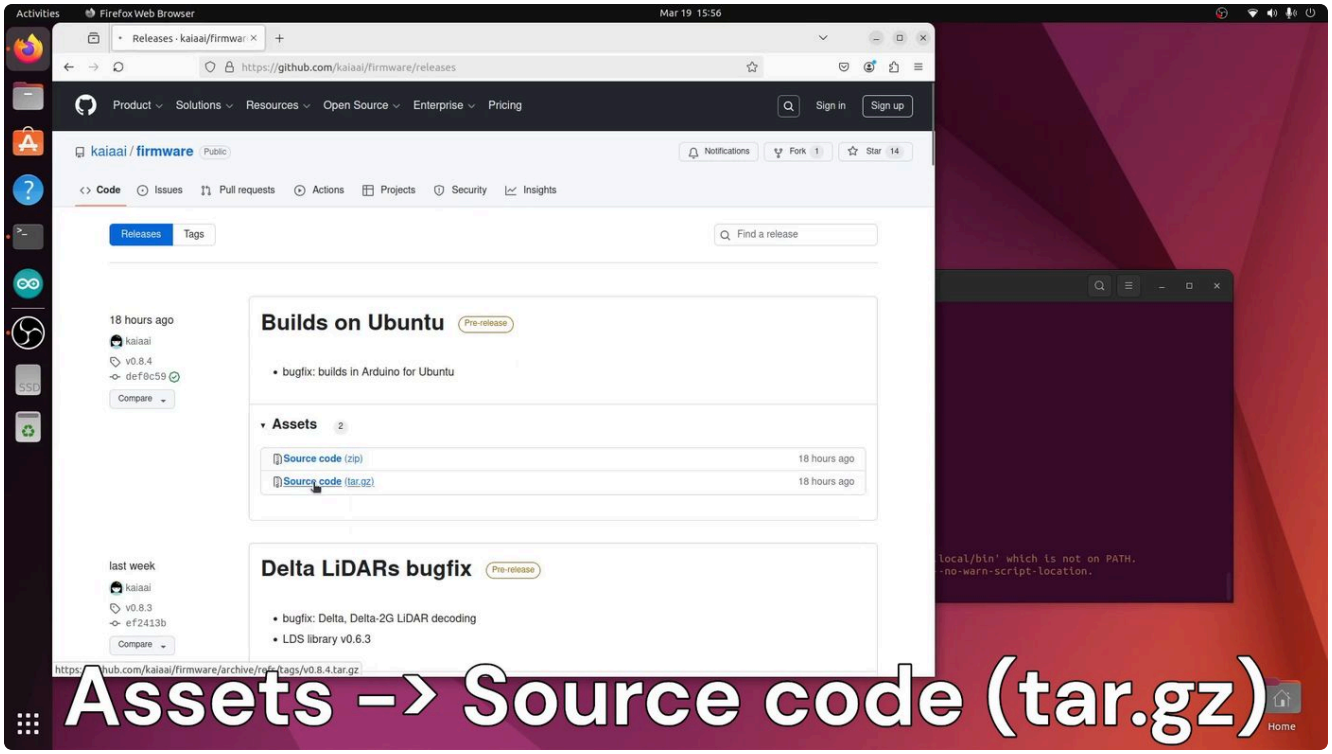
pyserial installed.

Ignore the warning that `pyserial-miniterm` and `pyserial-ports` are not on your PATH; the Arduino IDE doesn't use them.

> **UPLOAD FAILS WITH \"NO MODULE NAMED 'SERIAL'\"**
> Without pyserial, the upload ends with `ModuleNotFoundError: No module named 'serial'` and `Error compiling for board ESP32 Dev Module.` Run the two commands above and upload again.

Download the firmware and copy it into the sketchbook

1. On github.com/kaiaai/firmware, click **Releases**, expand **Assets** under the latest release and download **Source code (tar.gz)**.



Download the latest firmware release.

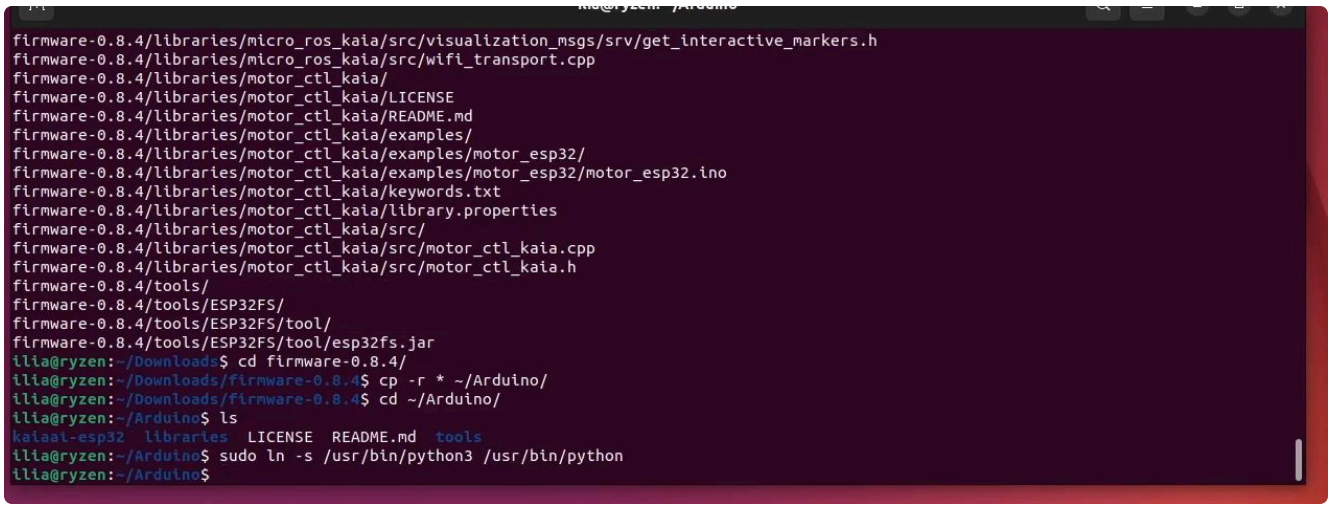
2. Unpack it and copy its contents into your sketchbook, `~/Arduino`. The video uses release 0.8.4; use your file's version number:

```
cd ~/Downloads
ls
tar -xvf firmware-0.8.4.tar.gz
cd firmware-0.8.4/
cp -r * ~/Arduino/
cd ~/Arduino/
ls
```

`ls` should list `kaiaai-esp32` (the sketch), `libraries`, `LICENSE`, `README.md` and `tools`. `tools` contains the **ESP32FS** plugin for **ESP32 Sketch Data Upload**, so unlike on Windows you don't install a separate SPIFFS plugin.

3. The ESP32 upload tools call `python`, but Ubuntu 22.04 only has `python3`. Link them:

```
sudo ln -s /usr/bin/python3 /usr/bin/python
```

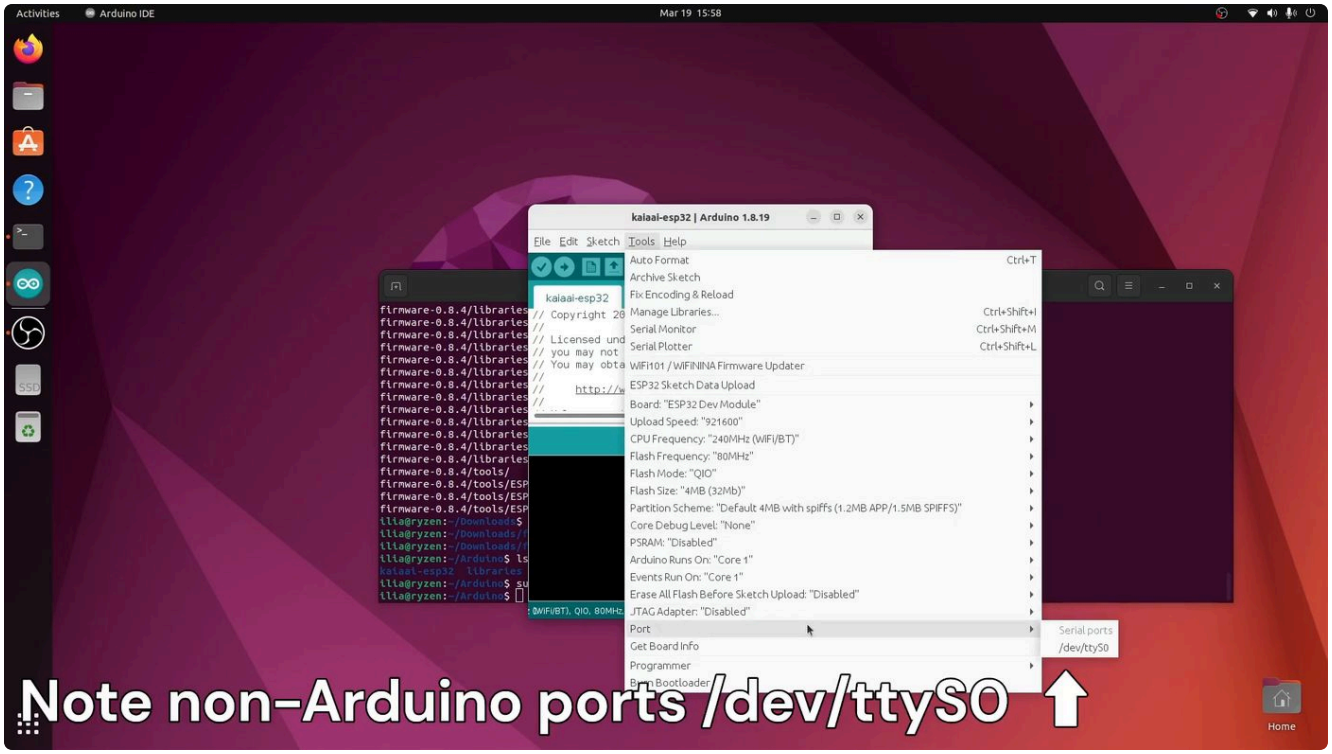


Firmware copied and the python link created.

> **TIP**
>
> `ln: failed to create symbolic link '/usr/bin/python': File exists` means the link is already there (for example, from `python-is-python3`); skip this step.

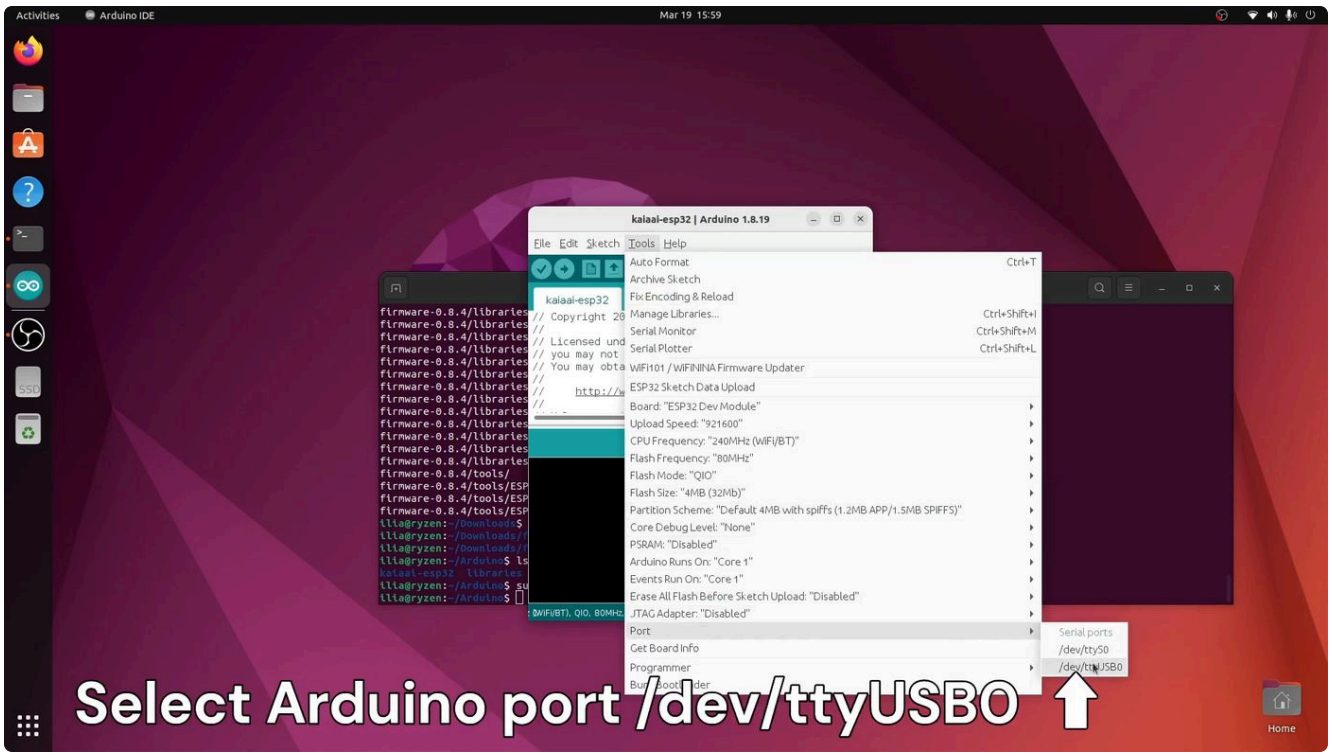
Upload the firmware

1. Restart the Arduino IDE so it picks up the new libraries and ESP32FS.
2. Open `~/Arduino/kaiaai-esp32/kaiaai-esp32.ino` (**File → Open**).
3. **Before** plugging in the robot, note the ports in **Tools → Port**. On my PC there is only `/dev/ttyS0`, a built-in serial port.



Before plugging in, only /dev/ttyS0 is listed.

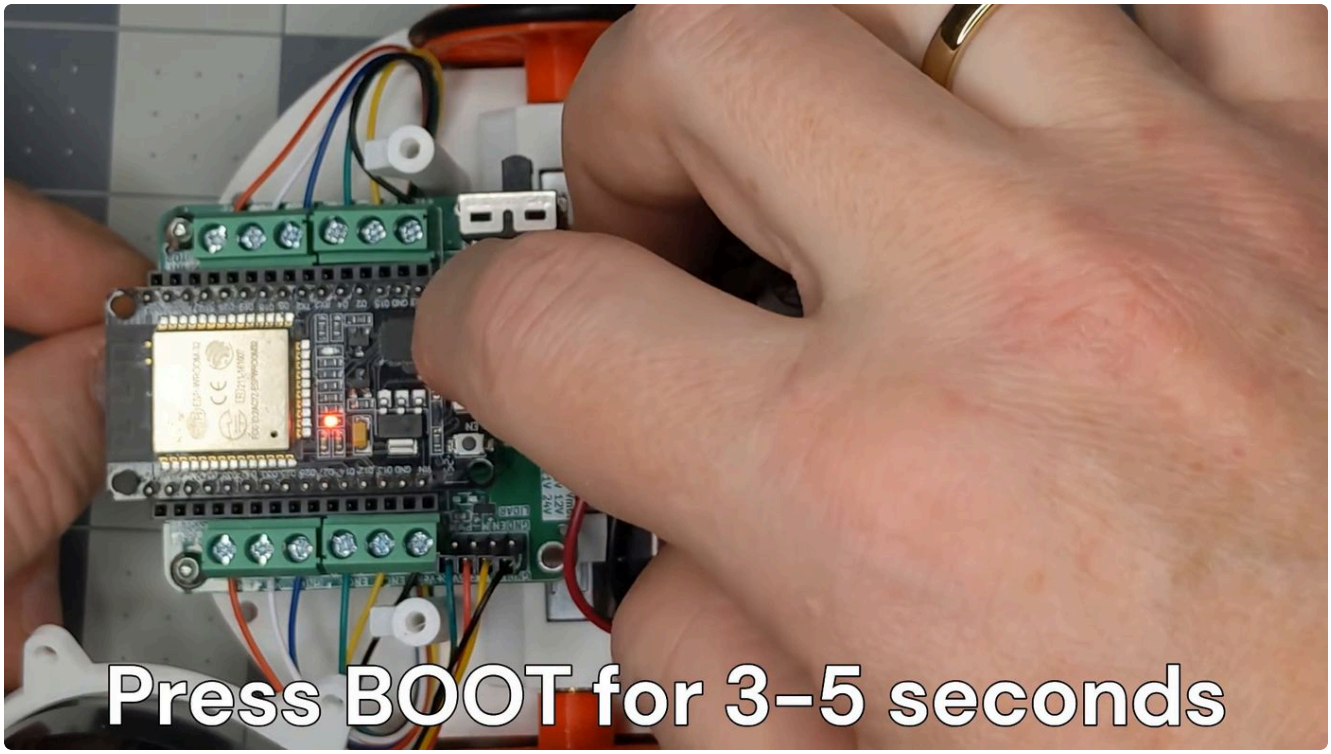
4. Plug in the ESP32 with the robot's power switch **OFF**.
5. In **Tools** → **Port**, select the new port, usually `/dev/ttyUSB0`.



Select /dev/ttyUSB0.

!!! tip Or run `ls /dev/ttyUSB*` before and after plugging in; the new entry is your robot.

6. Click **Upload**. When the output shows `Connecting....`, hold **BOOT** for 3–5 seconds.

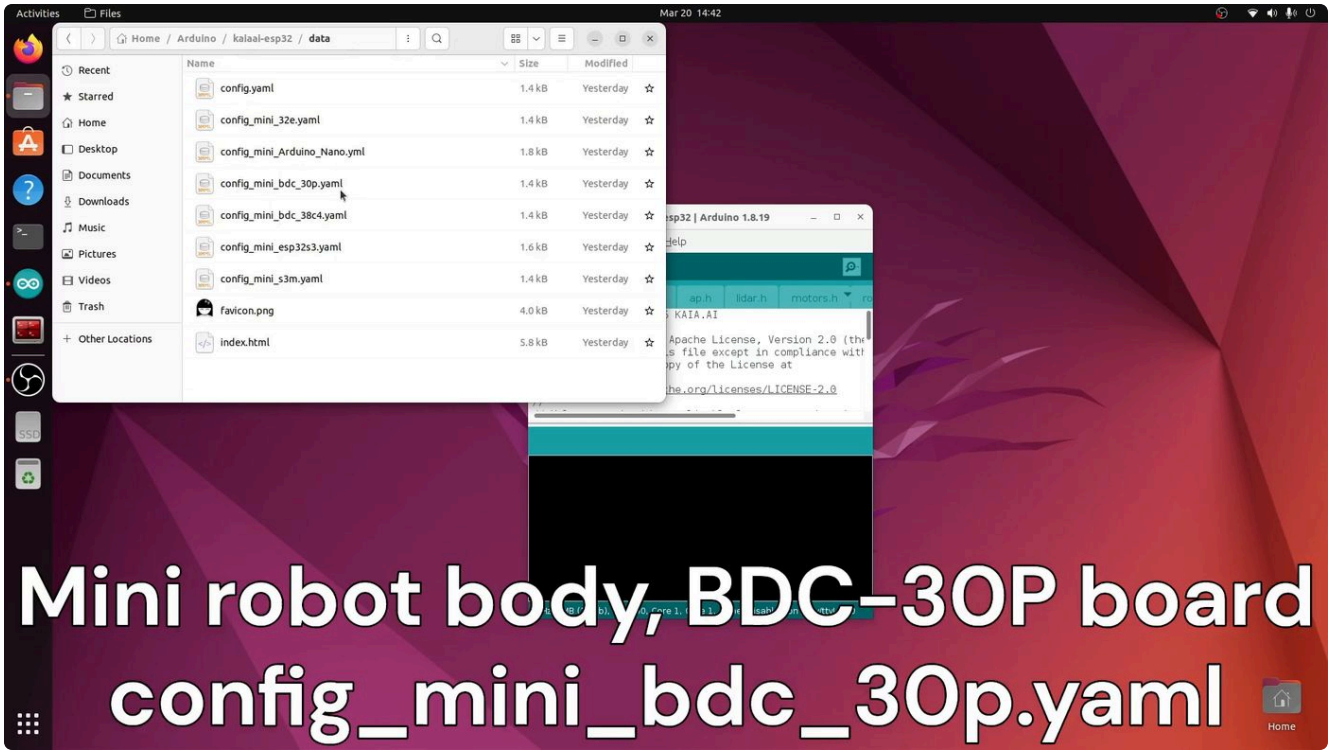


Hold BOOT for 3–5 seconds while the IDE connects.

