**You said:**
How would I install Gazebo on MacOS?

**ChatGPT said:**
Installing Gazebo on macOS depends on which version of Gazebo you want (the classic Gazebo vs. the newer Gazebo Fortress / Harmonic versions under the Gazebo Sim project).

Here's a complete guide for both approaches:

---

## 🧭 Option 1: Install Gazebo (Classic) on macOS

Gazebo Classic is older but still used in many robotics projects (especially with ROS 1).

### 1. Install Homebrew

If you don't already have Homebrew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

### 2. Install Gazebo Classic

Gazebo Classic versions can be installed via Homebrew:

```
brew install gazebo
```

To install a specific version (for example Gazebo 11):

```
brew install gazebo11
```

### 3. Verify Installation

Once installed:

```
gazebo
```

This should open the Gazebo GUI. If it doesn't launch, try running it with:

```
/opt/homebrew/bin/gazebo
```

---

## 🚀 Option 2: Install Gazebo Sim (Fortress / Harmonic) on macOS

The new generation of Gazebo (rebranded under Gazebo Sim, previously known as Ignition Gazebo) supports macOS but requires a few dependencies.

### 1. Tap the Gazebo Formula Repository

```
brew tap osrf/simulation
```

### 2. Install a Gazebo Sim Release

You can install the latest version:

```
brew install gz-harmonic
```

Or an earlier one, e.g. Fortress:

```
brew install gz-fortress
```

### 3. Check Your Installation

To verify:

```
gz sim
```

If everything is configured correctly, the Gazebo Sim GUI will launch.

---

## 🧩 Optional: ROS Integration (ROS 2)

If you're using ROS 2 (e.g. Humble, Iron, Jazzy), Gazebo Sim integrates smoothly.

You'll usually add:

```
brew install ros-humble-desktop
```

and then install the Gazebo packages for ROS:

```
sudo apt install ros-humble-gazebo-ros-pkgs
```

*(Note: The above `apt` command is Linux-only; on macOS you can build these packages from source using colcon.)*

---

## ⚠️ Common Issues on macOS

- OpenGL / GUI Crashes: Gazebo's rendering sometimes fails under macOS ARM (M1/M2). Run with `--render-engine ogre` or use a headless mode (`gz sim -r`).
- Missing Paths: Add Gazebo binaries to your shell PATH:

```
echo 'export PATH="/opt/homebrew/bin:$PATH"' >> ~/.zshrc source ~/.zshrc
```

---

## ✅ Quick Summary

| Version | Command | GUI Command | Notes |
|---|---|---|---|
| Gazebo Classic | `brew install gazebo` | `gazebo` | Stable, older |
| Gazebo 11 | `brew install gazebo11` | `gazebo` | Most used with ROS 1 |
| Gazebo Sim (Fortress) | `brew install gz-fortress` | `gz sim` | New generation |
| Gazebo Sim (Harmonic) | `brew install gz-harmonic` | `gz sim` | Latest |

---

Would you like me to tailor the setup for Apple Silicon (M1/M2/M3) or Intel Macs? The steps differ slightly in dependency handling.

**You said:**
How can I check if i have Homebrew installed?

**ChatGPT said:**
You can check if Homebrew is installed on your macOS system in a few simple ways:

---

## 🧩 Option 1: Check from Terminal

Open your Terminal app and run:

```
brew --version
```

- ✅ If Homebrew is installed, you'll see something like:

```
Homebrew 4.3.11
```

- 
- ❌ If you get:

```
zsh: command not found: brew
```

- that means Homebrew isn't installed (or it's not on your PATH).

---

## 🧭 Option 2: Check Installation Path

Try checking if the Homebrew directory exists:

For Apple Silicon (M1/M2/M3) Macs:

```
ls /opt/homebrew
```

For Intel Macs:

```
ls /usr/local/Homebrew
```

If one of these directories exists and contains files, you likely have Homebrew installed.

---

## 🧠 Option 3: Use the "which" Command

This tells you where `brew` is located:

```
which brew
```

- Typical output (Apple Silicon): `/opt/homebrew/bin/brew`
- Typical output (Intel): `/usr/local/bin/brew`
- No output → Homebrew not installed

---

Would you like me to show you how to install it if it's missing (and automatically detect Apple Silicon vs Intel)?

**You said:**

When running the install Homebrew command, it asks for if I have 'sudo' access and a password.