Upload the configuration data

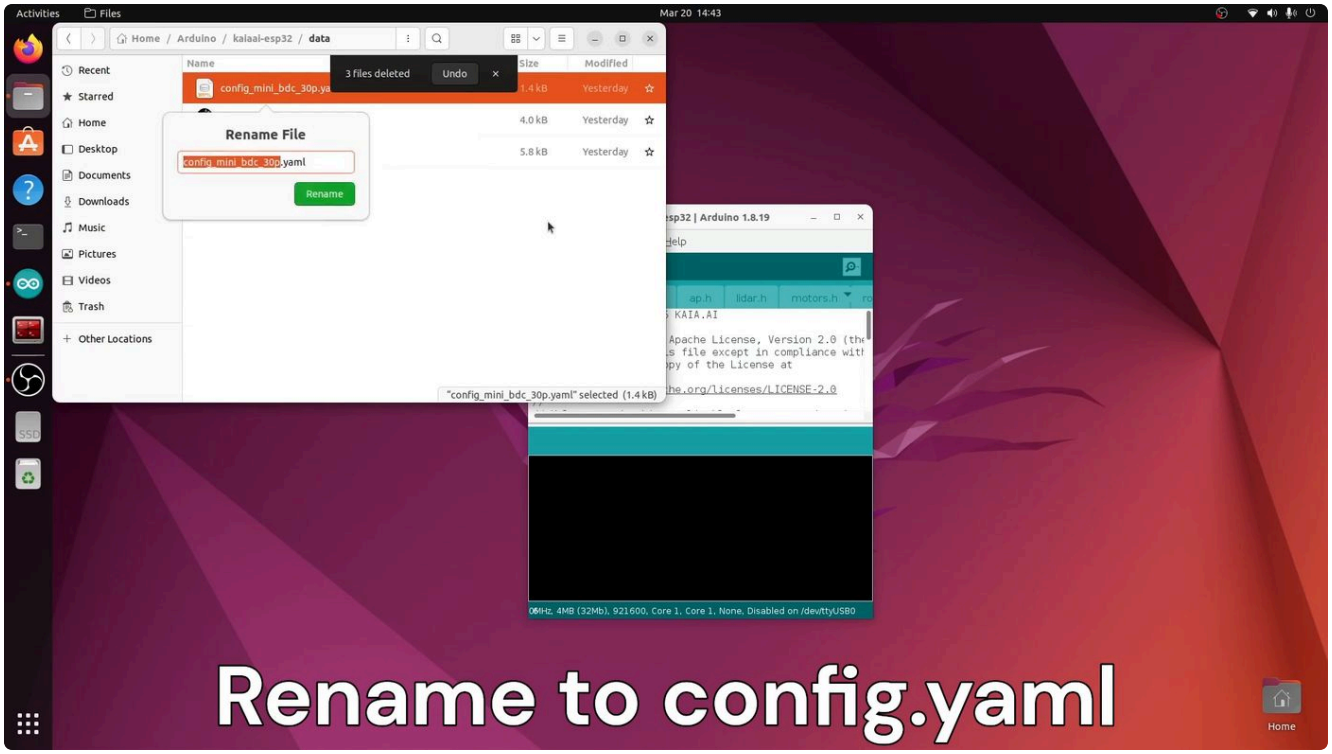
The sketch's `data` folder holds YAML files describing the board's ESP32 pins and the robot's properties (base and wheel diameter, wheel spacing, motor type, encoder pulses and more).

1. Press **Ctrl+K** (**Sketch → Show Sketch Folder**) and open the `data` folder.
2. Keep the YAML file for your board and robot and delete the other `config*.yaml` files. For the BLD-120MM-PACK (Mini body, BDC-30P board), keep `config_mini_bdc_30p.yaml`.



Keep config_mini_bdc_30p.yaml.

3. Rename it to `config.yaml`.



Rename it to config.yaml.

4. Unplug the ESP32 and plug it back in.

5. Choose **Tools → ESP32 Sketch Data Upload** and hold **BOOT** for 3–5 seconds as it connects.



Tools → ESP32 Sketch Data Upload.

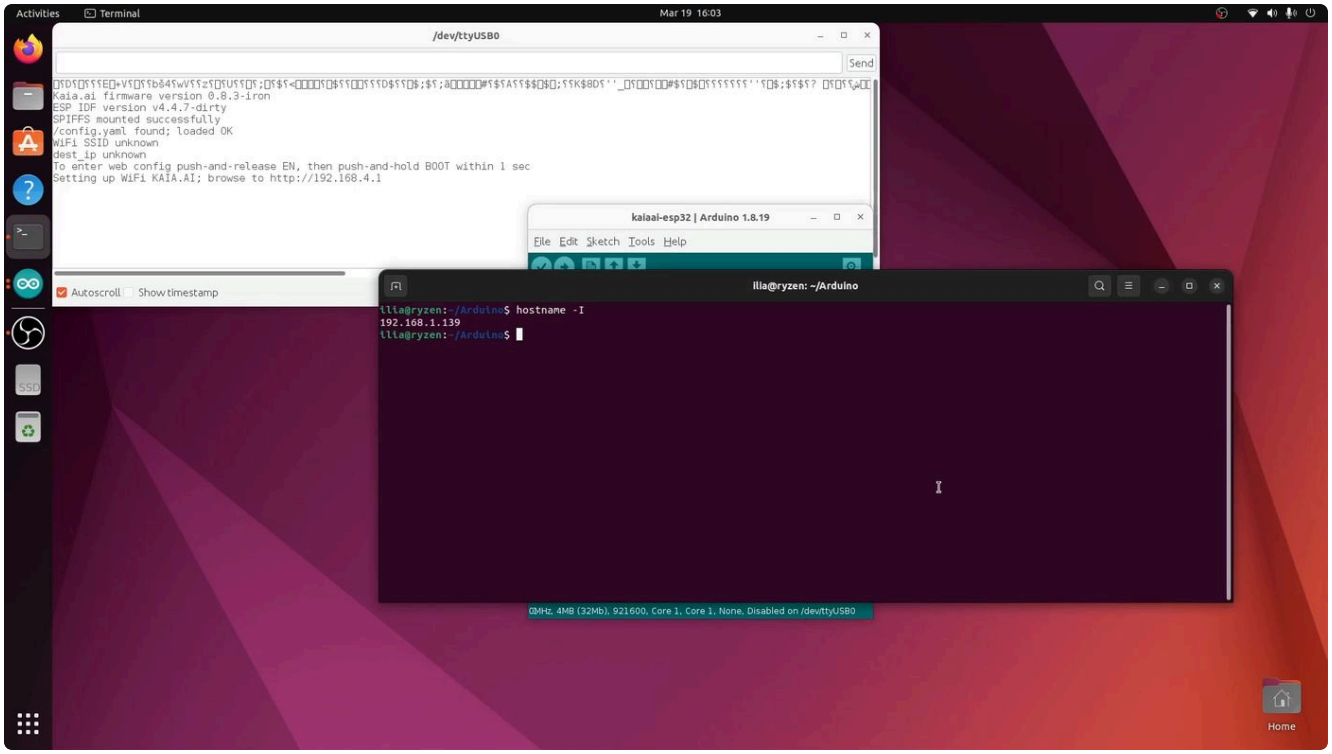
6. When it finishes, unplug and replug the ESP32.

> **SPIFFS UPLOAD FAILS ON UBUNTU**
> The Ubuntu SPIFFS plugin sometimes builds the image (`[SPIFFS] upload : /tmp/.../kaiaai-esp32.spiffs.bin`) and then prints `SPIFFS Upload failed!` without trying the upload. If so, upload the data once from a Windows PC (Chapter 3), then keep using Ubuntu.

Configure the robot's WiFi

1. Open **Tools → Serial Monitor** at **115200** baud. You should see the firmware version, `SPIFFS mounted successfully` , `/config.yaml found; loaded OK` and `Setting up WiFi KAIA.AI; browse to http://192.168.4.1` .
2. Copy your PC's IP address (note the capital `I`):

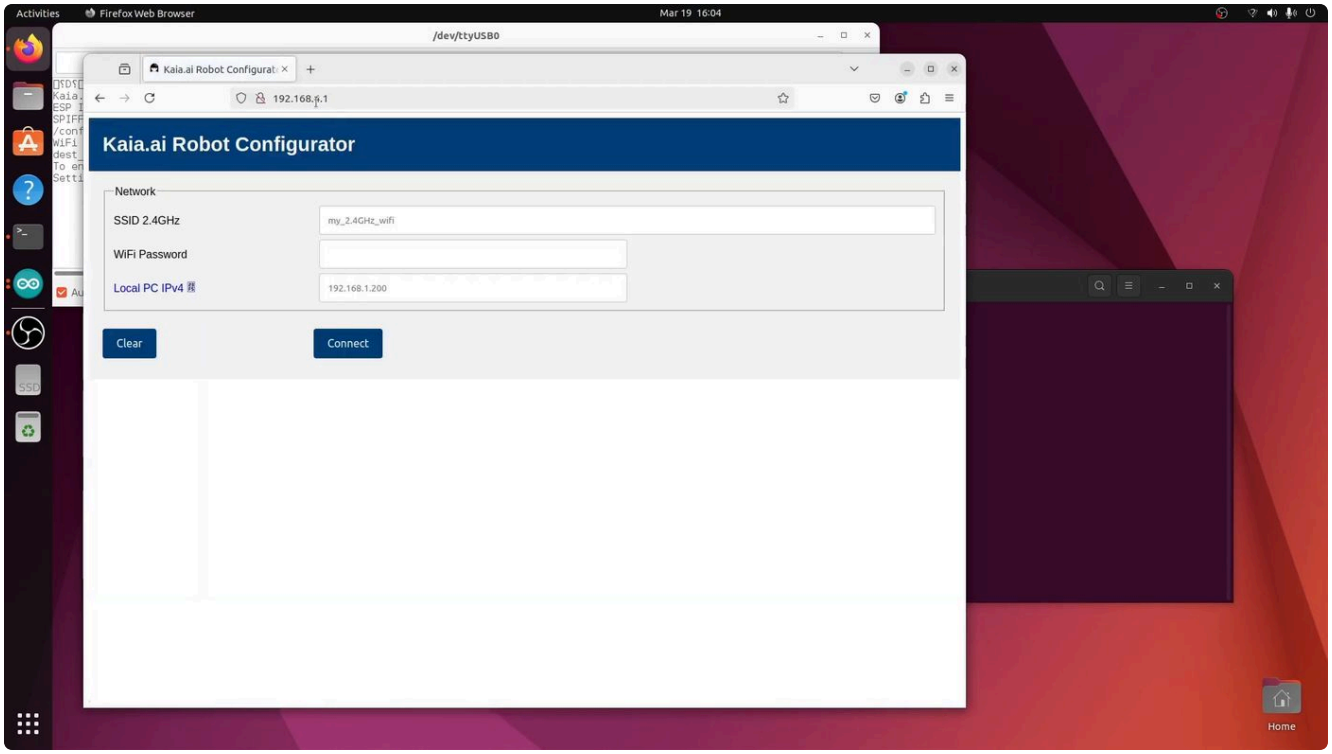
```
hostname -I
```



WiFi setup ready (top); the PC's IP address (bottom).

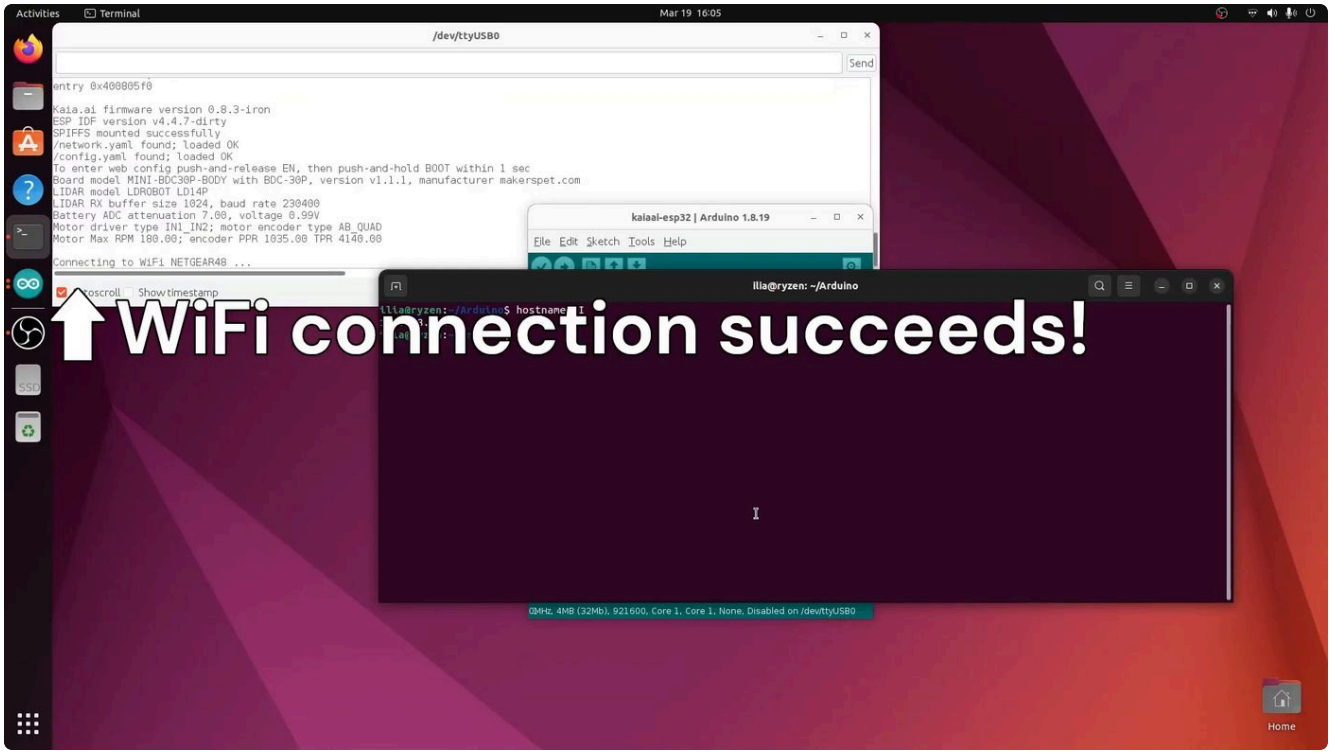
!!! note If it prints several addresses (one may be Docker's), use your home network's, such as `192.168.1.x`.

3. Connect your PC to the **KAIA.AI** WiFi network.
4. Browse to `http://192.168.4.1`, enter your 2.4 GHz **SSID 2.4GHz** and **WiFi Password**, paste your PC's IP into **Local PC IPv4**, and click **Connect**.



The Kaia.ai Robot Configurator page.

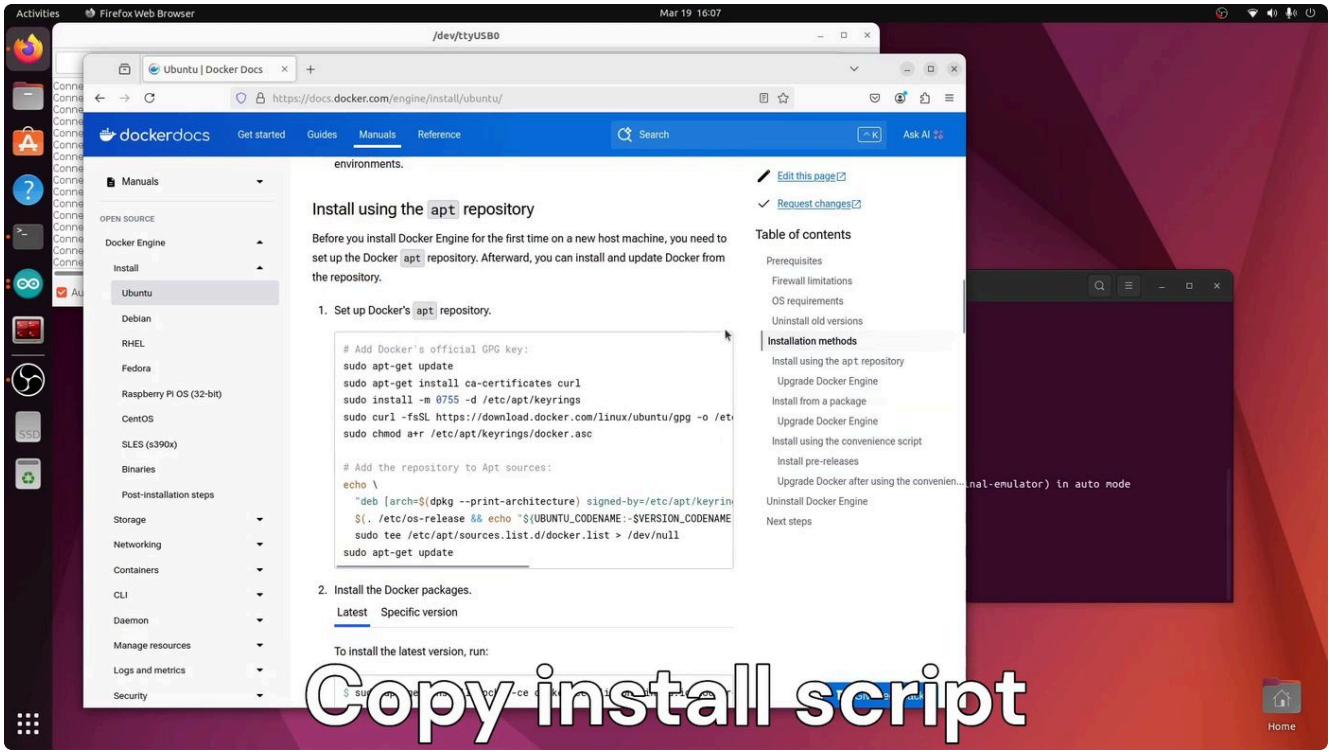
5. Reconnect your PC to your local WiFi.
6. In the Serial Monitor, the robot prints `Connecting to WiFi ...`; if it stays stuck, press **EN** (reset). Once connected, it loads `/network.yaml`, prints the board and LiDAR model, and repeats `Connecting to Micro-ROS agent 192.168.1.139 ...` with your PC's IP. That's expected: ROS2 isn't running yet.



The robot joins your WiFi network.

Install Docker Engine

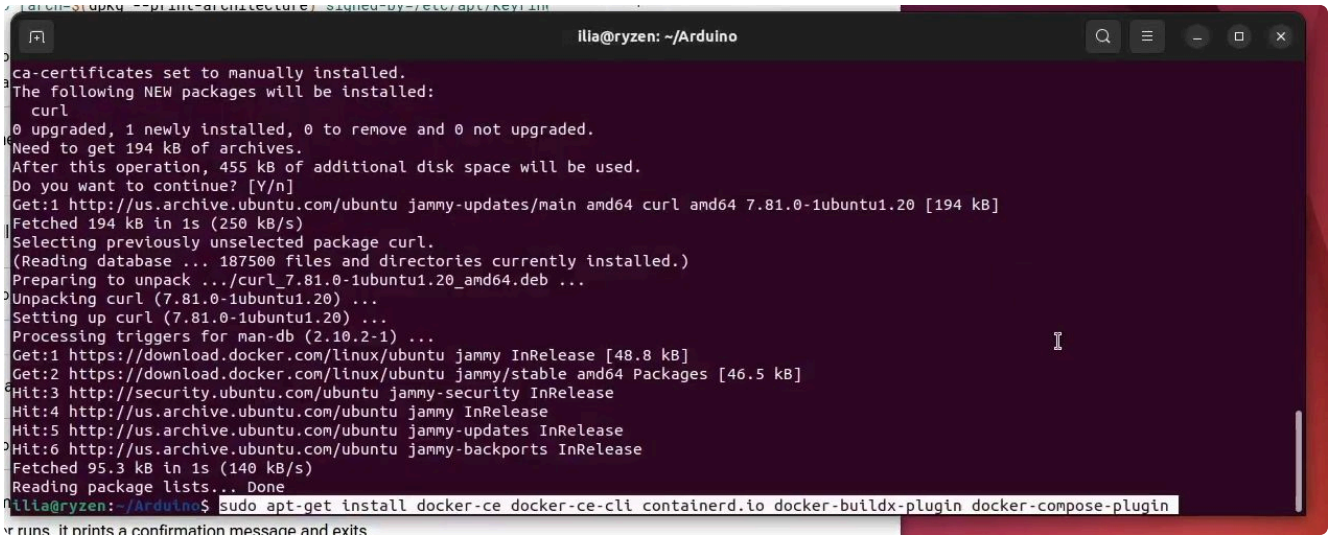
1. Open Docker's Install Docker Engine on Ubuntu page.
2. Under **Install using the apt repository**, copy step 1 (*Set up Docker's apt repository*) and run it in your terminal. Copy it from Docker's page, since they update it.