**ChatGPT said:**

Yep — that's totally normal 👍

Here's what's happening and what you should do 👇

---

## 🧩 Why Homebrew Asks About `sudo` and Your Password

When you install Homebrew using this command:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

…the script needs to:

- Create directories like `/usr/local/Homebrew` (on Intel) or `/opt/homebrew` (on Apple Silicon),
- Set permissions, and
- Add Homebrew to your system's PATH.

Those steps require administrator privileges, which is why macOS asks for your sudo password.

---

## 🔐 What to Do

When prompted like this:

```
Press RETURN to continue or any other key to abort
==> Checking for `sudo` access (which may request your password)...
Password:
```

✅ Do this:

1. Press Return to continue.
2. When it says `Password:`, type your Mac user password (the one you use to log in or unlock settings).

- You won't see any characters appear (not even `*`), but that's normal — macOS hides password input for security.

3. Press Return again.

---

### 🧠 Notes

- Homebrew only uses `sudo` *once* during installation to set permissions.
- After it's installed, you don't need `sudo` to use it — all installs and updates run in your user space.

---

If you'd like, I can walk you through the expected Homebrew install output step-by-step so you can confirm it's proceeding safely. Would you like that?

**You said:**
Even after installing Homebrew, it still claims that the brew command is not found

**ChatGPT said:**
Good question — that's a very common issue on macOS, especially on Apple Silicon (M1/M2/M3) Macs.

It usually means Homebrew installed correctly, but your shell PATH doesn't include the directory where `brew` lives. Let's fix that step-by-step 👇

---

## 🧠 Step 1: Check Where Homebrew Was Installed

Run these commands one at a time:

```
ls /opt/homebrew/bin/brew ls /usr/local/bin/brew
```

- If you see a file listed under `/opt/homebrew/bin/brew` → you're on Apple Silicon (M1/M2/M3).
- If it's under `/usr/local/bin/brew` → you're on Intel.
- If neither exists, the install didn't complete — we can reinstall safely afterward.

---

## ⚙️ Step 2: Add Homebrew to Your Shell PATH

Depending on which shell you're using (macOS uses zsh by default now):

### For Apple Silicon (M1/M2/M3):

Run:

```
echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprofile eval "$(/opt/homebrew/bin/brew shellenv)"
```

### For Intel Macs:

Run:

```
echo 'eval "$(/usr/local/bin/brew shellenv)"' >> ~/.zprofile eval "$(/usr/local/bin/brew shellenv)"
```

This adds Homebrew to your PATH permanently.

---

## ✅ Step 3: Verify Installation

Now test it:

```
brew --version
```

If you see something like:

```
Homebrew 4.3.11
```

then you're good to go 🎉

---

## 🧩 Step 4 (Optional): Re-run the Installer if Needed

If `brew` still isn't found and Step 1 showed no `brew` binary at all, reinstall it cleanly:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Then repeat Step 2.

---

Would you like to tell me the output of:

```
ls /opt/homebrew/bin/brew
```

and

```
ls /usr/local/bin/brew
```

?

I can then tell you exactly which PATH command to use for your setup.

**You said:**
Now the 'brew install gazebo' command is not working, and is claiming that there is no available formula of the name gazebo. I am running on Homebrew version 4.6.17.

**ChatGPT said:**
That error ("no available formula named `gazebo`") suggests that Homebrew doesn't currently have a `gazebo` formula in its default repositories. Gazebo (classic) is provided via a special "tap" from OSRF (Open Source Robotics Foundation).

Here's what you should try:

---

## ✅ Step-by-step: Tap the OSRF repo, then install a specific Gazebo version

1. Tap the OSRF Homebrew simulation repository

   This adds robotics-related formulae (including Gazebo) to your Homebrew.
2. `brew tap osrf/simulation brew update`
3. Install a specific Gazebo version

   For the "classic" Gazebo, the common version is gazebo11.
4. `brew install gazebo11`
   Alternatively, newer "Gazebo Sim / Ignition" versions are named differently under Homebrew, e.g. `gz-jetty` for the newer Jetty release.
5. Gazebo

6. For example:
7. `brew install gz-jetty`
8. Run it

   After installation, try:
9. `gazebo`

   or
10. `gz sim`

---

## ⚠️ Things to watch out for / possible issues

- On macOS 14 / 15 / Apple Silicon, the Gazebo Classic formula is deprecated / EOL (end-of-life), so support might break.
- Gazebo
- +2
- Robotics Stack Exchange
- +2
- Some of the newer Gazebo versions require you to install via their “gz-sim” naming scheme.
- Gazebo
- +2
- Gazebo
- +2
- On Apple Silicon, you may need to go through additional compatibility steps or use Rosetta for parts of the build toolchain. Users have reported issues around Ogre and library linking.
- Robotics Stack Exchange
- +3
- Reddit
- +3
- Open Robotics Discourse
- +3

---

If you tell me which macOS version you're on (e.g. Sonoma 14, Sequoia 15) and whether your Mac is Intel or Apple Silicon, I can give you the exact commands for your setup. Do you want me to tailor the commands for your system?