Copy step 1 from Docker's page.

3. Run step 2 (*Install the Docker packages*). At the time of the video it was:

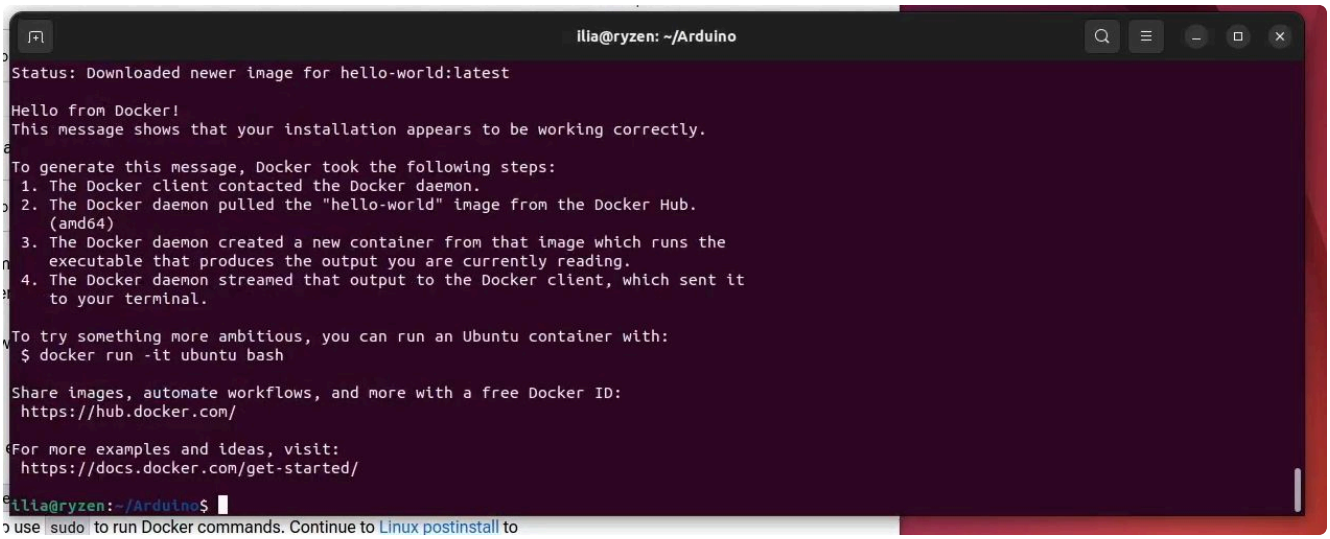
```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-com
pose-plugin
```



Installing the Docker packages.

4. Test it; you should see **Hello from Docker!**:

```
sudo docker run hello-world
```



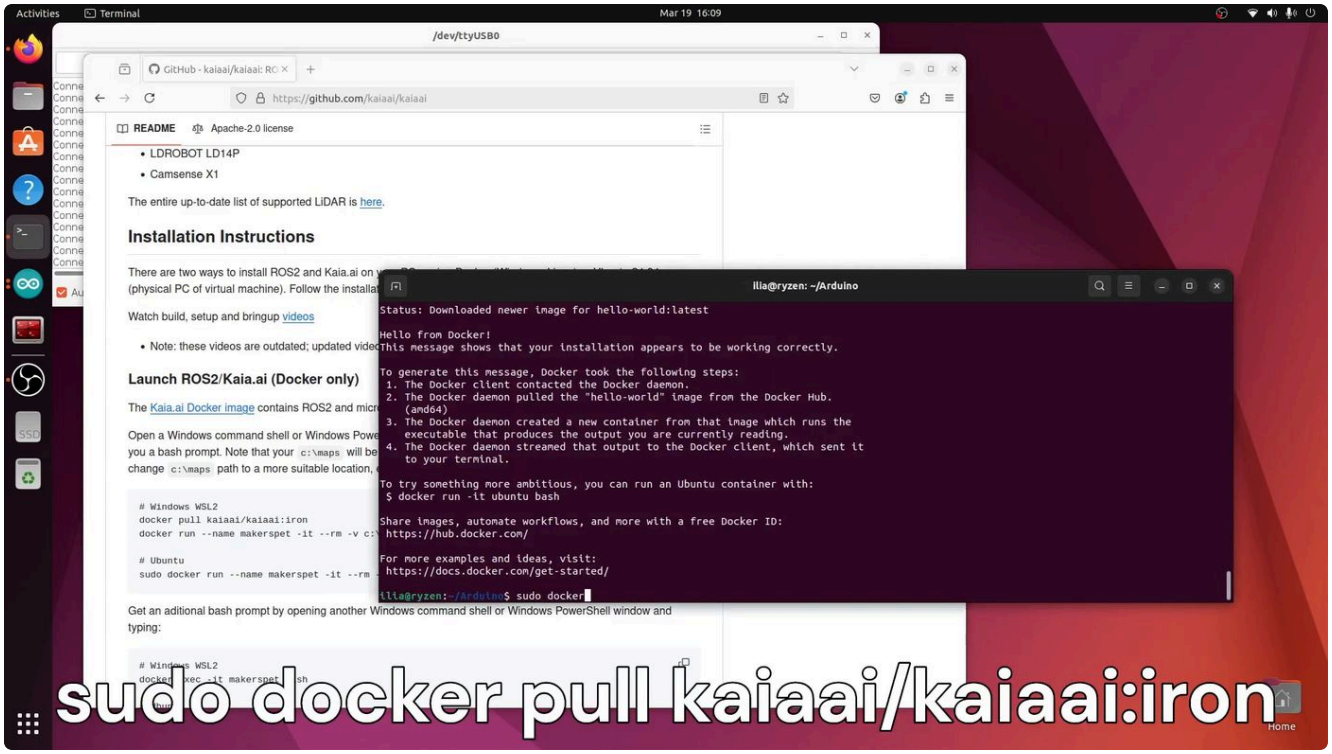
Docker installed.

> **NOTE**
> Every `docker` command here starts with `sudo`. To drop it, see Docker's Linux post-installation steps.

Download the robot software image

1. Keep github.com/kaiaai/kaiaai open at **Launch ROS2/Kaia.ai (Docker only)**; its `# Ubuntu` lines are the commands below.
2. Pull the image (several gigabytes):

```
sudo docker pull kaiaai/kaiaai:iron
```



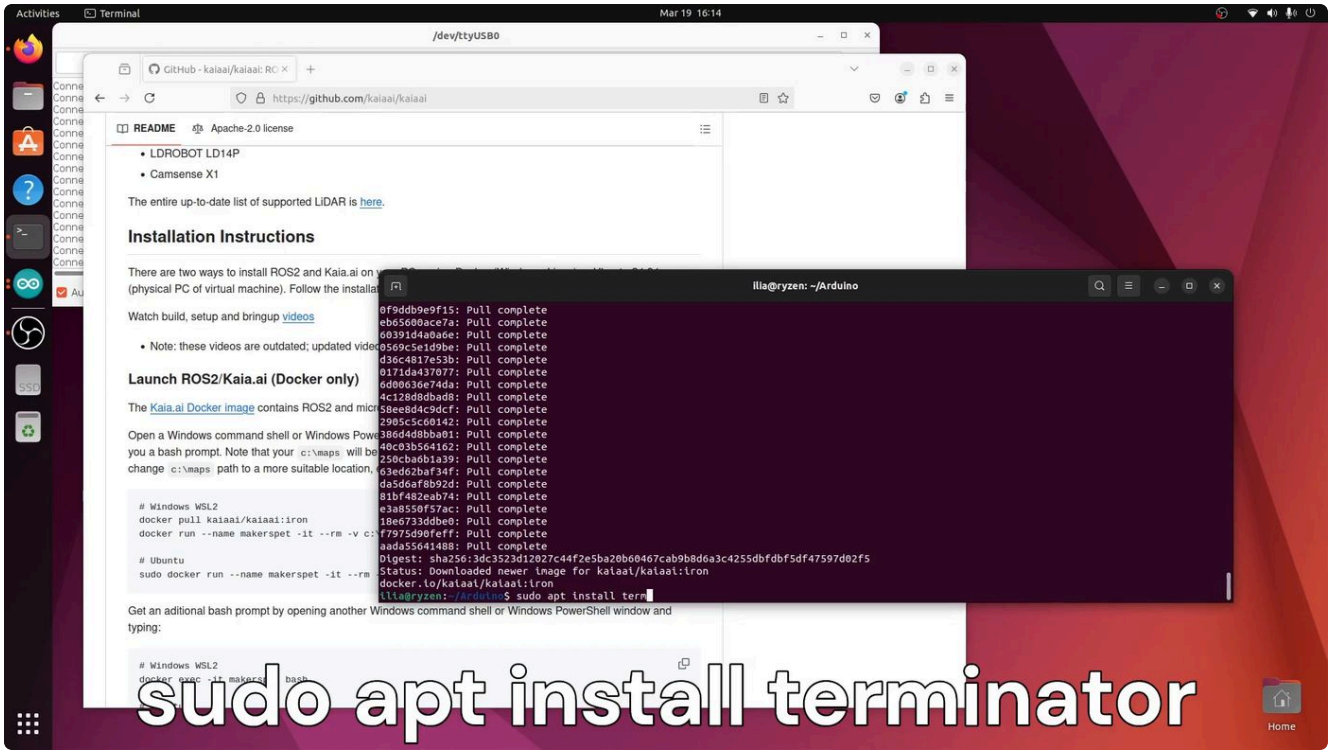
Pulling kaiaai/kaiaai:iron.

Install Terminator

Terminator splits one window into the three or four shells you'll need.

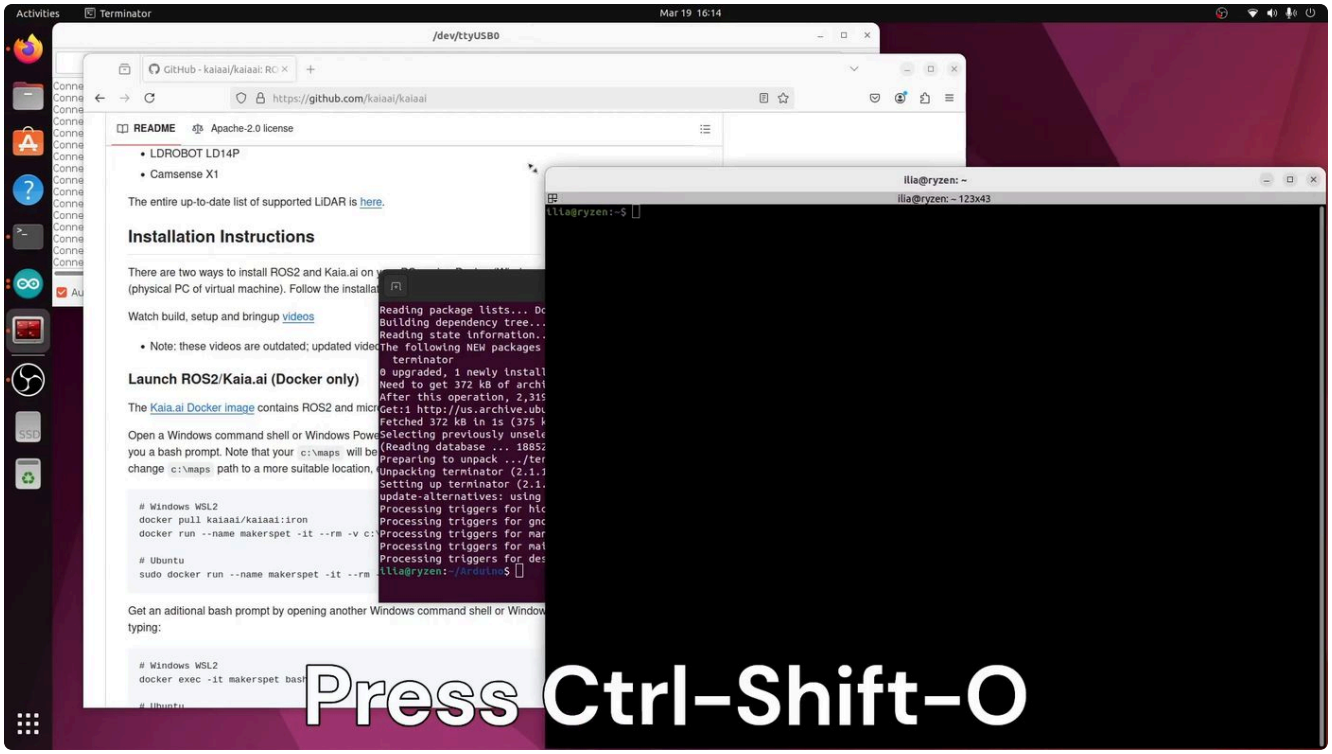
1. Install it:

```
sudo apt install terminator
```



Installing Terminator.

2. Add **Terminator** to Favorites and launch it.
3. Press **Ctrl+Shift+O** (or right-click, **Split Horizontally**) to split it into two panes.



Ctrl+Shift+O splits the window.

Launch the robot software

1. In the **top** pane, start the container (the Ubuntu version of Chapter 3's `docker run`):

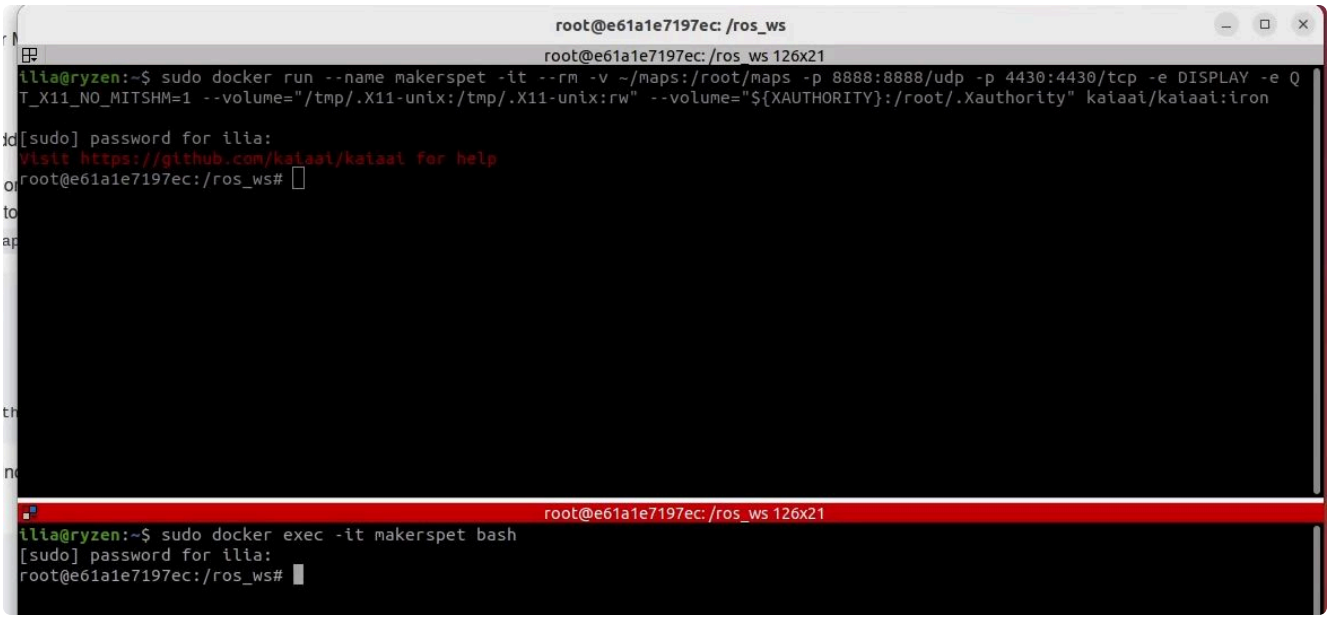
```
sudo docker run --name makerspet -it --rm -v ~/maps:/root/maps -p 8888:8888/udp -p 4430:44
30/tcp -e DISPLAY -e QT_X11_NO_MITSHM=1 --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" --volu
me="${XAUTHORITY}:/root/.Xauthority" kaiaai/kaiaai:iron
```

Compared with Windows:

- `-v ~/maps:/root/maps` stores maps in `~/maps` instead of `c:\maps`.
- `-p 8888:8888/udp` is the robot's micro-ROS agent port; `-p 4430:4430/tcp` is the same extra port as on Windows.
- `-e DISPLAY`, `-e QT_X11_NO_MITSHM=1` and the `/tmp/.X11-unix` and `${XAUTHORITY}` volumes let RViz draw on your desktop, so no X server is needed.

2. In the **bottom** pane, open a second shell in the container:

```
sudo docker exec -it makerspet bash
```



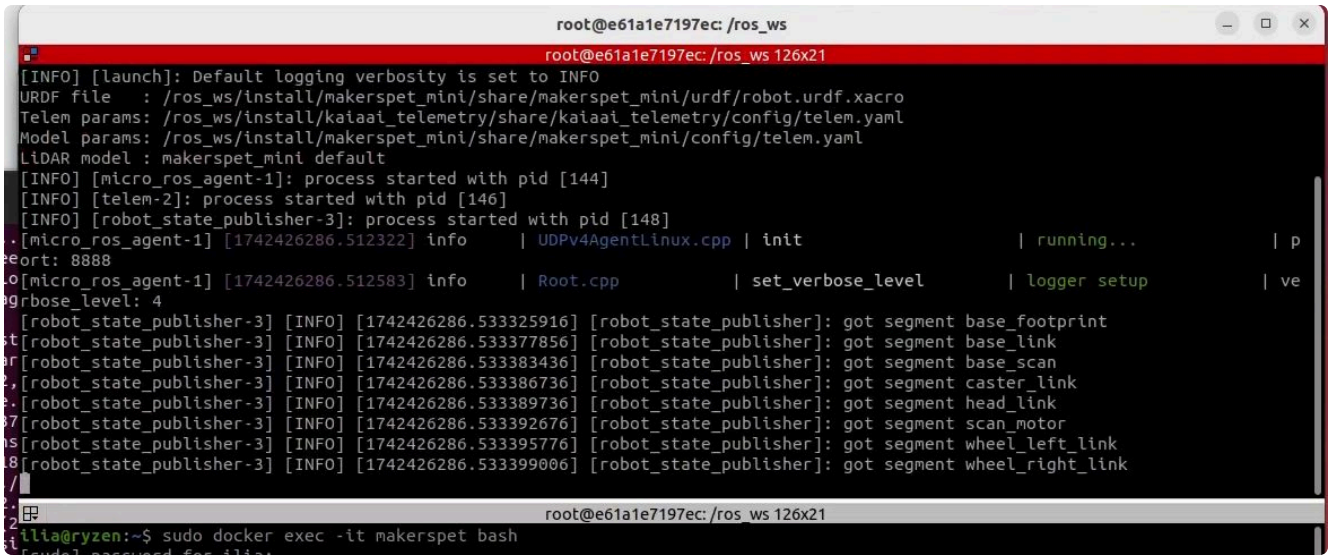
Both panes at the container prompt, root@...:/ros_ws#.

Bring up the robot

From here on, everything works as in Chapter 3.

1. In the top pane, start the bring-up:

```
ros2 launch kaiaai_bringup physical.launch.py
```



The micro-ROS agent starts on port 8888.

2. Unplug the USB cable, mount the LiDAR if you removed it, and turn the robot **ON**.
3. The robot connects, the top pane prints telemetry and the LiDAR spins.