**You said:**
What is QGroundControl?

**ChatGPT said:**
QGroundControl (often abbreviated QGC) is a ground control station (GCS) software — basically, a mission planner and control interface for drones and other unmanned vehicles.

It's open source, cross-platform (works on macOS, Windows, Linux, iOS, and Android), and is most commonly used with flight stacks like PX4 and ArduPilot.

---

## 🧭 What It Does

QGroundControl provides a full set of tools to:

- Connect to a drone over USB, Wi-Fi, or telemetry radio
- Configure flight controllers (calibrate sensors, tune parameters, set flight modes)
- Plan missions using a map interface (set waypoints, camera actions, etc.)
- Monitor live telemetry — altitude, battery, GPS, attitude, etc.
- Control the vehicle manually or autonomously
- Record flight logs and view data

Essentially, it's the "cockpit" software that lets you see and control what your drone is doing in real time.

---

## ✈️ Supported Systems

| Flight Stack | Supported? |
|---|---|
| PX4 | ✅ Officially supported and maintained |

| | |
|---|---|
| ArduPilot | ✅ Supported (with some feature differences) |
| MAVLink | ✅ Core communication protocol |

QGroundControl uses the MAVLink communication protocol — the same one that PX4 and ArduPilot use — to talk to vehicles.

---

## 💻 Key Features

- Mission planning with drag-and-drop waypoints
- Camera control for photo/video missions
- Map integration (Bing, Google, or custom tiles)
- Firmware flashing for PX4/ArduPilot boards
- Parameter tuning (PID gains, flight modes, etc.)
- Safety and failsafe settings
- Joystick/gamepad input support
- Log file analysis

---

## 🧩 Installation

On macOS, you can install it easily:

### Option 1: Direct Download

Download the latest stable release here:

👉 https://qgroundcontrol.com/downloads/

Then drag the `QGroundControl.app` to your Applications folder.

### Option 2: Homebrew (less common)

```
brew install --cask qgroundcontrol
```

---

## 🚀 Typical Use Case

If you're working with PX4 and Gazebo, you can:

1. Simulate your drone in Gazebo.
2. Connect QGroundControl to the simulator via UDP.
3. Monitor and control the simulated drone in real time.

This setup is widely used in research, development, and education for autonomous UAV projects.

---

Would you like me to show you how to connect QGroundControl to a PX4/Gazebo simulation on macOS? That's a common next step once you have both installed.

**You said:**
How would I download PX4_SITL?

**ChatGPT said:**
Excellent — you're diving into the right part of the PX4 ecosystem. 🚀

Let's walk through exactly how to download and run PX4 SITL (Software-In-The-Loop) on macOS.

---

## 🧠 What PX4 SITL Is

PX4 SITL is a simulation mode of the PX4 autopilot that runs entirely on your computer — no hardware required.

It simulates the flight controller and connects to a simulated environment (like Gazebo, AirSim, or JMAVSim) and QGroundControl.

---

## 🧩 Step-by-Step: Install PX4 SITL on macOS

### ✅ Step 1: Install Dependencies

Open Terminal and run:

```
brew install git cmake ninja python@3.11 brew install px4-dev
```

ℹ️ If `px4-dev` isn't available, we'll handle the build manually in Step 3.

Make sure you also have QGroundControl installed — it helps visualize and control your simulated drone.

---

## ✅ Step 2: Clone the PX4 Source Code

```
cd ~ git clone https://github.com/PX4/PX4-Autopilot.git --recursive cd PX4-Autopilot
```

The `--recursive` flag ensures you also pull in all the submodules (PX4 has quite a few).

---

## ✅ Step 3: Install PX4 Python Tools (used for SITL)

PX4 provides helper tools through `pip`. Run:

```
python3 -m pip install --upgrade pip python3 -m pip install -r Tools/setup/requirements.txt
```

---

## ✅ Step 4: Build and Run the PX4 SITL Simulator

You can choose from several simulation backends:

| Simulator | Command | Description |
|---|---|---|
| JMAVSim (lightweight) | `make px4_sitl jmavsim` | Default and fastest |
| Gazebo Classic | `make px4_sitl gazebo-classic` | 3D environment (deprecated) |
| Gazebo Harmonic / Fortress | `make px4_sitl gz_sim` | New Gazebo Sim integration |

Example (JMAVSim):

```
make px4_sitl jmavsim
```

PX4 will compile, launch a small 3D simulator window, and start sending telemetry over UDP.

You can then open QGroundControl, and it will auto-connect to the simulated drone. 🛰️

---

## ✅ Step 5 (Optional): Connect QGroundControl

Open QGroundControl.app, and it should automatically detect the PX4 SITL over UDP on port 14550.

You'll see:

- A map view with the drone's position,
- Telemetry data (altitude, GPS, etc.),
- Parameter configuration options.

---

## ⚙️ Optional: Using Gazebo Sim Instead of JMAVSim

If you installed Gazebo Harmonic (the new Gazebo Sim), you can use:

```
make px4_sitl gz_sim
```

You'll get a richer environment with physics, 3D terrain, and multiple drones.

---

## 🧩 Troubleshooting

| Issue | Fix |
|---|---|
| `command not found: make` | Install build tools: `xcode-select --install` |
| `brew install px4-dev` fails | Just skip it — the dependencies are covered by CMake + pip |
| Gazebo not found | Install via Homebrew: `brew install gz-harmonic` |

| | |
|---|---|
| QGroundControl not connecting | Make sure UDP 14550 is not blocked (firewall off or allowed) |

---

## ✅ Quick Summary

| Step | Command |
|---|---|
| Clone PX4 | `git clone https://github.com/PX4/PX4-Autopilot.git --recursive` |
| Install deps | `brew install git cmake ninja python@3.11` |
| Setup Python tools | `pip install -r Tools/setup/requirements.txt` |
| Run SITL (JMAVSim) | `make px4_sitl jmavsim` |
| Run SITL (Gazebo Sim) | `make px4_sitl gz_sim` |

---

Would you like me to show you how to set it up with Gazebo Sim and QGroundControl connected automatically, so you can test a virtual drone flight from takeoff to landing?

**You said:**
Currently it displays the error ninja: error: unknown target 'gz_sim' even though I have Gazebo Classic installed.

**ChatGPT said:**
Excellent debugging — you've just uncovered an important detail about PX4 and Gazebo naming. 👇

Let's clear it up carefully:

---

## 🧠 Why You See

```
ninja: error: unknown target 'gz_sim'
```

That happens because:

- The target `gz_sim` refers to the new Gazebo (called *Gazebo Sim*, or *Ignition Gazebo*).
- You said you have Gazebo Classic installed — that's the *older* version (often `gazebo11`).
- PX4 uses different make targets for Classic vs. the new Gazebo Sim.

So PX4 doesn't recognize `gz_sim` because your setup only supports Gazebo Classic, not the new Gazebo Sim.

---

## ✅ Fix: Use the Correct PX4 Target for Gazebo Classic

Instead of:

```
make px4_sitl gz_sim
```

Use:

```
make px4_sitl gazebo-classic
```

or simply:

```
make px4_sitl gazebo
```

(both work depending on PX4's branch version — in newer PX4 releases it's `gazebo-classic`.)

This will:

- Launch PX4 SITL
- Start Gazebo Classic
- Spawn a default quadrotor (e.g., `iris`)
- Open Gazebo's 3D window

---

## 🧩 Optional: Verify PX4 Sees Gazebo

After you run:

```
make px4_sitl gazebo-classic
```

you should see console output like:

```
[INFO] [px4] Starting SITL Gazebo Classic...
INFO  [simulator] Waiting for simulator to connect on TCP port 4560
INFO  [mavlink] MAVLink on UDP port 14560 started
```

Then, QGroundControl should automatically detect the drone within a few seconds.

---

## ⚙️ If You Want to Use the New Gazebo (Gazebo Sim)

If you eventually want the newer Gazebo Sim (Ignition / Harmonic) instead of Gazebo Classic:

1. Install it via Homebrew:
2. `brew tap osrf/simulation brew install gz-harmonic`
3. Then use the newer PX4 build target:
4. `make px4_sitl gz_sim`

…but that's optional — PX4 still works great with Gazebo Classic for now.