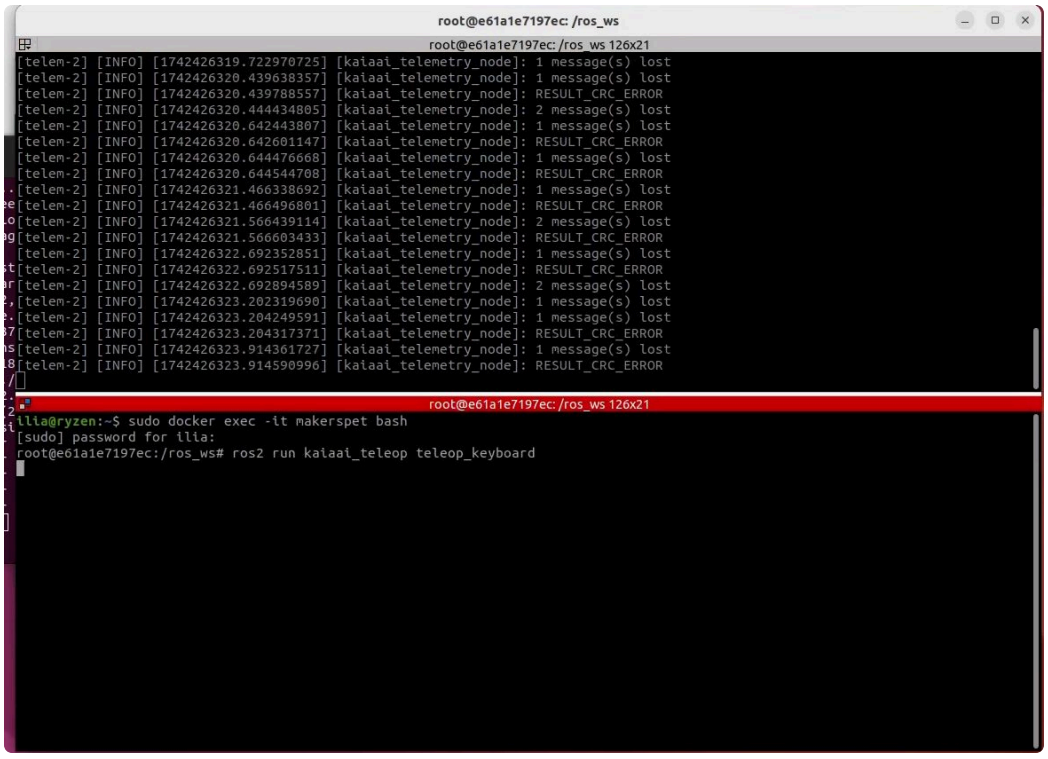
The LiDAR spins once the robot connects.

!!! note Occasional `1 message(s) lost` and `RESULT_CRC_ERROR` lines are lost WiFi packets and harmless. If they scroll constantly or RViz stalls, see "Frequent CRC errors" in Appendix B.

4. In the bottom pane, start keyboard teleoperation:

```
ros2 run kaiaai_teleop teleop_keyboard
```

w drives forward, **x** backs up, **a** and **d** turn left and right, **space** stops.



Teleop in the bottom pane.

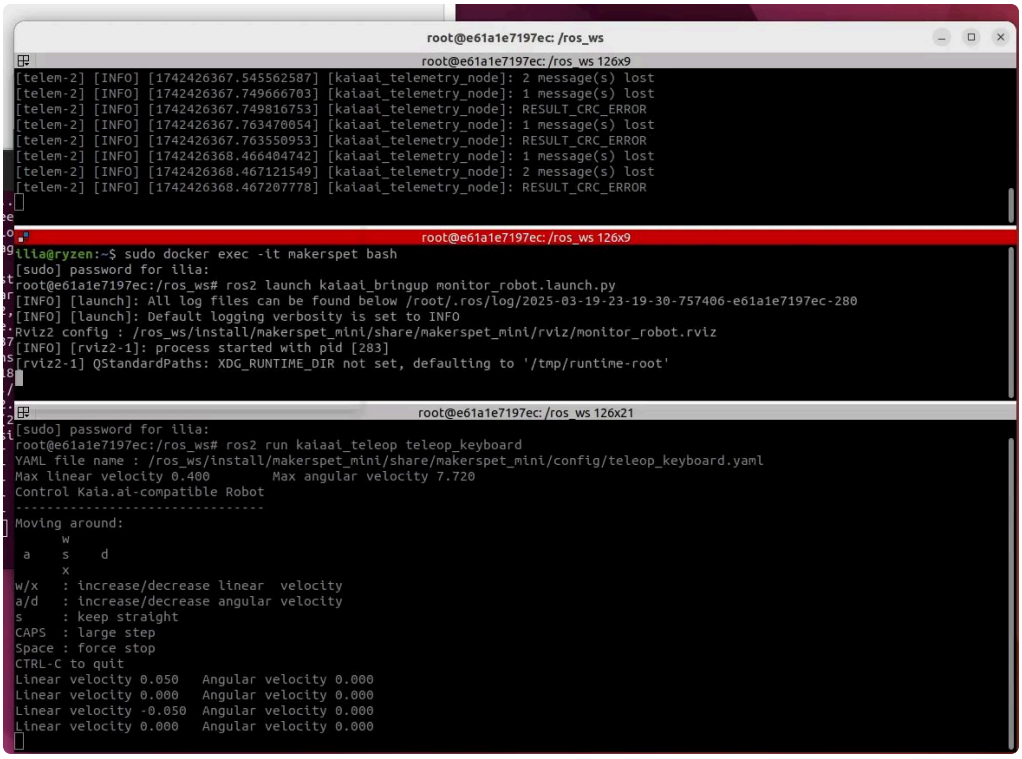
Visualize the robot

1. Right-click the top pane, choose **Split Horizontally**, and open another container shell:

```
sudo docker exec -it makerspet bash
```

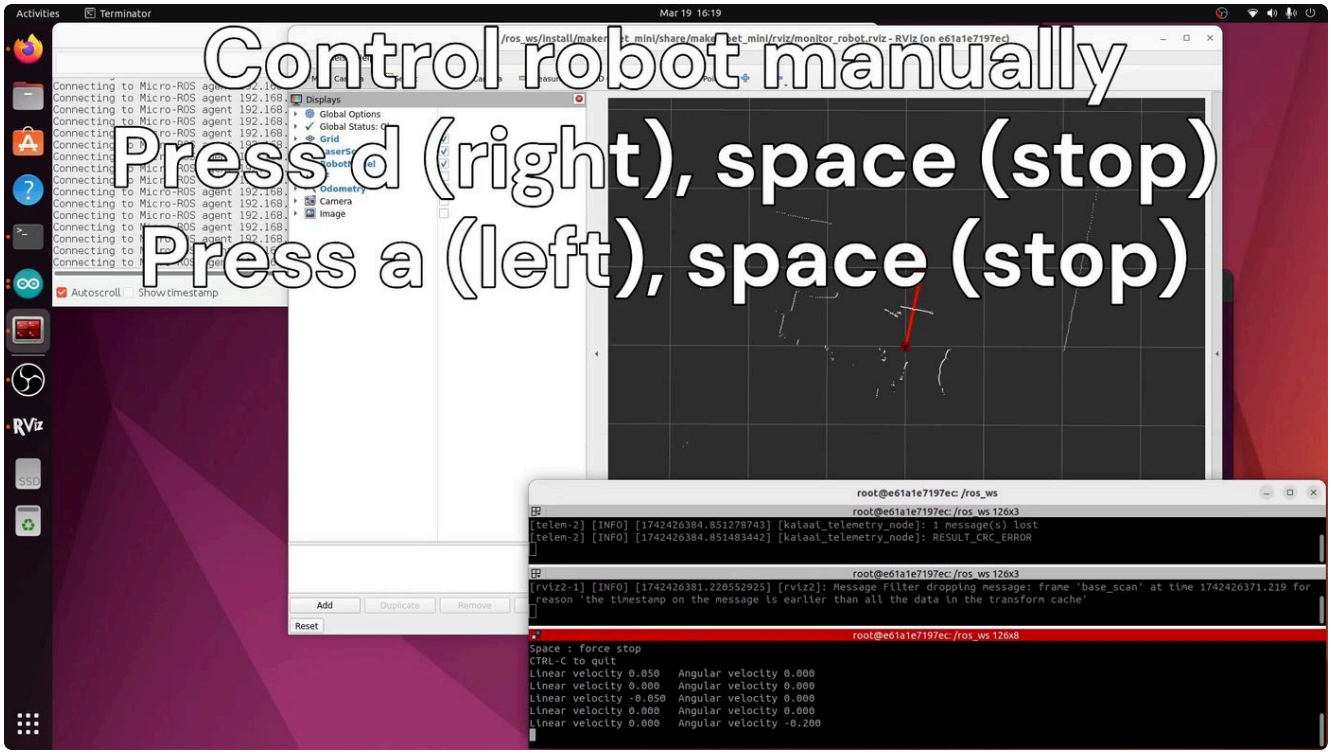
2. Launch the visualization:

```
ros2 launch kaiaai_bringup monitor_robot.launch.py
```



Three panes: bring-up, monitor and teleop.

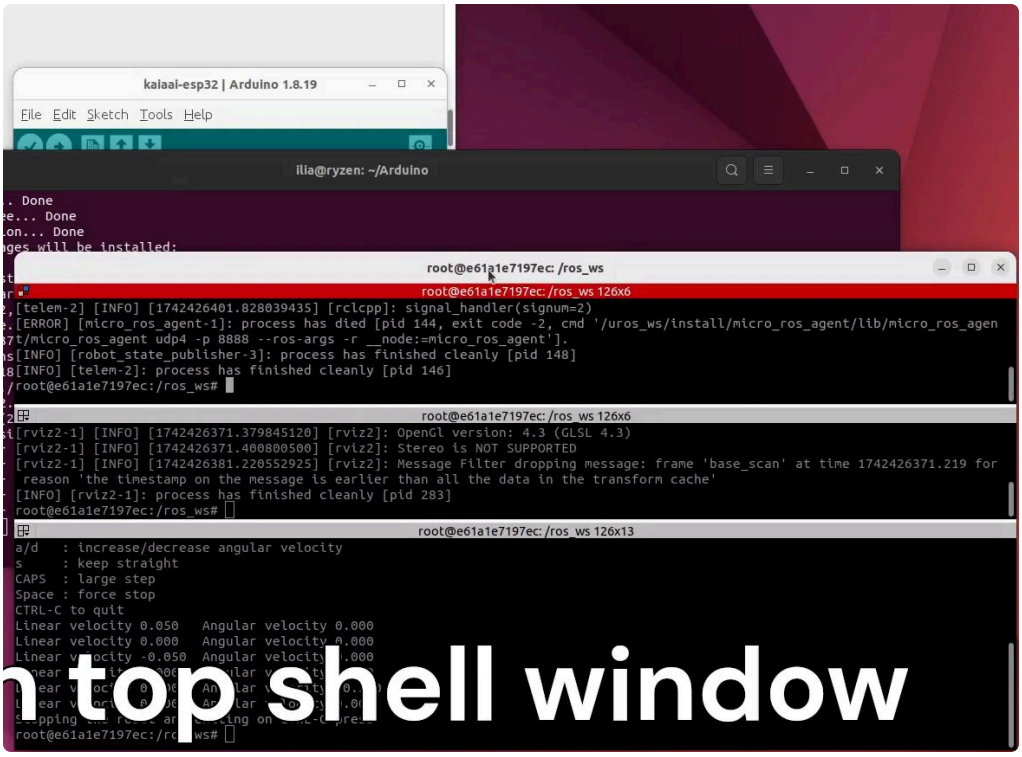
3. RViz shows the LiDAR scan and a red heading arrow. Drive from the teleop pane and watch the scan move.



RViz while driving with d, a and space.

Shut down

1. Close RViz.
2. Press **Ctrl+C** in each pane.
3. Type `exit` in the top pane. `--rm` removes the container and ends the other shells. Close Terminator.



Ctrl+C in each pane, then exit in the top pane.

Next time, just start the container, run `physical.launch.py` and turn the robot on. In the following chapters, use this chapter's `sudo docker run` wherever they show the Windows one.

> **MAPPING ON UBUNTU**
>
> Maps saved to `/root/maps` in the container land in `~/maps` on your PC. For simulation, drop `-p 4430:4430/tcp`:
>
> ```
> sudo docker run --name makerspet -it --rm -v ~/maps:/root/maps -p 8888:8888/udp -e DISPLAY
> -e QT_X11_NO_MITSHM=1 --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" --volume="${XAUTHORIT
> Y}:/root/.Xauthority" kaiaai/kaiaai:iron
> ```

Alternatives

ROS2 without Docker. On Ubuntu 22.04, install ROS2 Iron and the Kaia.ai packages with two scripts from github.com/kaiaai/install. Run them as your normal user; they ask for your password when they need `sudo`:

```
wget https://raw.githubusercontent.com/kaiaai/install/iron/ubuntu/install_ros2_iron_ubuntu_22_04.sh
bash install_ros2_iron_ubuntu_22_04.sh
wget https://raw.githubusercontent.com/kaiaai/install/iron/ubuntu/install_kaiaai_iron.sh
bash install_kaiaai_iron.sh
```

Wherever this chapter runs `sudo docker exec -it makerspet bash`, open an ordinary terminal instead.

ROS2 Jazzy. The Jazzy image, `kaiaai/kaiaai:jazzy`, starts on Linux with `start_jazzy.sh`:

```
wget https://raw.githubusercontent.com/kaiaai/install/jazzy/docker/utils/start_jazzy.sh
bash start_jazzy.sh
```

Without Docker, on Ubuntu 24.04:

```
wget https://raw.githubusercontent.com/kaiaai/install/jazzy/ubuntu/install_ros2_jazzy_ubuntu_24_04.sh
bash install_ros2_jazzy_ubuntu_24_04.sh
wget https://raw.githubusercontent.com/kaiaai/install/jazzy/ubuntu/install_kaiaai_jazzy.sh
bash install_kaiaai_jazzy.sh
```

> **JAZZY IMAGE: ADD --IPC=HOST WHENEVER YOU USE --NET=HOST**
> The Jazzy image passes ROS2 messages through shared memory, and `start_jazzy.sh` uses `--net=host`. In your own `docker run` with `--net=host`, add `--ipc=host`. Without it, ROS2 nodes outside the container (on your PC or in another `--net=host` container) see its topics but receive no messages. Nodes inside the container, and the robot, work either way.

Raspberry Pi and other ARM64 boards. The Docker image is x86-64 only. Install Ubuntu 22.04 on the Pi and run the two Iron scripts above (or follow the official ROS2 Iron Ubuntu .deb instructions). For Jazzy, use Ubuntu 24.04 and the two Jazzy scripts.

Ubuntu in a virtual machine on Windows. If WSL2 doesn't work, you can follow this chapter in an Ubuntu 22.04 VM. I don't recommend it: in my experience the VM makes ROS2 laggy. You can also upload the firmware from Windows (Chapter 3) and run only ROS2 in the VM.

Appendix A. Run Ubuntu in a Virtual Machine on Windows

Video: How to set up Ubuntu 22.04 on a Windows PC - using a virtual machine

> **USE THIS ONLY IF WSL2 WON'T INSTALL**
> I don't recommend this approach: in my experience, a virtual machine makes ROS2 laggy. Use WSL2 if you can.

Here you will create a virtual machine (VM) with VMware Workstation Player, free for personal use, and install Ubuntu 22.04 in it. Then continue with Chapter 8, running every command inside the VM. You can keep Arduino IDE on Windows (Chapter 2) instead of installing it in the VM.

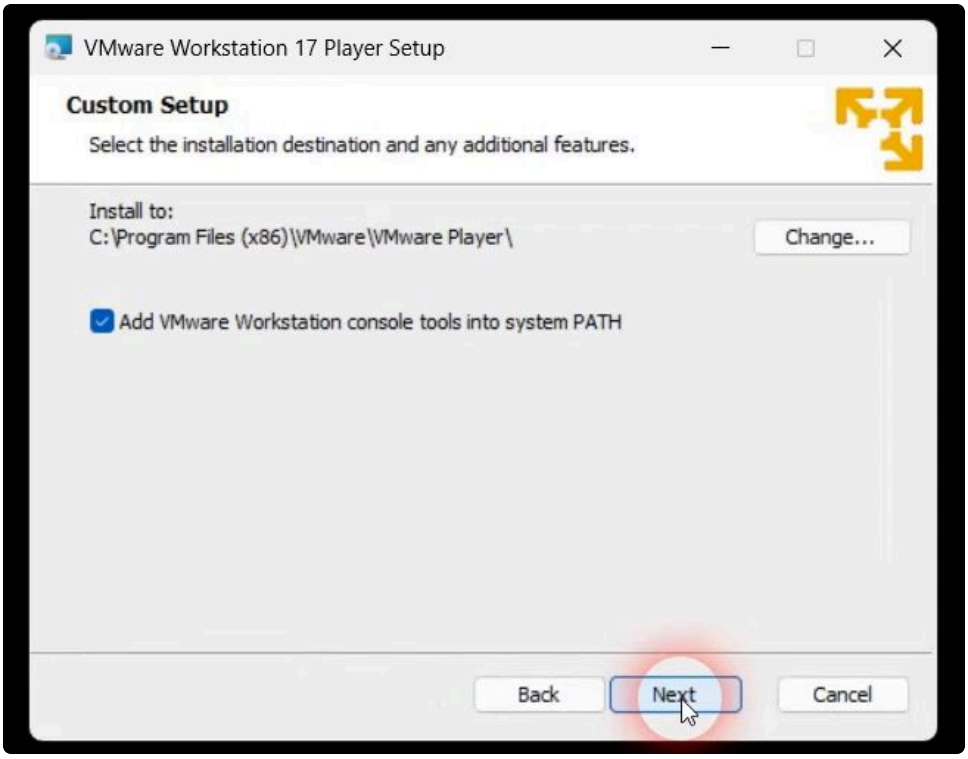
You need about 50 GB of free disk space and enough RAM to give the VM at least 4 GB; 16 GB in the PC is comfortable.

Install VMware Workstation Player

1. Download the VMware Workstation installer for Windows. It's now free for personal, educational and commercial use, with no license key. The video uses VMware Player 17.6.3.

   > **WHERE TO DOWNLOAD**
   > I've had a good experience downloading it from TechSpot (search for "VMware Workstation TechSpot"). You can also download it directly from Broadcom, but that needs a Broadcom account and, in my experience, is a hassle.

2. Run the installer, accept the license agreement and click **Next**.
3. On **Custom Setup**, keep the default folder (`C:\Program Files (x86)\VMware\VMware Player\`) and the **Add VMware Workstation console tools into system PATH** checkbox. Click **Next**.

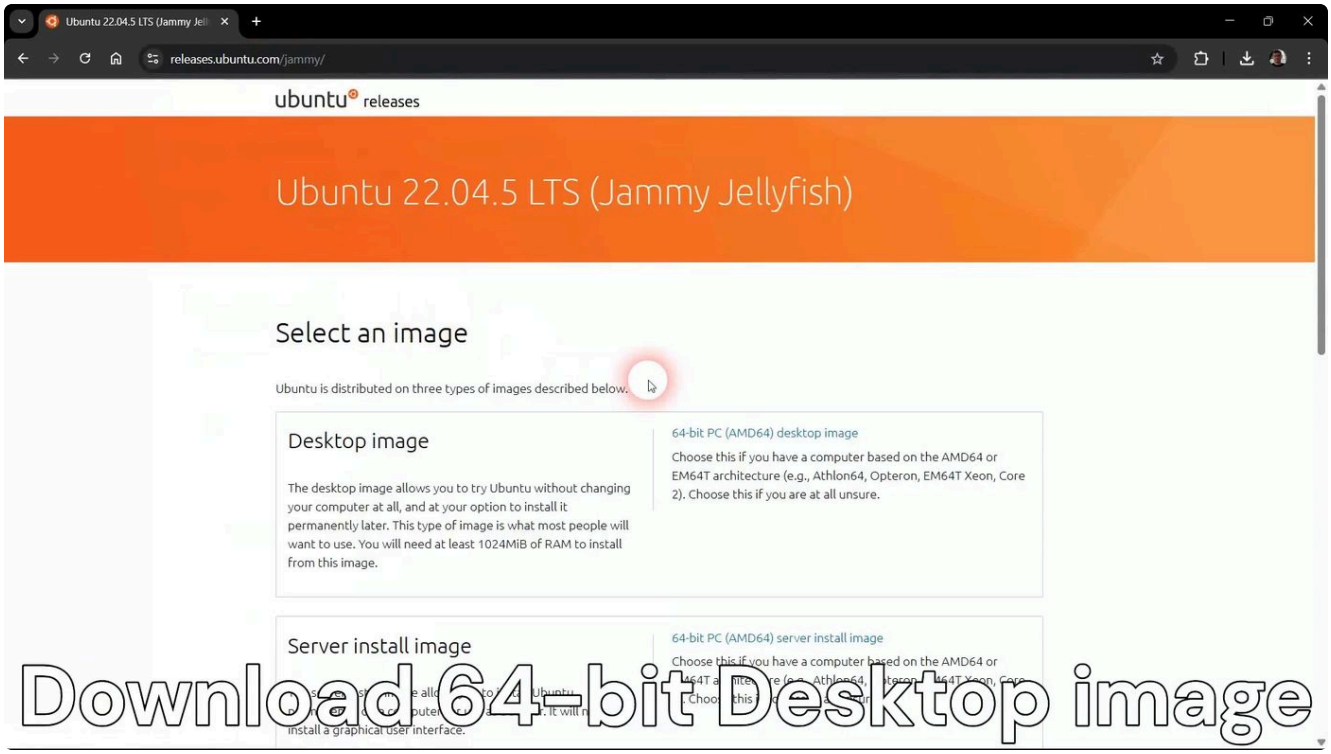


Custom Setup page.