---

## ✅ Summary

| Simulator | PX4 Command | Notes |
|---|---|---|
| JMAVSim | `make px4_sitl jmavsim` | Lightweight, simple 3D |
| Gazebo Classic | `make px4_sitl gazebo-classic` | Works with `gazebo11` |
| Gazebo Sim (Harmonic/Fortress) | `make px4_sitl gz_sim` | Requires `gz-*` installed via Homebrew |

---

Would you like me to show you how to spawn a specific drone model (like a VTOL or rover) inside your Gazebo Classic simulation?

**You said:**
Currently there is the same error for both gazebo-classic and gazebo.

**ChatGPT said:**
Perfect — that gives us a good clue about what's going on.

If PX4 is saying

```
ninja: error: unknown target 'gazebo' (or 'gazebo-classic')
```

it means your PX4 build environment doesn't have Gazebo support configured — not that Gazebo itself is broken.

Let's fix that properly 👇

---

## 🧠 What's Happening

PX4 uses CMake build targets (like `gazebo-classic`, `gz_sim`, or `jmavsim`) that are only available if the required simulator is detected at build configuration time.

If Gazebo Classic isn't installed where PX4 expects, those targets won't be generated — so `ninja` reports them as "unknown".

---

## ✅ Step-by-Step Fix for macOS

### 1. Confirm Which Gazebo You Installed

Run this:

```
gazebo --version
```

You should see something like:

```
Gazebo multi-robot simulator, version 11.13.0
```

```
```

If you get “command not found,” then Gazebo isn’t correctly installed via Homebrew yet.

Install it with:

```
brew tap osrf/simulation brew install gazebo11
```

Then confirm again with `gazebo --version`.

---

## 2. Install Gazebo Development Headers

PX4 needs Gazebo’s headers and libraries to build the integration. On macOS,

Homebrew usually installs them under:

- `/opt/homebrew/include/gazebo-11/`
- `/opt/homebrew/lib/`

If you installed via `brew install gazebo11`, you’re good — but if not, reinstall with:

```
brew reinstall gazebo11
```

---

## 3. Clean and Rebuild PX4

Let’s ensure PX4 reconfigures and picks up the Gazebo Classic install.

From your PX4 root directory:

```
cd ~/PX4-Autopilot make clean
```

Then rebuild with explicit target:

```
make px4_sitl gazebo
```

If you still see the same error, we can manually verify available build targets next.

---

## 4. Check Which Targets PX4 Sees

Run:

```
make help | grep gazebo
```

If the output is empty — PX4 doesn't recognize any Gazebo targets.

That means the CMake config couldn't find Gazebo headers or CMake modules.

---

### 5. Set Environment Variables (if needed)

You can tell PX4 where Gazebo is installed manually:

```
export GAZEBO_INSTALL_PATH=/opt/homebrew export GAZEBO_VERSION=11
```

Then re-run:

```
make px4_sitl gazebo
```

---

### 6. (Alternative) Use JMAVSim to Confirm PX4 Itself Works

To ensure your PX4 setup is otherwise fine, run:

```
make px4_sitl jmavsim
```

If JMAVSim works, your PX4 SITL setup is good — it's just the Gazebo detection that's off.

---

## ✅ Summary

| Problem | Likely Cause | Fix |
|---|---|---|
| `unknown target 'gazebo'` | Gazebo not installed or not found | `brew install gazebo11` |
| Still missing | PX4 not detecting Gazebo headers | `export GAZEBO_INSTALL_PATH=/opt/homebrew` |
| JMAVSim works | PX4 installed correctly | Continue debugging Gazebo path |

---

If you'd like, you can paste the output of:

```
gazebo --version make help | grep gazebo
```

and I'll tell you *exactly* why PX4 doesn't see Gazebo on your setup (and what line to add). Would you like to do that?

**You said:**
The terminal displays that ignition-msgs5's formula is deprecated when I try to install gazebo11.

**ChatGPT said:**
Excellent observation — and you've just hit a known issue with installing Gazebo Classic (gazebo11) on modern macOS and recent Homebrew versions (like 4.6.x).

Let's unpack what's going on and how to fix it safely. 👇

---

## 🧠 Why You're Seeing

```
Warning: ignition-msgs5 has been deprecated because it is unmaintained
```

Gazebo Classic (v11) depends on older Ignition libraries (like `ignition-msgs5`, `ignition-transport8`, etc.), but:

- These "Ignition" libraries have since been renamed and replaced by the Gazebo Sim (gz) series (e.g., `gz-msgs`, `gz-transport`).
- Homebrew recently deprecated many of the old `ignition-*` formulas.
- As a result, the Gazebo Classic formula (`gazebo11`) is now partially broken or unmaintained on Homebrew 4.6+ — especially on Apple Silicon Macs.

So, the deprecation warning (and often build failure) means Gazebo Classic isn't a reliable install on current macOS anymore.

---

## ✅ Options to Fix or Work Around It

### 🟢 Option 1: Use the New Gazebo Sim (Harmonic / Jetty)

The modern version of Gazebo, now called Gazebo Sim, fully replaces Gazebo Classic.

This is the recommended path forward (supported by PX4 as `gz_sim`).

Run:

```
brew tap osrf/simulation brew install gz-harmonic
```

Then test:

```
gz sim
```

If you see a GUI window open, you're good to go.

You can then build PX4 with:

```
make px4_sitl gz_sim
```

🔧 If you previously got "unknown target 'gz_sim'," that's because PX4 didn't detect Gazebo Sim — installing `gz-harmonic` fixes that.

---

## 🟠 Option 2: Still Want Gazebo Classic?

You *can* still try, but you'll likely hit more dependency warnings or build errors.

To force installation despite deprecation:

```
brew install gazebo11 --ignore-dependencies
```

Then manually install dependencies (not recommended unless you need an exact environment for research or an old PX4 version).

---

## 🔵 Option 3: Use JMAVSim (Lightweight Alternative)

If you just want to run PX4 SITL quickly without 3D Gazebo simulation:

```
make px4_sitl jmavsim
```

JMAVSim doesn't depend on any of the Gazebo or Ignition libraries and works cleanly on macOS.

---

## ✅ Recommended Path (for macOS, 2025)

| Simulator | Status | Command | Notes |
|---|---|---|---|
| Gazebo Classic | ⚠️ Deprecated | `brew install gazebo11` | Broken deps on modern Homebrew |
| Gazebo Sim (Harmonic) | ✅ Supported | `brew install gz-harmonic` | Works with `make px4_sitl gz_sim` |
| JMAVSim | ✅ Simple & stable | `make px4_sitl jmavsim` | Good for quick testing |

---

### TL;DR

🚫 gazebo11 → deprecated and breaks on macOS

✅ gz-harmonic → modern, supported Gazebo Sim

✅ jmavsim → easiest way to run PX4 SITL quickly

---

Would you like me to show you how to reconfigure PX4 to recognize `gz-harmonic` and verify that `make px4_sitl gz_sim` runs correctly on your Mac?

**You said:**
How can I check if gazebo sim is installed?

**ChatGPT said:**
Good question — and it's important to verify before PX4 can use it.

Here's how to check whether Gazebo Sim (the new "gz" version) is installed on your macOS system 👇

---

## 🧩 Step 1: Try the Command

Open your Terminal and run:

```
gz --version
```

### ✅ If Gazebo Sim is installed

You'll see output like:

```
gz 8.1.0 (Harmonic)
```

or

```
gz 9.0.0 (Jetty)
```

That confirms Gazebo Sim (Harmonic, Jetty, or a similar version) is correctly installed and available in your PATH.

### ❌ If you see:

```
zsh: command not found: gz
```

then Gazebo Sim isn't installed, or your PATH isn't set correctly.

---

## 🧩 Step 2: Check the Installed Package (via Homebrew)

Run:

```
brew list | grep gz-
```

If installed, you'll see something like:

```
gz-harmonic
gz-math8
gz-msgs9
gz-transport14
gz-rendering8
gz-sim8
```

Those are the component libraries that make up Gazebo Sim.

If nothing shows up, it's not installed.

---

## 🧩 Step 3: Check the Install Location (Apple Silicon Default)

On Apple Silicon Macs, Gazebo Sim binaries usually live here:

```
/opt/homebrew/bin/gz
```

Try:

```
ls /opt/homebrew/bin/gz
```

If you see a file there, it's installed — but you may need to add Homebrew's bin directory to your PATH.

Add this line to your shell config (`~/.zprofile` or `~/.zshrc`):

```
eval "$(/opt/homebrew/bin/brew shellenv)"
```

Then reload your terminal:

```
source ~/.zprofile
```

---

## 🧩 Step 4: (Optional) Verify It Launches

Run:

```
gz sim
```

- ✅ If the Gazebo GUI opens → you're all set.
- ⚠️ If you get an error like "command not found" or "missing plugin," installation is incomplete or missing dependencies.