4. On **User Experience Settings**, choose your update and Customer Experience options and click **Next**.
5. Keep the shortcuts, click **Next**, then **Install**.

Download Ubuntu 22.04

1. While VMware installs, search for "Ubuntu 22.04 download" to reach `releases.ubuntu.com/jammy/` (22.04.5 LTS at the time of recording).
2. Under **Desktop image**, click **64-bit PC (AMD64) desktop image**. The `.iso` is several gigabytes.

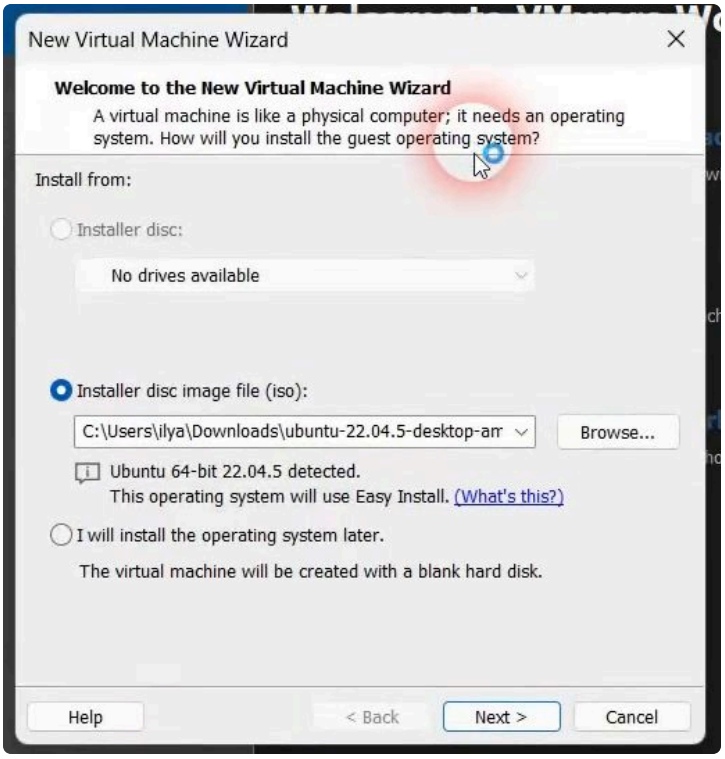


Download the 64-bit desktop image.

> **TIP**
> Download 22.04, not the newest release. The robot's ROS2 software is set up for 22.04.

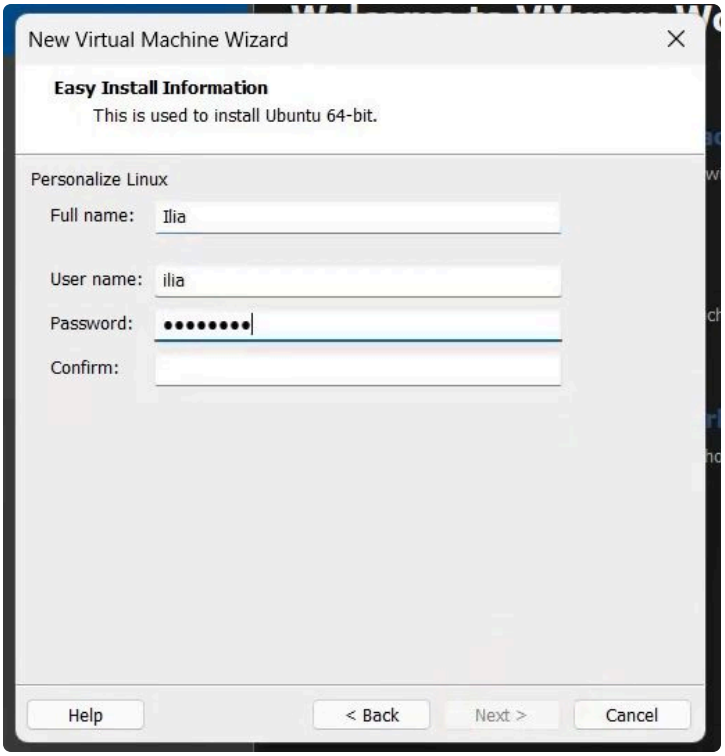
Create the virtual machine

1. Launch VMware Workstation Player and click **Create a New Virtual Machine**.
2. Select **Installer disc image file (iso)**, click **Browse...** and pick the Ubuntu `.iso`. VMware detects "Ubuntu 64-bit 22.04.5" and will use Easy Install. Click **Next**.



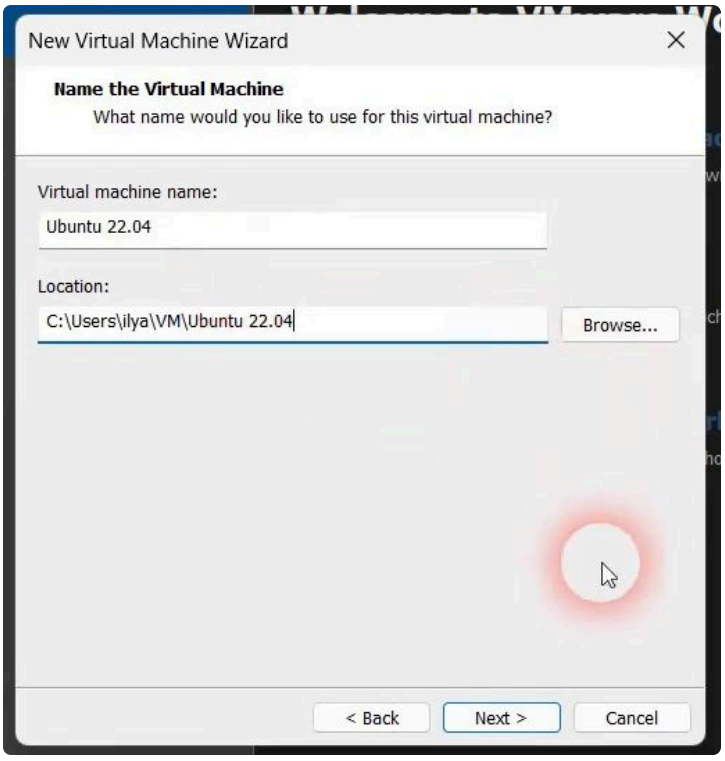
Select the downloaded Ubuntu image.

3. Enter your full name, Ubuntu user name and password. Click **Next**.



Easy Install Information.

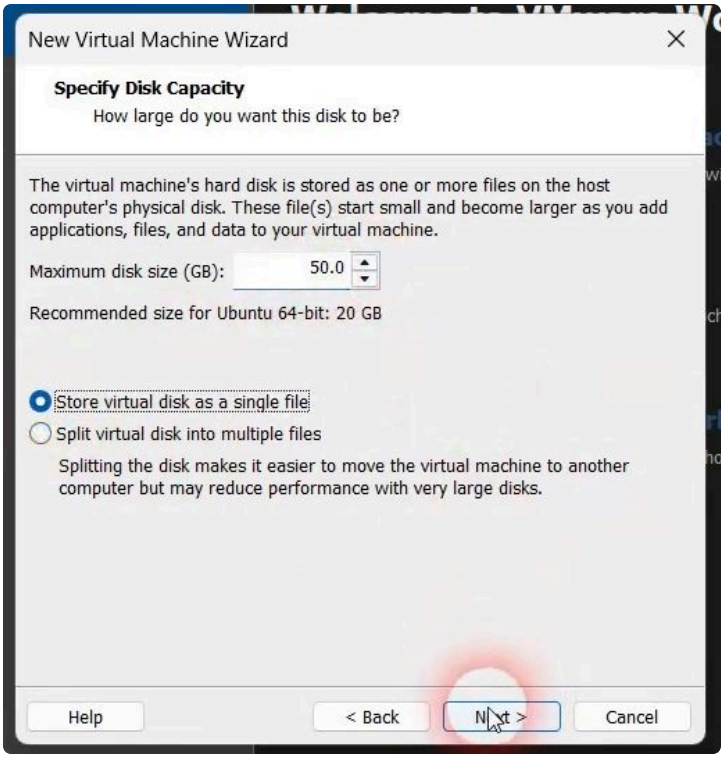
4. Name the VM, for example `Ubuntu 22.04`, and set **Location** outside OneDrive, for example `C:\Users\<you>\VM\Ubuntu 22.04`. Click **Next**.



Name the VM and store it outside OneDrive.

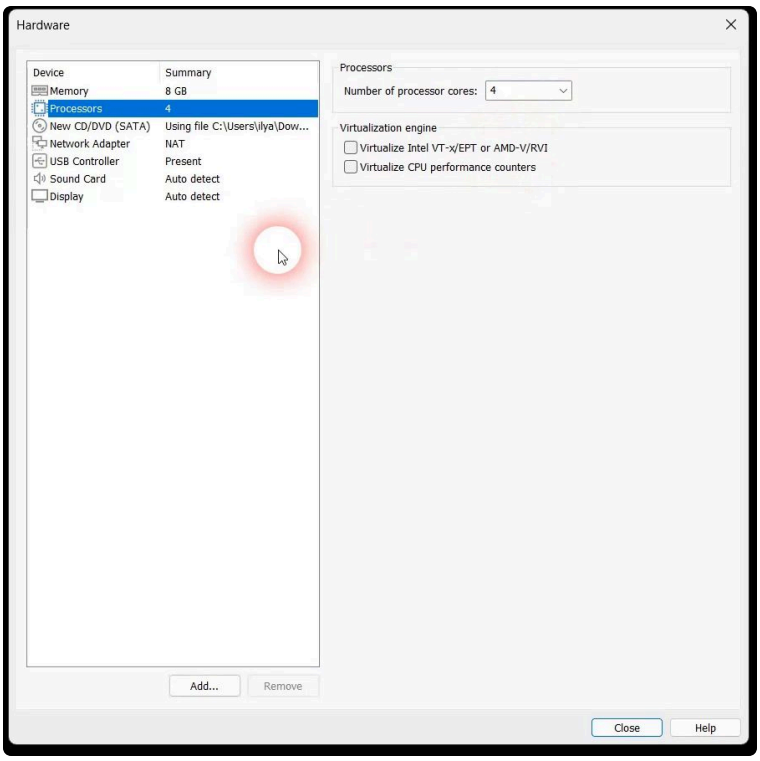
> **KEEP VM FILES OFF ONEDRIVE**
> The default location is `...\OneDrive\Documents\Virtual Machines\`. OneDrive would keep backing up the huge, constantly changing VM files, slowing your PC and network.

5. On **Specify Disk Capacity**, I set **Maximum disk size** to `50` GB instead of the recommended 20 GB and choose **Store virtual disk as a single file**. The file grows as needed. Click **Next**.



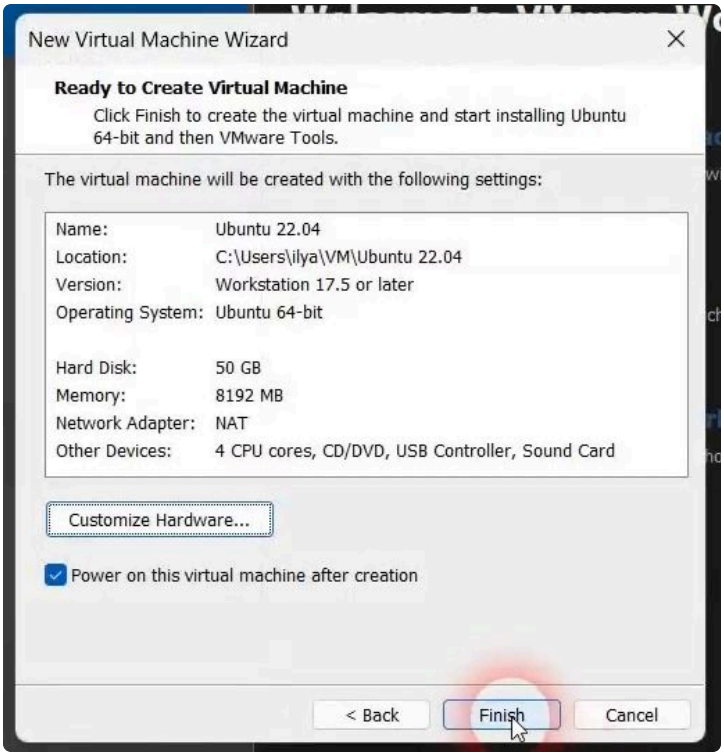
50 GB disk, single file.

6. On **Ready to Create Virtual Machine**, click **Customize Hardware...**.
7. Set **Memory** and **Processors**. My PC has 16 GB and 8 cores, so I give the VM 8 GB (8192 MB) and 4 cores. Use at least 4 GB. Click **Close**.



8 GB memory and 4 processor cores.

8. Leave **Power on this virtual machine after creation** checked and click **Finish**.



Summary: 50 GB disk, 8192 MB memory, 4 CPU cores.

Install Ubuntu in the VM

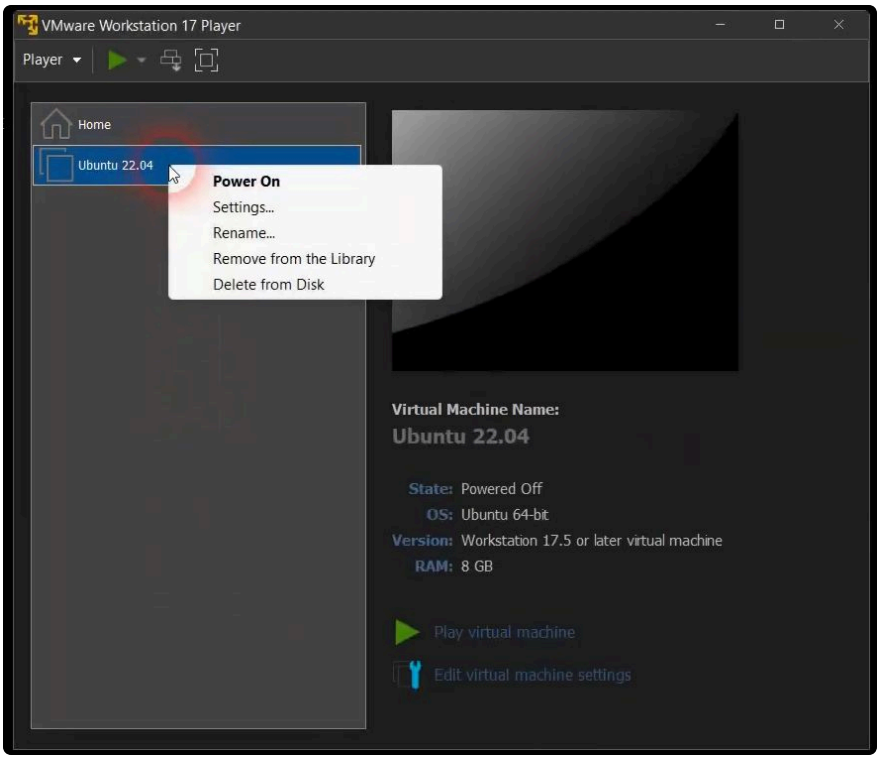
The VM boots into the Ubuntu installer.

1. Choose your keyboard layout and click **Continue**.
2. Follow the installer's prompts. Its defaults work; the disk it erases is the VM's virtual disk, not your Windows drive.
3. When **Installation Complete** appears, click **Restart Now**, then log in.

> **WARNING**
> If Ubuntu offers to upgrade to 24.04, decline. The robot software in this book needs 22.04.

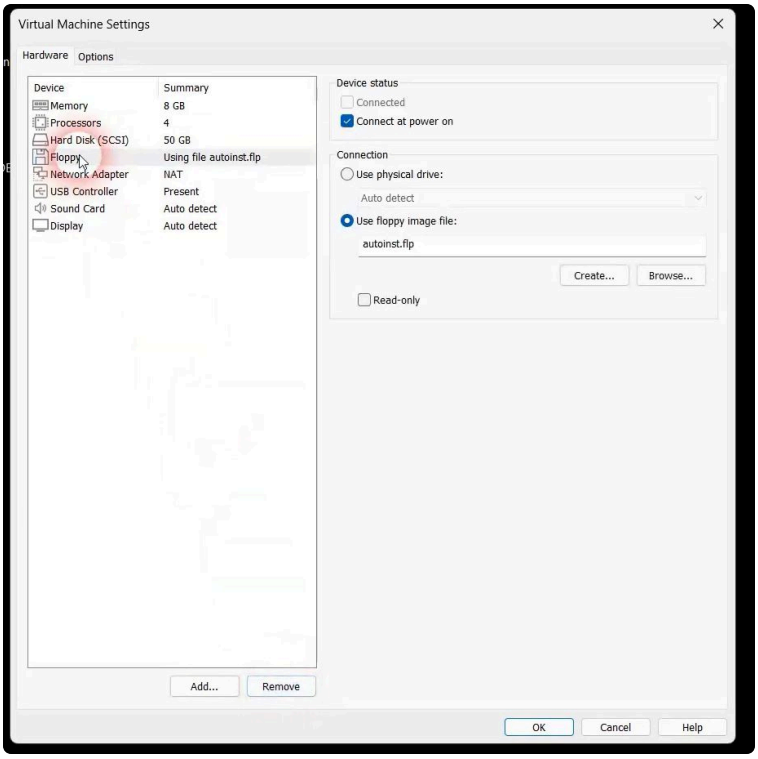
Remove unneeded hardware and set up networking

1. Shut down the VM: system menu (top right) > **Power Off / Log Out** > **Power Off...**.
2. If VMware shows a **Removable Devices** hint, click **OK**. Each USB device connects either to Windows or to the VM, not both.
3. In VMware Workstation Player, right-click your `Ubuntu 22.04` VM and choose **Settings...**.



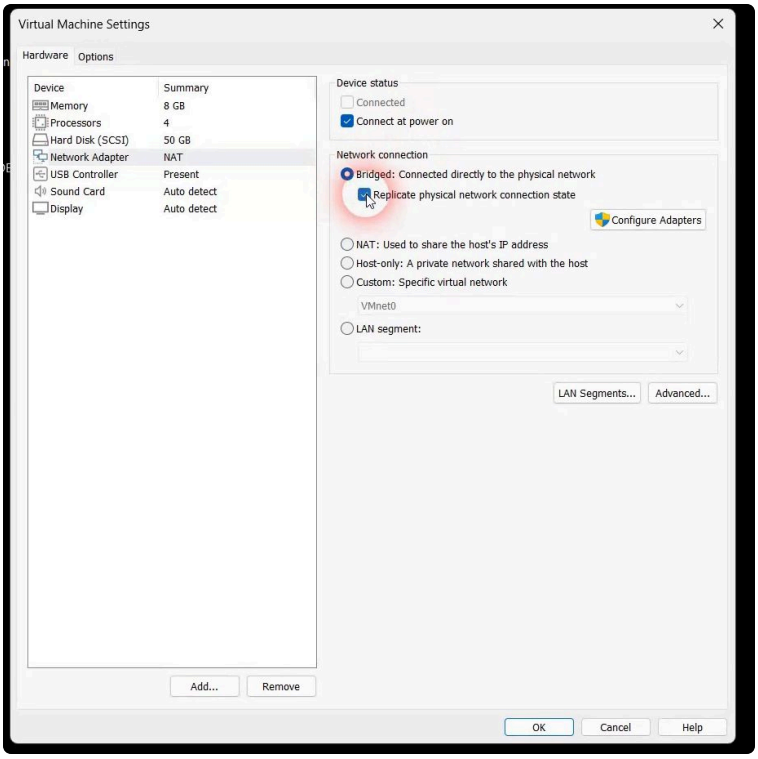
Open the VM settings.

4. Select and **Remove** the leftovers from Easy Install: both **CD/DVD (SATA)** drives (the Ubuntu `.iso` and `autoinst.iso`) and the **Floppy** (`autoinst.flp`).



Remove the CD/DVD drives and the floppy.

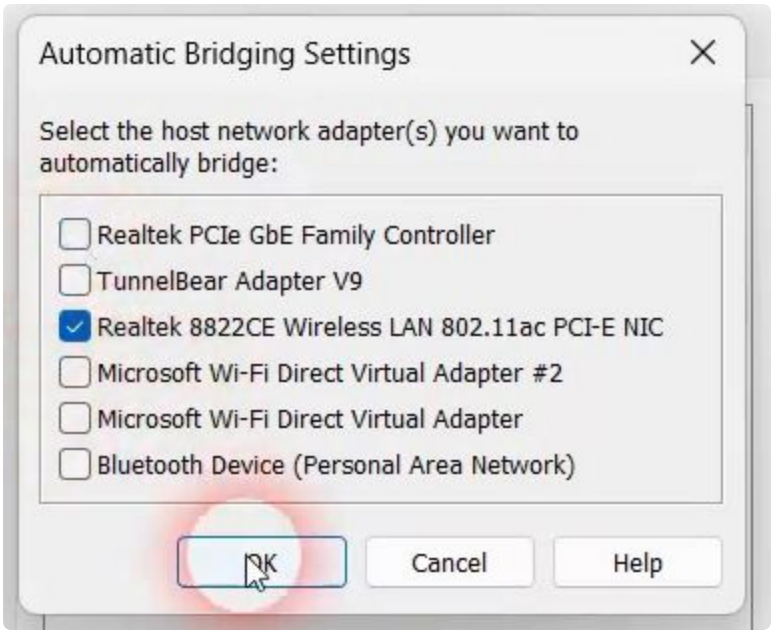
5. To make the VM reachable from your local network, select **Network Adapter** and change **Network connection** from **NAT** to **Bridged: Connected directly to the physical network**. Keep **Replicate physical network connection state** checked.



Set the network connection to Bridged.

> **WHY BRIDGED?**
> With NAT the VM hides behind your PC's IP address. Bridged gives it its own address on your network, like a separate computer.

6. Click **Configure Adapters** and check only the physical adapter you use. My PC is on Wi-Fi, so I keep only **Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC**. Your names will differ. Click **OK**.



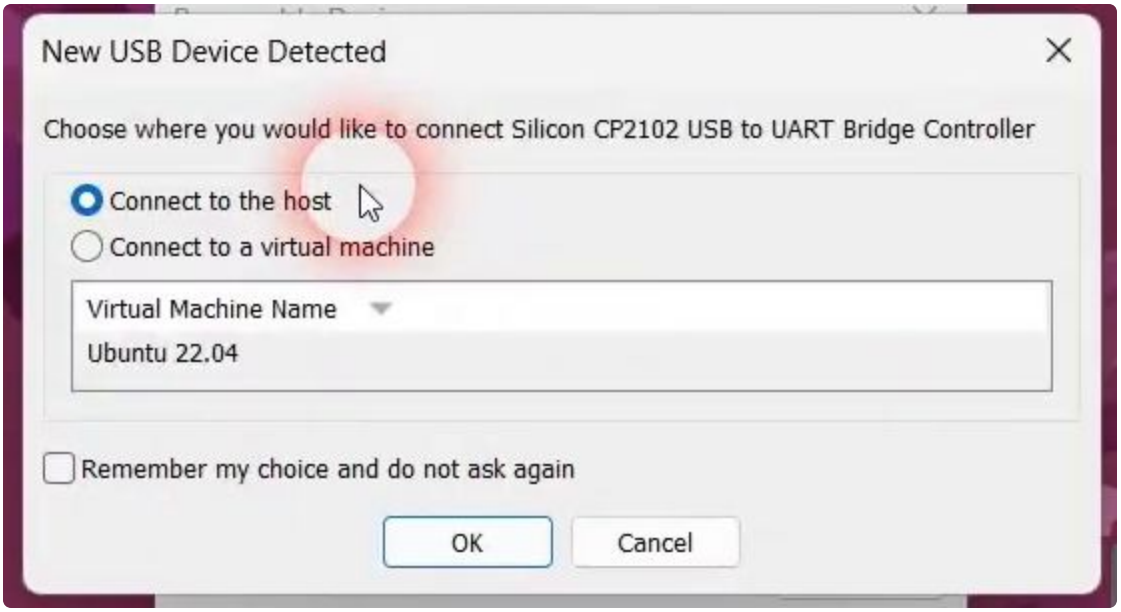
Bridge to the Wi-Fi adapter only.

7. Click **OK** and launch the VM.

Install the robot software and connect USB devices

Continue with Chapter 8 to install Arduino, Docker and ROS2, running every command inside the VM.

When you plug in a USB device, such as the robot's ESP32 board, VMware asks whether to connect it to Windows (the host) or to the VM.



Connect the ESP32's CP2102 USB-to-UART bridge to the host or the VM.

- **Connect to the host** if Arduino IDE runs on Windows.
- **Connect to a virtual machine** (`Ubuntu 22.04`) if Arduino IDE runs in the VM.

Don't check **Remember my choice and do not ask again** until you're sure.

Appendix B. Troubleshooting and FAQ

The most common problems, by symptom. The latest version is at makerspet.com/blog/bld-120mm-pack.

Look for clues in two places:

- **The ESP32.** Connect it over USB, open the Arduino Serial Monitor at 115200 baud and press EN (reset). The firmware prints its configuration, connection progress and telemetry (see B.9).
- **ROS2.** Read the output in the shell where you launched it.

B.1 Where to get help

If this appendix doesn't cover your problem, ask:

- On the support forum: github.com/makerspet/support/discussions.
- On the Maker's Pet Discord server.

Include your Serial Monitor output, full ROS2 output, exact `docker run` command, system (Windows with WSL, Ubuntu, a virtual machine) and wiring photos. For navigation problems, add a screen recording of RViz and the terminal plus a phone video of the robot.

B.2 Power and batteries

Battery connected, but no power

- Check the board's ON/OFF switch.
- Check the battery wires are in the correct screw terminals and make good contact. Measure the voltage at the board's power terminals.
- Check the polarity. **Reversed polarity will burn the board.**
- Wiggle the batteries in the holder. Measure each battery, and the voltage at the holder terminals.
- Check the ESP32 power LED lights up with the switch ON.

Can I use 18650 Li-Ion batteries instead of 6×AA?

The BDC-30P board is designed for alkaline batteries and has **no on-board protection** for rechargeable cells. If you still use them, the 120 mm base fits a 2-cell 18650 holder:

- Use **protected** cells in series (short circuit, overcurrent, overcharge, overdischarge and so on).
- Take the cells out to charge them in your own charger.
- The robot draws up to around 3 A continuous and around 5 A peak; typical consumption is much lower (see Power consumption).

Mount the rear posts in the outer spare holes of the base, fit the battery-holder backstop behind the battery with two M3 countersunk screws, and use the wider LiDAR skirt that matches the rear post positions.

> **WARNING**
> Using rechargeable batteries is at your own risk.

> **TIP**
> If the 18650 holder hits the LiDAR posts, check the assembly against the 18650 mod photo in Chapter 1: the rear posts go in the base's outer spare holes.

What battery voltage drives the motors? (Vbat, Vmot, JP2)

The board steps the battery voltage (Vbat) **up** to the motor voltage (Vmot), with about 2 V of headroom:

- For 12 V Vmot (JP2 closed, the default), Vbat must be about 6–10 V. A 3S LiPo (11.1 V) will not work reliably; use a 2S pack (about 7.4 V).
- For 24 V Vmot (JP2 open), Vbat may be about 6–21 V.

JP2 selects the Vmot output; it does not pass raw battery voltage to the motors.

Power consumption

Measured with a 9 V battery, the ESP32 connected to the ROS2 PC over WiFi (USB unplugged), an LDROBOT LD14P LiDAR running and two 12 V N20 motors, firmware 0.8.0-iron:

Condition	Battery current
Both motors off	about 240 mA
Both motors full on, spinning freely	about 430–450 mA

B.3 Motors and encoders

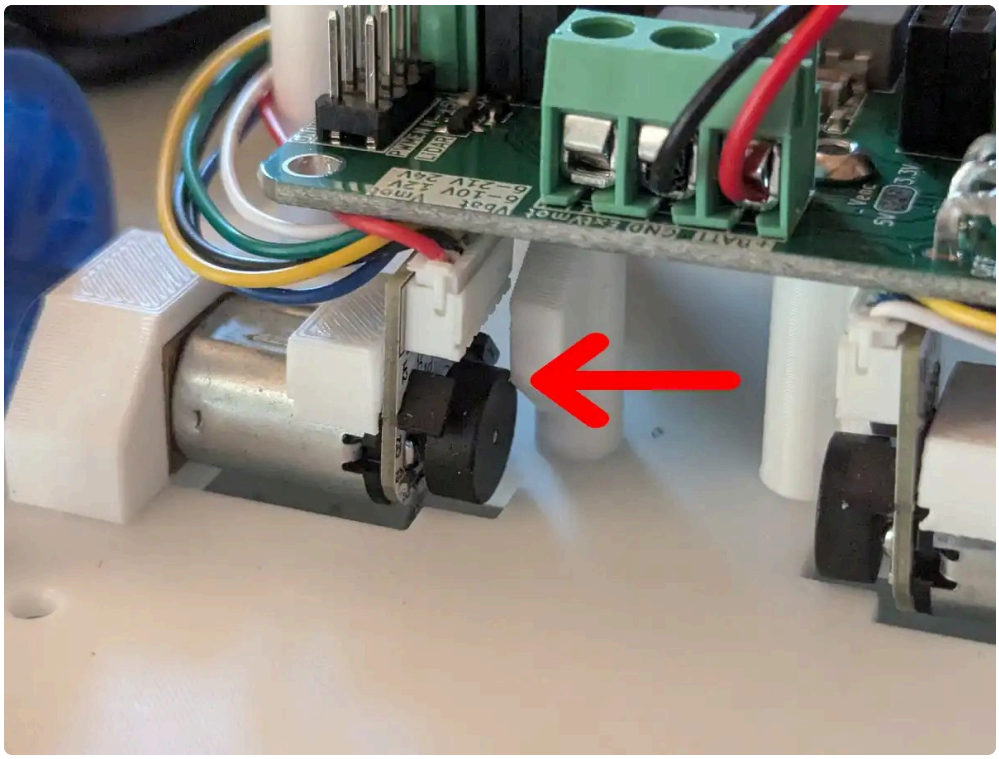
A motor doesn't turn, or behaves strangely

Start with the mechanics and wiring:

- Make sure the board's post tab doesn't touch a motor (see the figure). A blocked motor can burn out.
- Check the switch is ON and the batteries are connected and fresh.
- **Double-check the motor connections**, the most common issue:
  - The motor's connector housing can slide up, off its pins. Press it back down fully.
  - Make sure the motor wires go to the correct screw terminals.

    - A screw terminal wire can break off or fail to make contact. Follow Chapter 1 for reliable connections.

- Check the green LED on each motor. Off means the encoder has no power: check the battery, switch and cable connections at both ends.
- Keep debris out of the gearbox; it can seize the gears and burn out the motor.
- Don't overtighten the motor screws. Bent plastic (including the base) can catch the gears or the encoder disk and burn out the motor.
- Gently spin the magnet to turn the motor's core shaft. It should turn freely; if it doesn't, the motor or its gearbox may be jammed.
- Bent encoder sensors can touch the encoder wheel. Bend them back and check the magnet spins freely.



The post tab must not touch the motor, as it does here.

Then test: start teleop in ROS2 and press **W** to turn the motors forward.

A motor does not move at all

- Check for cables blocking the encoder magnet.
- Check that neither motor is mechanically stalled.
- Check that the M1 and M2 wires are firmly connected to the board.
- Check that the batteries are fresh.

One motor works, the other doesn't

Swap the two motors (see Chapter 1). If the problem follows the motor, the motor or its cable is at fault; if it stays on the same side, check that side's board connections.

A motor spins at full speed in one direction no matter what you command

Forward, backward and stop all give the same result.

- Swap the motor's ENCA and ENCB encoder wires.
- Check that both encoder wires are connected reliably.

A motor spins only at full speed, but does change direction

An encoder wire probably has a bad connection, at the board or at the motor.

Both motors turn at one fixed speed and pressing a key repeatedly doesn't change it

Usually motor wiring. Try swapping M+ and M−, and/or ENCA and ENCB (forum #63).

A motor turns the wrong way but otherwise responds correctly

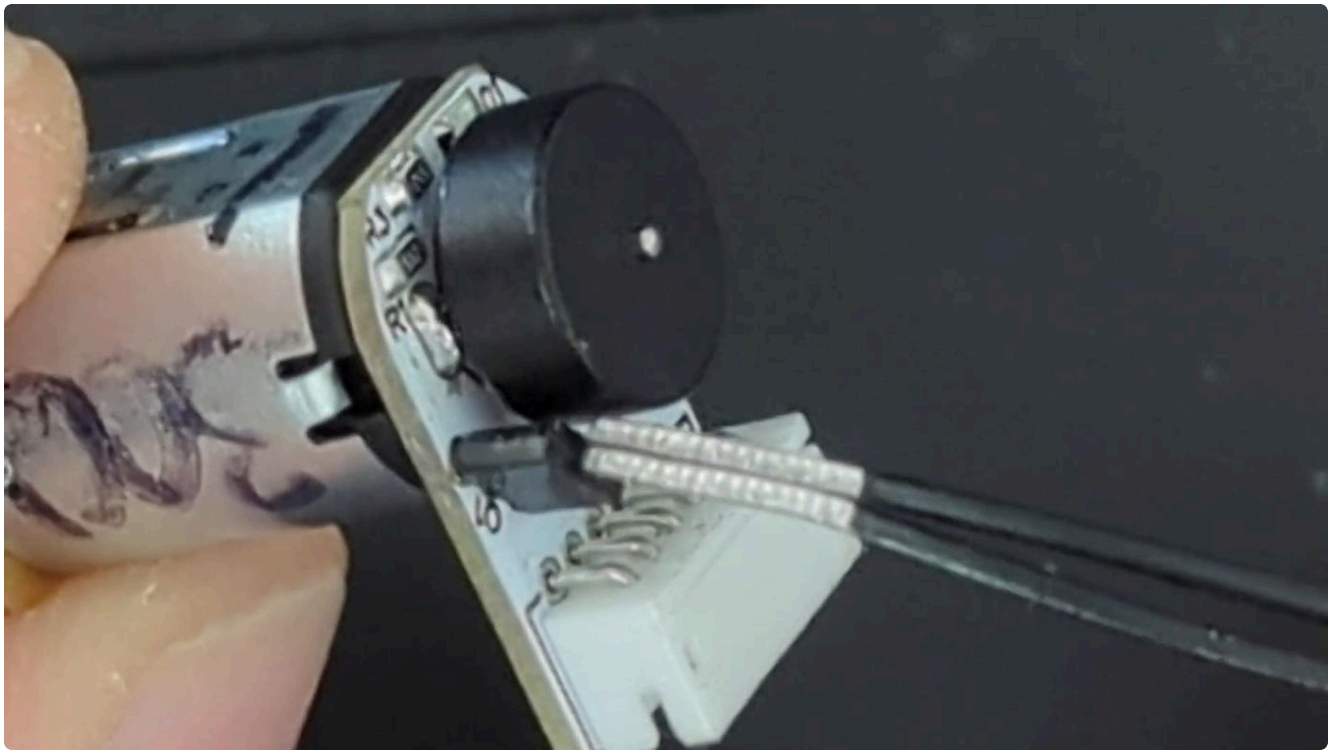
Swap the motor's ENCA and ENCB wires, then swap its M1 and M2 wires.

A motor turns unevenly

- Check for flaky wire connections.
- With third-party motors (not from Maker's Pet), you may need to tune the motor PID coefficients in `config.yaml`.

Motor changes speed abruptly

Check the soldering on the motor's encoder board and resolder any loose components.



A poorly soldered Hall sensor on the N20 encoder board.

A motor runs at full speed at power-up, before WiFi connects

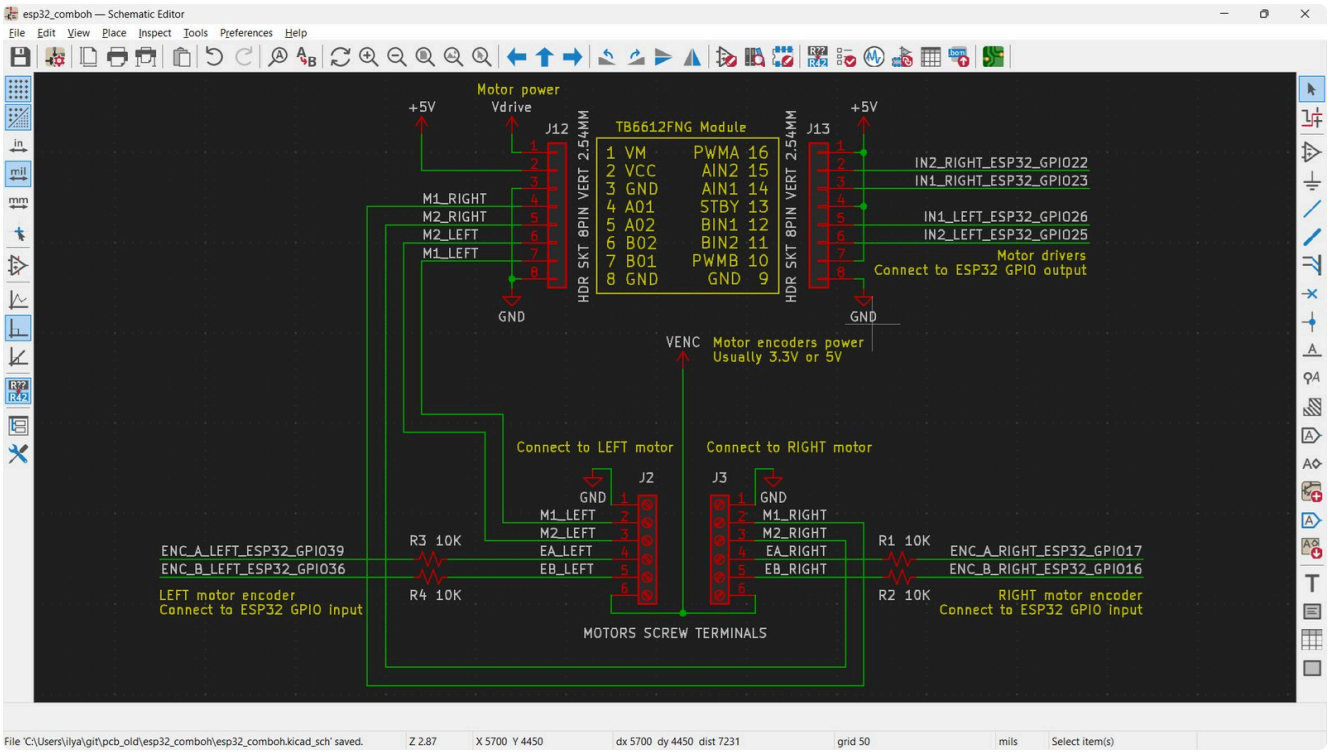
Some ESP32 motor-driver GPIOs float or go HIGH during boot or reset, spinning a motor at full speed until the firmware takes control. It's not a wiring fault. Drive those pins LOW with `digitalWrite(pin, LOW);` at the very start of `setup()`.