---

## ✅ TL;DR

| Check | Command | Expected Output |
|---|---|---|
| Version | `gz --version` | `gz 8.x (Harmonic)` |

| | | |
|---|---|---|
| Homebrew | `brew list | grep gz-` |
| Path | `ls /opt/homebrew/bin/gz` | Shows gz binary |
| Run GUI | `gz sim` | Opens window |

---

Would you like me to show you the correct install command for Gazebo Sim Harmonic and how to make sure PX4 can find it (`make px4_sitl gz_sim`)?

**You said:**

The terminal displays "make: *** No rule to make target `px4_sitl'.  Stop." when I run "make px4_sitl gz_sim" in the terminal

**ChatGPT said:**

Perfect — that's a really informative error message.

It means PX4's build system isn't initialized correctly or you're not in the PX4 source directory when you run the `make` command. Let's fix this step by step 👇

---

## 🧠 Why You're Seeing

```
make: *** No rule to make target `px4_sitl'.  Stop.
```

This happens when:

1. You're not inside the PX4 source directory (`PX4-Autopilot`),
2. Or the PX4 submodules weren't cloned (missing `Tools/`, `platforms/`, etc.),
3. Or you're running `make` in the wrong shell environment (missing `cmake` config).

---

## ✅ Step 1: Verify You're in the PX4 Source Directory

Run:

```
pwd
```

If it doesn't end with something like:

```
/Users/<yourname>/PX4-Autopilot
```

then you're in the wrong directory.

To fix that:

```
cd ~/PX4-Autopilot
```

Then try again:

```
make px4_sitl gz_sim
```

---

## ✅ Step 2: Ensure the Repo Was Cloned with Submodules

If PX4 was cloned without `--recursive`, some build files (like `Makefile`, `Tools/`, etc.) will be missing.

Check with:

```
ls
```

You should see folders like:

```
Makefile  Tools/  src/  platforms/  ROMFS/
```

If `Tools/` or `Makefile` are missing, re-clone PX4 properly:

```
cd ~ rm -rf PX4-Autopilot git clone
https://github.com/PX4/PX4-Autopilot.git --recursive cd PX4-Autopilot
```

---

## ✅ Step 3: Install PX4's Dependencies

On macOS:

```
brew install cmake ninja python@3.11 python3 -m pip install --upgrade pip
python3 -m pip install -r Tools/setup/requirements.txt
```

Then verify `cmake` and `ninja` are available:

```
cmake --version ninja --version
```

---

## ✅ Step 4: Build the Correct Simulator Target

Once you're in `~/PX4-Autopilot`, try:

### For Gazebo Sim (Harmonic / Jetty):

```
make px4_sitl gz_sim
```

### Or for JMAVSim (lightweight fallback):

```
make px4_sitl jmavsim
```

If you've installed Gazebo Sim (`brew install gz-harmonic`), `gz_sim` should now be recognized.

---

## 🧩 Step 5: Check Available Build Targets

If you're still unsure what PX4 recognizes, run:

```
make help | grep sitl
```

You should see something like:

```
px4_sitl (for jmavsim)
px4_sitl_gz (for Gazebo Sim)
```

If `px4_sitl` doesn't appear at all, it means PX4 wasn't cloned or built properly.

---

## ✅ Summary

| Problem | Cause | Fix |
|---|---|---|
| `No rule to make target 'px4_sitl'` | Not in PX4 source directory | `cd ~/PX4-Autopilot` |

| | | |
|---|---|---|
| Missing `Makefile` | Cloned without `--recursive` | Re-clone with submodules |
| Dependencies missing | CMake/Ninja not installed | `brew install cmake ninja` |
| Wrong simulator target | Use `jmavsim` or `gz_sim` | `make px4_sitl jmavsim` |

---

Would you like to paste the output of

```
ls ~/PX4-Autopilot
```

and

```
make help | grep sitl
```

so I can confirm exactly what's missing and give you the exact command that'll build successfully on your system?

**You said:**
This is the output of "make help | grep sitl": px4_sitl
px4_sitl_default-clang

**ChatGPT said:**
Perfect 👋 — that output tells us exactly what's going on.

Your PX4 source tree does recognize the base `px4_sitl` target, but not any simulator-specific targets (like `gz_sim` or `jmavsim`).

That means PX4 built the *core SITL environment*, but the simulator integration (Gazebo or JMAVSim) wasn't included or detected.

Let's fix that so you can actually *run* a simulation.

---

## 🧠 What This Means

The `px4_sitl` target by itself is the “bare” SITL environment — it runs the simulated PX4 flight stack without any world or vehicle simulator attached.