My motor won't spin, change speed or stop (external motor driver)

With a separate motor driver board, the firmware PWMs the driver's IN1/IN2 inputs. TB6612FNG, DRV8871, DRV8833, DRV8835 and L298N all work. Two things are required:

1. **Hold the driver's enable pins HIGH**, or the motor won't move. On a TB6612FNG, tie PWMA, PWMB and STBY high; on an L298N, ENA and ENB (with the on-board jumpers or to 5 V). Either wire them high, or add `digitalWrite(PWMA, HIGH);` (and so on) at the top of `setup()`.
2. **Set the driver type in** `config.yaml` (`IN1_IN2` for these drivers) and map each motor's GPIOs. If a wheel spins the wrong way or won't stop, swap that motor's `in1` / `in2`, or its encoder `a` / `b`.

See the TB6612FNG worked example, the schematic below (forum #24) and the configuration file reference. Pick any GPIOs within the ESP32's limitations (ESP32, ESP32-S3). An L298N wired this way works with the unmodified firmware and `config.yaml` (forum #77).



TB6612FNG wiring to a 30-pin ESP32; the GPIO numbers are an example. From forum #24.

Using a BLDC motor with a built-in controller (ESC-type, FG output)

Five-wire "ESC-type" BLDC motors have a built-in driver, so no external ESC is needed. In `config.yaml`, set `motor.driver.type: PWM_CW` and `motor.encoder.type: FG`, then map each side's FG, PWM and CW GPIOs.

- FG reports speed only, not direction; direction follows the CW/CCW input.
- The PID needs roughly 1000 PPR. A low-PPR FG signal makes low-speed control twitchy.

> **WARNING**
> A BLDC motor's PWM and CW inputs may be pulled up internally to 12 V or 24 V, which will damage the ESP32. Use a logic-level converter.

Can I use an IMU (BNO055, MPU6050) instead of wheel encoders?

No. There's no built-in IMU support, and the motor PID needs encoders to control RPM. You can map and navigate without odometry (an option in `navigation.yaml`), but not without encoders. IMU streaming is possible but not trivial: the WiFi/UDP link drops some high-rate IMU packets.

B.4 Arduino IDE: compile and upload

Compilation fails with "This code runs on ESP32"

```
aiaai-esp32:16:4: error: #error This code runs on ESP32
   #error This code runs on ESP32
    ^~~~~
In file included from /home/ilia/Arduino/kaiaai-esp32/kaiaai-esp32.ino:19:0:
robot_config.h:16:10: fatal error: SPIFFS.h: No such file or directory
 #include <SPIFFS.h>
          ^~~~~~~~~~
compilation terminated.
exit status 1
#error This code runs on ESP32
```

Select your board under **Tools → Board → ESP32 Arduino** (usually ESP32 Dev Module). If ESP32 Arduino isn't listed, add the Espressif ESP32 board package (Chapters 2 and 8).

Which ESP32 board package version should I use?

Use **2.0.17**.

- Not 2.0.15: its HardwareSerial is broken, so the LiDAR returns no data. Use 2.0.16 or newer.
- Not 3.x: it removed `ledcAttachPin` and `ledcSetup`, so the firmware won't compile.

"use of deleted function MotorController()" or "ledcAttachPin was not declared"

The Arduino IDE 2.x Library Manager can silently replace the firmware's MotorController and PID libraries with same-named ones. Copy the firmware's libraries back into your Arduino libraries folder and rename them (for example `MotorController_kaia`).

The sketch's `#include` lines don't need to change: Arduino finds libraries by their header files, not their folder names. Alternatively, reinstall the Arduino IDE from scratch and don't install any third-party libraries; use only the ones bundled with the firmware.

Upload fails with "Failed to connect to ESP32: No serial data received"

The upload stops at `Connecting.....`:

```
esptool.py v4.5.3
Serial port COM3
Connecting.....................................

A fatal error occurred: Failed to connect to ESP32: No serial data received.
```

Retry, and as soon as `Connecting...` appears, hold the ESP32's **BOOT** button for 3–5 seconds, then release it. The same applies to sketch data (SPIFFS) uploads. Some boards don't need BOOT at all.

Upload fails on Ubuntu: "No module named 'serial'"

```
Traceback (most recent call last):
  File "/home/ilia/.arduino15/packages/esp32/tools/esptool_py/4.5.1/esptool.py", line 31, in <
module>
    import esptool
  ...
  File "/home/ilia/.arduino15/packages/esp32/tools/esptool_py/4.5.1/esptool/loader.py", line 3
0, in <module>
    import serial
ModuleNotFoundError: No module named 'serial'
exit status 1
Error compiling for board ESP32 Dev Module.
```

Install pyserial (Chapter 8):

```
sudo apt install -y python3-pip
pip3 install pyserial
```

Arduino IDE doesn't detect the ESP32 COM port

- Make sure the ESP32 is powered and its power LED is on.
- Try another USB cable. Some are **charge-only**, with no data wires.
- The ESP32's USB-to-serial chip can fail. Try the ESP32 on another PC, and another ESP32 on yours.

SPIFFS upload fails: "No port specified" or "Could not open COM3"

```
SPIFFS Filesystem Uploader

Using partition: default
ERROR: No port specified, check IDE menus
```

Restart the Arduino IDE and retry.

```
Uploading SPIFFS filesystem
C:\Users\ilya\AppData\Local\Arduino15\packages\esp32\tools\esptool_py\4.5.1/esptool.exe --chip
esp32 --port COM3 --baud 921600 --before default_reset --after hard_reset write_flash -z --fla
sh_mode dio --flash_freq 80m --flash_size detect 2686976 C:\Users\ilya\AppData\Local\Temp\tmp-
16336-VVuaXOVTskId-.spiffs.bin
esptool.py v4.5.1
Serial port COM3

A fatal error occurred: Could not open COM3, the port doesn't exist
ERROR: Upload failed, error code: 2
```

- **Close the Serial Monitor**, the most common cause; it holds the port.
- Make sure the ESP32 is connected and its USB serial port works.
- Try another PC, or another ESP32.

SPIFFS upload fails: "SPIFFS_write error(-10010)"

```
SPIFFS_write error(-10010): unknown
error adding file!
```

A file name in the sketch's `data` folder is too long. Keep names to 30 characters or less.

SPIFFS upload fails: "This chip is ESP32-S3 not ESP32"

```
A fatal error occurred: This chip is ESP32-S3 not ESP32. Wrong --chip argument?
SPIFFS Upload failed!
```

Select the correct board in the Arduino IDE.

The SPIFFS upload command doesn't appear in Arduino IDE 2.x

Arduino IDE 2.x needs a separate SPIFFS upload plugin (Chapter 2): copy everything inside the plugin's `.arduinoIDE` folder into `C:\Users\<your user name>\.arduinoIDE\plugins\` and restart the IDE. If the command still doesn't show up, install Arduino IDE 1.8.19 with its SPIFFS plugin alongside 2.x (forum #87).

SPIFFS upload fails on Ubuntu without a reason

The plugin builds the image, doesn't seem to try uploading, and fails:

```
[SPIFFS] upload : /tmp/arduino_build_928471/kaiaai-esp32.spiffs.bin
[SPIFFS] address: 2686976
[SPIFFS] port   : /dev/ttyUSB0
[SPIFFS] speed  : 921600
[SPIFFS] mode   : dio
[SPIFFS] freq   : 80m

SPIFFS Upload failed!
```

Upload the sketch data from a Windows PC instead. One builder saw this every time with Arduino IDE 1.8.19 on Ubuntu 24.04; Arduino IDE 2.x on Windows worked (forum #70).

The Serial Monitor prints garbage

- Set the Serial Monitor to 115200 baud.
- Press the ESP32's EN (reset) button. Garbage often appears right after a firmware upload.

Which ESP32 boards are supported?

ESP32, ESP32-S3 and ESP32-E. Each has a matching config file in the firmware's data folder. The kit's board uses the default `config.yaml`.

> **WARNING**
> An ESP32-S3 config on a plain ESP32 (or the other way round) can make the board keep booting into WiFi setup (AP) mode: the configs use a different reset-button GPIO, so the button looks permanently pressed.

B.5 WiFi and connecting to the ROS2 PC

What the ESP32 activity LED tells you

LED	Meaning
Solid on	AP mode: the ESP32 runs its own `KAIA.AI` WiFi hotspot for setup
Slow blinking, once per second	Connecting to your WiFi
Very slow blinking, once every 20 seconds	Connected to WiFi, connecting to the ROS2 PC
Fast blinking	Connected to the ROS2 PC

How to reset the WiFi configuration

To re-enter your WiFi name and password or your PC's IP address:

1. Power up the ESP32 (USB or battery).
2. Press the ESP32 reset (EN) button.
3. Within one second, press and hold BOOT. The LED blinks very fast.
4. Keep holding BOOT until the LED turns solid on.
5. Release BOOT. The ESP32 restarts in AP (WiFi configuration) mode.
6. Connect your phone or PC to the `KAIA.AI` WiFi, open `http://192.168.4.1`, enter your settings and click **Connect**. Connect saves the settings and restarts the ESP32; there's no Save button.

> **TIP**
> While connected to `KAIA.AI`, you can check what the ESP32 stored at `http://192.168.4.1/network.yaml` (WiFi settings) and `http://192.168.4.1/config.yaml` (board configuration) (forum #93).

The ESP32 doesn't connect to WiFi

- Check the WiFi name and password you entered.
- Your WiFi must be **2.4 GHz**; the ESP32 doesn't support 5 GHz.
- Press the ESP32 reset (EN) button. This often helps if the WiFi is configured correctly.
- Make sure you uploaded the correct `config.yaml` for your board (see "Which ESP32 boards are supported?" above).
- Reset the WiFi configuration (above) and enter your settings again.

The ESP32 connects to WiFi, but not to the ROS2 PC

The Serial Monitor keeps printing `Connecting to Micro-ROS agent ...` without `success`.

- Make sure the `kaiaai/kaiaai:iron` container is running and that inside it you launched `ros2 launch kaiaai_bringup physical.launch.py`.
- Make sure the robot and the PC are on the same local WiFi network.
- Cafe, restaurant and university WiFi often blocks devices from talking to each other ("client isolation"). Test with a phone hotspot (forum #69).
- Make sure your PC's firewall accepts incoming local connections.
- Try disabling your antivirus. (Avast started blocking these ports in December 2025.)
- Check whether your PC's IP address has changed. If so, reset the WiFi configuration and enter the new address.
- As a last resort, bind the published ports to your PC's IPv4 address in the `docker run` command. This fixed it for a Windows user who could ping the robot (forum #93):

```
docker run ... -p <PC IPv4 address>:8888:8888/udp -p <PC IPv4 address>:4430:4430/tcp ...
```

Frequent CRC errors or "message(s) lost"

Frequent `RESULT_CRC_ERROR` and `N message(s) lost` lines in the ROS2 output (and "queue full" stalls in RViz) almost always mean dropped WiFi packets, not a LiDAR fault.

- Keep the robot and the PC close to the WiFi router.
- Use a dedicated 2.4 GHz network, and avoid heavy WiFi use while the robot runs: video streaming, large downloads, cloud sync (OneDrive, for example).
- Check that the PC isn't short of CPU or memory, and that the battery isn't low.
- Keep the ESP32's PCB antenna a few centimetres clear of copper and other boards, above and below, with the antenna/USB end facing outward. Don't sandwich the ESP32 between the LiDAR and the main board. Prefer a module whose antenna sticks out past the board edge.
- If it happens on one robot only, try a known-good ESP32. Some modules transmit poorly (forum #47).

Micro-ROS errors in the Serial Monitor

```
rmw_uros_sync_session() error 1
OK
UTC time Thu Jan  1 00:00:06 1970
rclc_node_init_default(pet) error 1
addROSParams() error 1
...
rclc_executor_spin_some() error 1
rcl_publish(telem_msg) error 300
rclc_executor_spin_some() error 1
...
```

Press the ESP32 reset (EN) button.

B.6 Docker and ROS2 on the PC

`docker run` or `docker pull` fails: "error during connect"

```
docker: error during connect: Head "http://%2F%2F.%2Fpipe%2FdockerDesktopLinuxEngine/_ping": o
pen //./pipe/dockerDesktopLinuxEngine: The system cannot find the file specified.
```

The Docker engine isn't running. Start Docker Desktop and try again.

`docker run` fails: "Ports are not available"

```
docker: Error response from daemon: Ports are not available: exposing port TCP 0.0.0.0:4430 →
0.0.0.0:0: listen tcp 0.0.0.0:4430: bind: An attempt was made to access a socket in a way forb
idden by its access permissions.
```

Reboot your Windows PC.

`docker run` fails: container name "/makerspet" is already in use

```
docker: Error response from daemon: Conflict. The container name "/makerspet" is already in us
e by container "e86601354246...". You have to remove (or rename) that container to be able to
reuse that name.
```

A `makerspet` container is still running from last time. Stop it and run `docker run` again:

```
docker container stop makerspet
```

To avoid this, exit all the container's shells when you finish instead of just closing the PowerShell window.

RViz or another GUI fails to start: "could not connect to display"

```
[rviz2-1] qt.qpa.xcb: could not connect to display host.docker.internal:0.0
[rviz2-1] qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was
found.
...
[ERROR] [rviz2-1]: process has died [pid 76, exit code -6, ...]
```

On Windows, start the X server before launching any GUI. On Ubuntu, check that you used the Ubuntu `docker run` command (Chapter 8), not the Windows one.

I can't get X11 working: view the GUIs over VNC instead

Use VNC (for example, instead of XQuartz on a Mac); TigerVNC and Xfce are in the image. Start the container and the VNC server, then point a VNC viewer at `localhost:5901`:

```
docker run --name makerspet -it --rm -p 8888:8888/udp -p 4430:4430/tcp -p 5901:5901 -e DISPLAY
=:1 kaiaai/kaiaai:iron
# then, inside the container:
vncserver :1 -geometry 1920x1080 -depth 24 -localhost no
```

A ROS2 command doesn't exit even after Ctrl-C

From another Bash window in the container, list processes with `ps -al` and `kill` the stuck `ros2` and `python3` processes by PID (here 334 and 272):

```
root@8ec422cb4258:/ros_ws# ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0   272     1  0  80   0 - 107559 -     pts/0    00:00:00 ros2
4 S     0   334     1  0  80   0 - 210129 x64_sy pts/2   00:00:01 python3
4 R     0   713    66  0  80   0 -  1871 -      pts/1    00:00:00 ps
root@8ec422cb4258:/ros_ws# kill 334
root@8ec422cb4258:/ros_ws# kill 272
```

Windows WSL2 install fails

Follow Microsoft's WSL installation instructions and WSL troubleshooting page. As a last resort, run Ubuntu in a virtual machine (Appendix A); ROS2 will be laggy.

How do I keep my changes when I restart the container?

A fresh container from the image loses changes made inside the old one. Commit the running container to the image; to back it up off your PC, push it to your own Docker Hub repository:

```
docker container commit makerspet kaiaai/kaiaai:iron
# optional backup to your own Docker Hub repo:
docker tag kaiaai/kaiaai:iron <your-docker-hub-user>/kaiaai:iron
docker push <your-docker-hub-user>/kaiaai:iron
```

For the Jazzy image, use `kaiaai/kaiaai:jazzy`.

> **WARNING**
> Don't commit or publish an image that contains secrets, such as your WiFi credentials.

Running more than one robot from one PC

Simplest: one Docker container per robot, each with its own `--name` and host UDP port. Point each robot's WiFi configuration at its port:

```
docker run --name makerspet1 ... -p 8888:8888/udp ... kaiaai/kaiaai:iron
docker run --name makerspet2 ... -p 8889:8888/udp ... kaiaai/kaiaai:iron
```

For true topic namespacing, edit the YAML files in `/ros_ws/src/makerspet_mini/config` (for example `telem.yaml`), Turtlebot3 style. With the Jazzy image, each container gets its own network, so no `--ipc=host` is needed.

Running the ROS2 side on a Raspberry Pi (ARM64)

The Docker image is x86-64 only, so on an ARM64 board such as a Raspberry Pi, install ROS2 natively:

1. Install Ubuntu 22.04 on the Pi.
2. Install ROS2 Iron with `install_ros2_iron_ubuntu_22_04.sh` (or the official ROS2 Iron Ubuntu instructions).
3. Install the Kaia.ai add-ons with `install_kaiaai_iron.sh`.

For ROS2 Jazzy, use Ubuntu 24.04 with `install_ros2_jazzy_ubuntu_24_04.sh` and `install_kaiaai_jazzy.sh`.

> **NOTE**
> The kit's robot has no Raspberry Pi on board; the Pi only replaces the PC.

B.7 LiDAR

Check in order: the LiDAR spins, the ESP32 captures its data, the data reaches ROS2.

The LiDAR doesn't spin

The LiDAR should start spinning once the robot connects to the ROS2 PC. If it doesn't:

- Check the LiDAR wiring.
- Check that your model is on the list of supported LiDAR models.
- Make sure the firmware and ROS2 are both configured for your model (see below). Both default to the LDROBOT LD14P.

Check that the ESP32 captures LiDAR data

With the LiDAR spinning, watch the Serial Monitor telemetry. **LiDAR RPM must not be zero:**

```
Telem avg 46 max 52ms, LiDAR RPM 4.75, wheels RPM 0.00 0.00, battery 8.00V, RSSI -61dBm
Telem avg 46 max 52ms, LiDAR RPM 4.75, wheels RPM 0.00 0.00, battery 8.00V, RSSI -62dBm
```

`LiDAR RPM 0.00` means the ESP32 gets no LiDAR data. Check the wiring and the model configuration.

```
Telem avg 50 max 51ms, LiDAR RPM 0.00, wheels RPM 0.00 0.00, battery 0.99V, RSSI -59dBm
```

> **NOTE**
> Look-alike models aren't necessarily compatible. A builder with an LDROBOT LD14 (not LD14P) saw `LiDAR RPM 0.00` and no scan in RViz; an LD14P fixed it (forum #62).

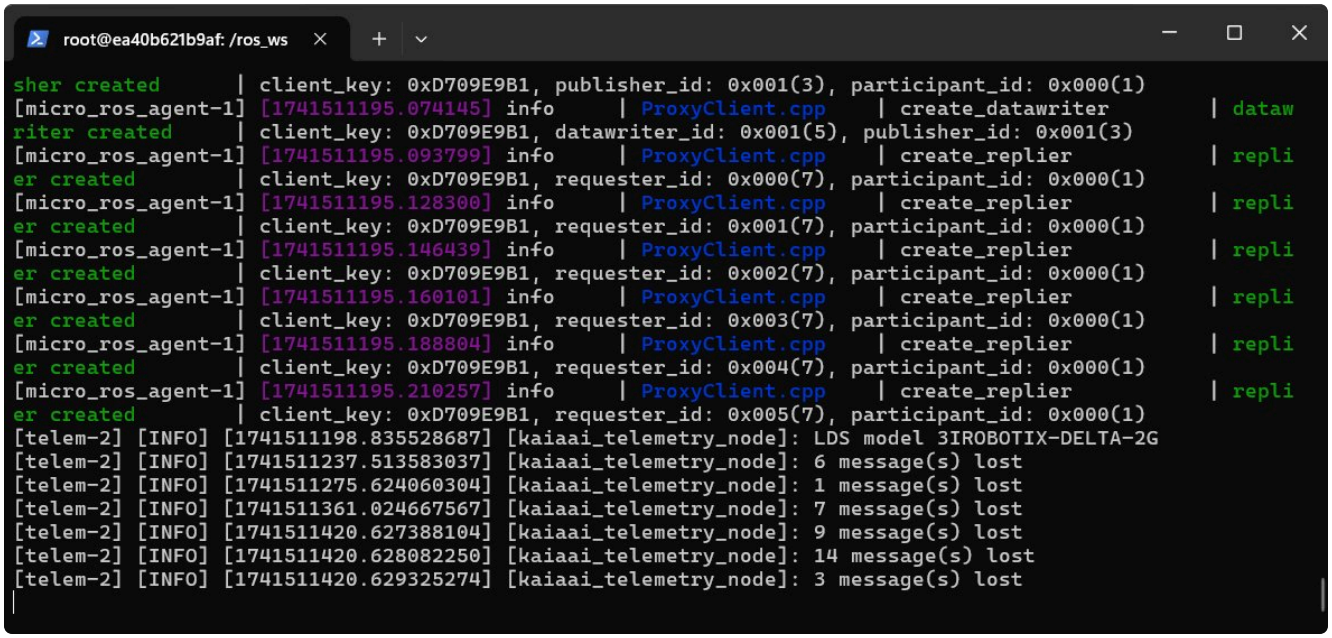
Check that LiDAR data reaches ROS2

Launch RViz:

```
ros2 launch kaiaai_bringup monitor_robot.launch.py
```

You should see live LiDAR points. If not, read the ROS2 terminal output:

- Check the printed LiDAR model (`LDS model ...`). If it isn't yours, the ROS2 side is misconfigured (see below).
- Look for CRC errors and "message(s) lost" lines. Unreliable WiFi causes them (B.5), and so can a wrong LiDAR model in the ROS2 configuration.



The expected LiDAR model is printed; "message(s) lost" means dropped WiFi packets. From forum #25.

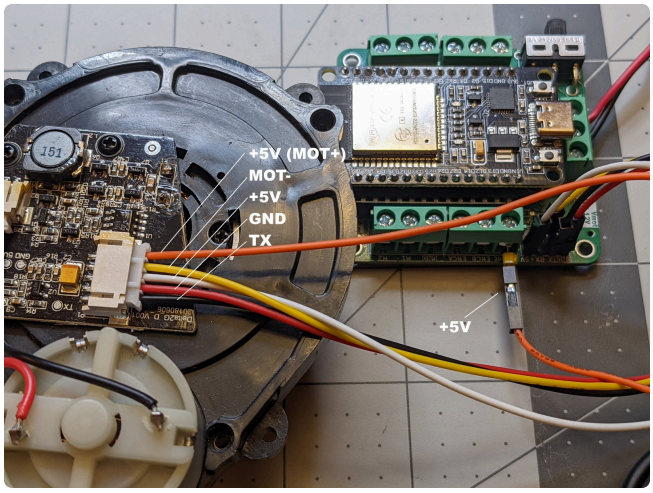
Using a LiDAR other than the LD14P

Change the model in **two** places:

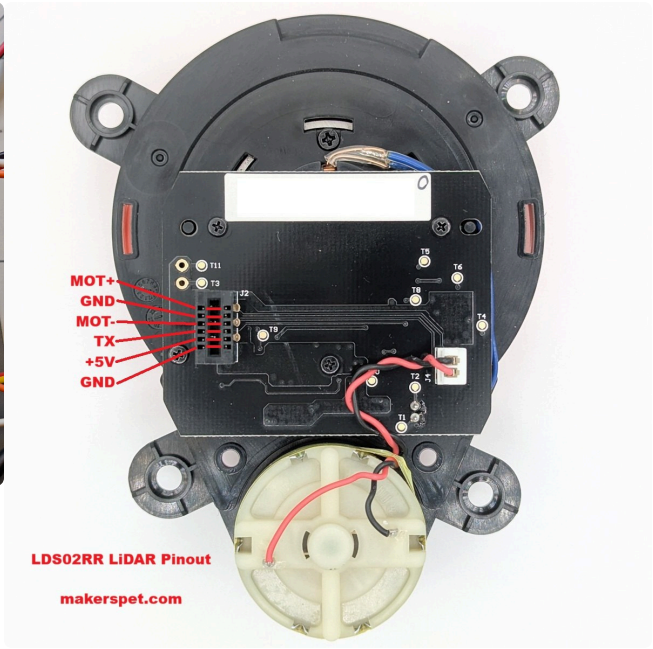
1. **The firmware.** In `config.yaml`'s `lidar:` section, comment out every `model:` line except yours, then upload the sketch data again. At boot, the Serial Monitor should print `LIDAR model` and your model.
2. **ROS2.** In the container, set `lidar_model` in `/ros_ws/src/makerspet_mini/config/telem.yaml` (for example `3IROBOTIX-DELTA-2G`). Commit the container to keep the change (B.6).

(forum #25.) Wiring guides:

- How to Connect YDLIDAR X3, X3PRO, X2, X2L, X4 and SCL to Maker's Pet ESP32 Boards
- How to Connect Delta-2A, 2B and 2G LiDAR to Maker's Pet ESP32 Boards
- How-to: Connect Xiaomi $15 LDS02RR LiDAR to ESP32, Arduino



Delta-2G wired to the BDC-30P LiDAR port. From forum #25.



LDS02RR connector pinout. From forum #45.

> **TIP**
> The Delta-2G's TX outputs 5 V, not 3.3 V. You can put a resistor (100 Ω to 10 kΩ) in series with TX (forum #25).

Cheaper DIY adapter for the LDS02RR

If shipping the official LDS02RR adapter is too expensive, drive the LiDAR motor with a TB6612FNG wired per the LDS02RR-ADPT-V030 schematic. The connector location is in the product photos (and the pinout above, from forum #45), so you don't need to open the housing. The adapter's manufacturing (CAM/CPL) files are no longer released.

Which ESP32 pins for the LiDAR UART? (ESP32-S3, WROVER)

- **ESP32-S3:** hardware serial port 1 on GPIO17 (TX) and GPIO18 (RX).
- **ESP32-WROVER:** GPIO16 and GPIO17 are reserved for PSRAM. Move the LiDAR TX/RX to GPIO32/33.

If the LiDAR shuts off before reaching speed (a motor RPM error), raise `scan_rpm_err_thresh` to give it time to spin up.

B.8 Mapping and navigation

Prepare your place first

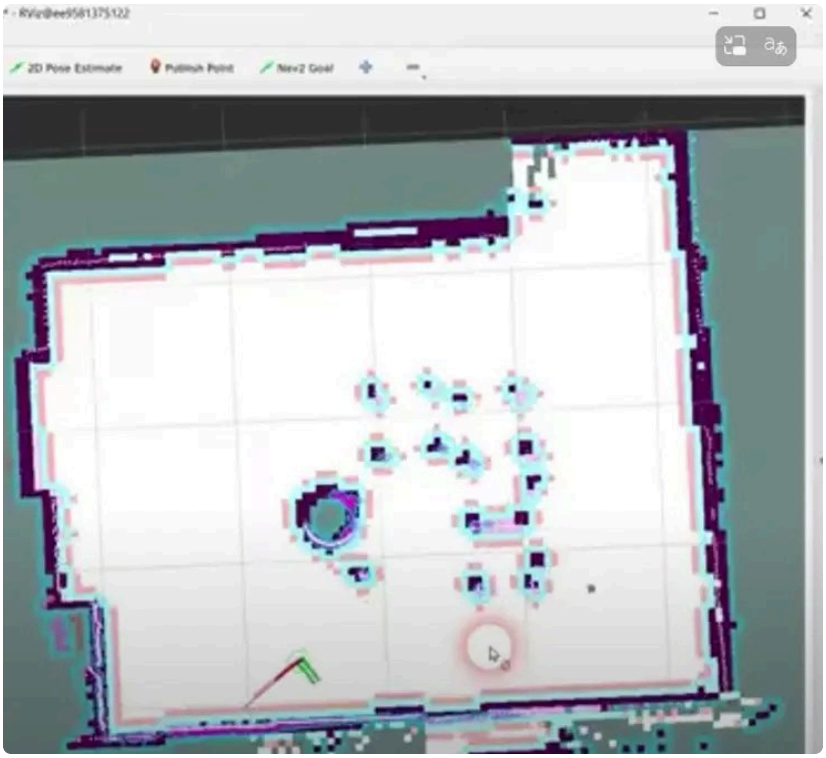
- Block (with cardboard, for example) objects the LiDAR can't detect: black (non-reflective) and transparent objects, objects below the laser plane, very thin objects such as fine sparse mesh,

mirrors and highly reflective objects, and objects lit by near-infrared-rich light (sunlight, incandescent bulbs).
- The robot can't drive over thick carpet or thresholds higher than about 5 mm (1/5").
- Keep the scene still while mapping; don't open or close doors.
- Declutter, and leave plenty of space between obstacles.
- Block off ramps. Sloped flooring tilts the LiDAR's laser plane and messes up the map.

"Goal Failed" errors

The robot can't find a safe path, usually because obstacles are too close together. It refuses a path that passes too close to an obstacle, even without touching it.

- Make sure you have a clean map.
- Remove obstacles, and leave plenty of room between the rest.



A clean map.

A slow PC can also fail navigation. Errors like `Transform data too old when converting from odom to map` mean odometry processing took too long. Close heavy programs, or use a faster PC.

Speed, inflation radius (how far the robot keeps from obstacles) and other settings are in `/ros_ws/src/makerspet_mini/config/navigation.yaml` in the container (on GitHub). For tuning, search "ROS2 Nav2 tuning"; if you find better values, please share them on the forum.

> **NOTE**
> If you've changed the robot's size (your own build), update it in the firmware `config.yaml`, the URDF and possibly `navigation.yaml` (forum #60).

The map is skewed or distorted

Map slowly, both driving straight and turning; fast motion makes SLAM skew the map badly. Check the sensors first with `ros2 launch kaiaai_bringup monitor_robot.launch.py`, then map by driving with teleop. Start with a simple room (one central obstacle).

If the map distorts specifically while the robot moves, also check encoder calibration (next).

The map distorts, or the robot drives off the map (encoder calibration)

The real wheel speed probably doesn't match the commanded speed, usually because the encoder PPR (pulses per revolution) in `config.yaml` is wrong.

1. Read the live values:

   ```
   ros2 param get /pet motor.encoder.ppr
   ```

   Also useful: `motor.left.encoder.now`, `motor.right.encoder.now`, `motor.left.rpm.now`, `motor.right.rpm.now`, `base.wheel.diameter` and `base.wheel.track`.

2. Count the encoder ticks over one full wheel revolution and divide by 4. The firmware multiplies the configured PPR by 4 for the quadrature edges, so enter the 1× value.

3. Set the correct PPR and a Max RPM the motor can actually reach, then upload the sketch data again.

The scan or map slides sideways when the robot drives straight

If the scan or map slides with the robot when driving straight, but turning looks right (and odometry is calibrated), the LiDAR is mounted rotated relative to its URDF frame. Add the matching yaw offset to the laser joint's `rpy`; for a LiDAR mounted 180 degrees round, add 3.14 rad.

The robot moves too slowly during navigation

The Nav2 controller, not teleop, sets navigation speed. In `navigation.yaml`, `controller_server` section, under `FollowPath`:

- `max_vel_x`: top forward speed in m/s (default 0.22), the main one. Raise it gradually, for example to 0.30, and set `max_speed_xy` to the same value.
- `max_vel_theta`: top turning speed in rad/s (default 1.0).
- `acc_lim_x` and `acc_lim_theta`: optionally raise these to reach target speed sooner.

`config/teleop_keyboard.yaml` lists the drivetrain limit as `max_lin_vel: 0.4` m/s and `max_ang_vel: 7.72` rad/s (for the 200 RPM motor, derated to 90%), so keep `max_vel_x` comfortably below 0.4. Higher speed leaves the planner less time to react, so raise values in small steps and test.

Restart the navigation launch after editing. To keep the change, commit the container with the tag of the image you run:

```
docker container commit makerspet kaiaai/kaiaai:iron
# or, if you run the Jazzy image:
docker container commit makerspet kaiaai/kaiaai:jazzy
```

The robot makes a knocking sound

If you 3D-printed the parts, set **Seam Position** to **Random** in your slicer for the caster wheel, or the seam can make it knock. Do the same for the wheels, or the robot may wobble slightly (Appendix D).

B.9 Reference Serial Monitor output

Healthy boot: connected to WiFi and the ROS2 PC

A working robot after a reset (boot ROM lines left out):

```
Kaia.ai firmware version 0.8.0-iron
ESP IDF version v4.4.7-dirty
SPIFFS mounted successfully
/network.yaml found; loaded OK
/config.yaml found; loaded OK
To enter web config push-and-release EN, then push-and-hold BOOT within 1 sec
Board model BDC-30P, version v1.1.0, manufacturer makerspet.com
LIDAR model LDROBOT LD14P
LIDAR RX buffer size 1024, baud rate 230400
Battery ADC attenuation 7.00, voltage 0.99V
Motor driver type IN1_IN2; motor encoder type AB_QUAD
Motor Max RPM 180.00; encoder PPR 1035.00 TPR 4140.00

Connecting to WiFi NETGEAR48 ... connected, IP 192.168.1.5
Connecting to Micro-ROS agent 192.168.1.113 ... success
Syncing time ... OK
UTC time Tue Feb 11 23:02:53 2025
micro-ROS client key 0xDD9181CC; ROS2 node /pet
Micro-ROS initialized
LiDAR info Model: LDROBOT LD14P
startLIDAR() result: OK

Telem avg 106 max 5667ms, LiDAR RPM 0.00, wheels RPM 0.00 0.00, battery 0.99V, RSSI -59dBm
Telem avg 50 max 51ms, LiDAR RPM 0.00, wheels RPM 0.00 0.00, battery 0.99V, RSSI -59dBm
```

This was captured on USB power with the battery off, hence about 1 V and zero LiDAR RPM. With the battery on, expect your battery voltage and a non-zero LiDAR RPM (B.7).

Where it stops tells you where to look:

- Stops at `Setting up WiFi KAIA.AI; browse to http://192.168.4.1`: WiFi setup mode. Connect to `KAIA.AI` and configure it (B.5).
- Repeats `Connecting to WiFi ...`: see "The ESP32 doesn't connect to WiFi" (B.5).
- Repeats `Connecting to Micro-ROS agent ...`: see "The ESP32 connects to WiFi, but not to the ROS2 PC" (B.5).

Firmware uploaded, but no sketch data

The kit's ESP32-E prints:

```
Kaia.ai firmware version 0.8.0-iron
ESP IDF version v4.4.7-dirty
SPIFFS mounted successfully
Sketch data not found. Please upload sketch data.
```

Upload the sketch data (Chapter 3).

Appendix C. Quick Reference

Start the robot software

Windows (PowerShell). Start Docker Desktop and the X server (XLaunch, display number 0) first.

```
docker run --name makerspet -it --rm -v c:\maps:/root/maps -p 8888:8888/udp -p 4430:4430/tcp -
e DISPLAY=host.docker.internal:0.0 -e LIBGL_ALWAYS_INDIRECT=0 kaiaai/kaiaai:iron
```

Ubuntu (terminal).

```
sudo docker run --name makerspet -it --rm -v ~/maps:/root/maps -p 8888:8888/udp -p 4430:4430/t
cp -e DISPLAY -e QT_X11_NO_MITSHM=1 --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" --volume="${XA
UTHORITY}:/root/.Xauthority" kaiaai/kaiaai:iron
```

Open another shell in the running container (on Ubuntu, prefix with `sudo`):

```
docker exec -it makerspet bash
```

Stop a container that is still running (e.g. after closing a window without `exit`):

```
docker container stop makerspet
```

Keep changes you made inside the container:

```
docker container commit makerspet kaiaai/kaiaai:iron
```

ROS 2 commands (inside the container)

Task	Command
Connect to the physical robot	`ros2 launch kaiaai_bringup physical.launch.py`
Drive with the keyboard	`ros2 run kaiaai_teleop teleop_keyboard`
View the LiDAR scan in RViz	`ros2 launch kaiaai_bringup monitor_robot.launch.py`
Map while driving (SLAM + Nav2)	`ros2 launch kaiaai_bringup navigation.launch.py slam:=True`
Explore automatically	`ros2 launch explore_lite explore.launch.py`
Save the map	`ros2 run nav2_map_server map_saver_cli -f ~/maps/map --ros-args -p save_map_timeout:=60.0`
Navigate on a saved map	`ros2 launch kaiaai_bringup navigation.launch.py map:=$HOME/maps/map.yaml`
Start the simulation (Gazebo)	`ros2 launch kaiaai_gazebo world.launch.py`
Map in simulation	`ros2 launch kaiaai_bringup navigation.launch.py use_sim_time:=true slam:=True`
Read a robot parameter	`ros2 param get /pet motor.encoder.ppr`

More commands are in the command cheat sheets at github.com/kaiaai/kaiaai; the container prints this link when it starts.

Teleop keys

Key	Action
`w` / `x`	Increase / decrease forward speed (keep pressing `x` to reverse)
`a` / `d`	Turn left / right faster
`s`	Stop turning, keep going straight
`Space`	Stop
`Ctrl-C`	Quit

ESP32 activity LED

LED	Meaning
Solid on	WiFi configuration (AP) mode: connect to the `KAIA.AI` network and browse to http://192.168.4.1
Slow blink, once per second	Connecting to WiFi
Very slow blink, once per 20 seconds	Connected to WiFi, connecting to the ROS 2 PC
Fast blink	Connected to the ROS 2 PC

Reset the WiFi settings: press and release the ESP32 reset (EN) button, then within one second press and hold BOOT until the LED turns solid on. Release BOOT; the ESP32 restarts in WiFi configuration mode.

Versions used in this book

Item	Version
Arduino IDE	1.8.19 (recommended) or 2.x
ESP32 board package (esp32 by Espressif)	2.0.17; avoid 2.0.15 and 3.x
Robot firmware	Kaia.ai 0.8.x (`-iron`)
ROS 2	Iron, Docker image `kaiaai/kaiaai:iron` (Jazzy: `kaiaai/kaiaai:jazzy`)
PC operating system	Windows 10/11 with WSL2, or Ubuntu 22.04

Links

Instructions and help

- Video course playlist: youtube.com/playlist?list=PLOSXKDW70aR8uA1IFahSKVuk5ODDfjTZV
- Up-to-date build guide and FAQ: makerspet.com/blog/bld-120mm-pack
- Simulation tutorial: makerspet.com/blog/tutorial-map-navigate-ros2-robot-in-simulation
- Support forum: github.com/makerspet/support/discussions
- Discord: discord.gg/3y2JKz5T25

Software and configuration

- Robot firmware: github.com/kaiaai/firmware
- ROS 2 packages and command cheat sheets: github.com/kaiaai/kaiaai

- Robot model, navigation and teleop settings: github.com/makerspet/makerspet_mini
- Default `config.yaml` (BDC-30P): github.com/kaiaai/firmware/.../data/config.yaml
- Configuration file reference: blog.kaia.ai/kaiaai-configuration-file
- Native install scripts (no Docker): github.com/kaiaai/install

Other LiDARs

- YDLIDAR X3, X3PRO, X2, X2L, X4 and SCL
- Delta-2A, 2B and 2G
- Xiaomi LDS02RR

3D printing

Print files and the Fusion 360 model: the "Download Files for 3D Printing" section of makerspet.com/blog/bld-120mm-pack. See Appendix D.

Appendix D. 3D Print the Robot Parts

The robot's plastic parts are open source. If you have a 3D printer, you can print them instead of buying them. The electronics, motors and hardware still come from the build pack or your own sources.

Get the files

Download the files from the "Download Files for 3D Printing" section of the build guide at makerspet.com/blog/bld-120mm-pack. That page always has the latest versions:

- **3MF files** for the base, LiDAR skirts, LiDAR posts, board posts, caster roller, caster mounts and the 2S battery holder backstop
- **3MF files** for the wheels
- **3MF files** for the motor clamps
- The full **Autodesk Fusion 360** model (`.f3d`), if you want to modify the design

The base takes the Maker's Pet ESP32-E dev kit module.

Filament

Print the parts in **PETG**. PLA works too, but it softens at a much lower temperature: a PLA robot left in the sun, or in a hot car, can warp or melt.

Slicer settings

Set **Seam Position** to **Random** in your slicer for the **caster roller** and the **wheels**. With an aligned seam, the caster roller knocks rhythmically as it rolls and the wheels make the robot wobble slightly.

Parts you don't print

From the build pack list in Chapter 1, these aren't printed:

- BDC-30P driver board and Maker's Pet ESP32-E 30-pin dev kit module
- LDROBOT LD14P LiDAR and its breakout cable
- Two N20 gear motors with encoders, and two motor cables
- Two tires and the roller's metal shaft
- Battery case for 6 AA batteries
- M3 screws: countersunk and hex button

Optional: 18650 battery holder mod

The 2×18650 mod uses two printed parts from the same file set: the **battery holder backstop** and the **wider LiDAR skirt**. Chapter 1 shows how to assemble it, with the safety warnings for rechargeable cells.