To actually simulate flight, PX4 needs to be linked to either:

- JMAVSim → lightweight, Java-based, or
- Gazebo Sim (formerly Ignition Gazebo).

Since your setup shows only:

```
px4_sitl
px4_sitl_default-clang
```

that means the extra simulator targets (like `px4_sitl jmavsim` or `px4_sitl gz_sim`) weren’t configured — likely because dependencies for those simulators were missing when PX4 first configured its build.

---

## ✅ Step 1: Verify Your Current Directory

Make sure you’re inside the PX4 root folder:

```
cd ~/PX4-Autopilot
```

---

## ✅ Step 2: Install JMAVSim (Quickest Option)

JMAVSim is the easiest simulator to get running on macOS — no Gazebo dependencies needed.

Run:

```
brew install openjdk
```

Then set Java in your environment:

```
export JAVA_HOME=$(/usr/libexec/java_home)
```

Now try:

```
make px4_sitl jmavsim
```

If JMAVSim builds and launches, you'll see a small 3D window and the PX4 console.

Then open QGroundControl — it should connect automatically over UDP (port 14550).

---

## ✅ Step 3: (Optional) Enable Gazebo Sim Support

If you want the newer Gazebo Sim environment, make sure it's installed and visible to PX4:

```
brew tap osrf/simulation brew install gz-harmonic
```

Then reconfigure PX4 so it detects Gazebo Sim:

```
cd ~/PX4-Autopilot make clean make px4_sitl gz_sim
```

Now PX4 should automatically build the `gz_sim` integration and launch Gazebo Sim.

---

## 🧩 Step 4: If You Want to See Available PX4 Targets Again

Run:

```
make help | grep px4_sitl
```

After Gazebo or JMAVSim dependencies are properly detected, you should see more options like:

```
px4_sitl jmavsim
px4_sitl gz_sim
```

---

## ✅ TL;DR (Your Next Step)

If you just want something working right now:

```
brew install openjdk export JAVA_HOME=$(/usr/libexec/java_home) cd ~/PX4-Autopilot make px4_sitl jmavsim
```

That should start the PX4 SITL + JMAVSim environment cleanly on macOS.

Then open QGroundControl, and you'll have a virtual drone flying! 🚁

---

Would you like me to show you how to make QGroundControl automatically connect and visualize the simulated drone's telemetry and GPS position?

**You said:**

Even after the commands to enable gazebo sim support, the "make help | grep px4_sitl" command shows the same output

**ChatGPT said:**

Perfect — you've done everything right so far, and your observation is very important:

If `make help | grep px4_sitl` still only lists

```
px4_sitl
px4_sitl_default-clang
```

even after installing `gz-harmonic`, it means PX4's build system is not detecting Gazebo Sim (gz-sim) at all.

This happens often on macOS (especially on Apple Silicon) because CMake can't find Gazebo's CMake configuration files unless you explicitly tell PX4 where to look.

Let's walk through the fix step-by-step 👇

---

## 🧩 Step 1: Confirm Gazebo Sim Is Installed

Run these in Terminal:

```
gz --version
```

✅ You should see something like:

```
gz 8.1.0 (Harmonic)
```

and check the CMake package files exist:

```
ls /opt/homebrew/lib/cmake/gz*
```

You should see directories like:

```
/opt/homebrew/lib/cmake/gz-sim8
/opt/homebrew/lib/cmake/gz-math8
/opt/homebrew/lib/cmake/gz-msgs9
...
```

If those are there, Gazebo Sim is installed — we just need PX4 to *find* them.

---

## 🧩 Step 2: Tell PX4 Where to Find Gazebo Sim

Set environment variables so PX4's CMake system can locate the Gazebo CMake packages:

```
export CMAKE_PREFIX_PATH=/opt/homebrew/lib/cmake export GZ_VERSION=harmonic export GZ_SIM_RESOURCE_PATH=/opt/homebrew/share/gz
```

You can verify:

```
echo $CMAKE_PREFIX_PATH
```

Then rebuild PX4 completely:

```
cd ~/PX4-Autopilot make clean make px4_sitl gz_sim
```

If it works, PX4 will start building a Gazebo Sim target (you'll see it downloading SDF models and linking `gz-transport`, `gz-math`, etc.).

---

## 🧩 Step 3: Check the Build Output for Gazebo Detection

During the build, look for lines like:

```
-- Found gz-sim8
-- Found gz-msgs9
-- Found gz-transport14
```

If you see those — success! 🎉

If you don't, it means PX4's CMake still isn't finding the Gazebo Sim packages.

---

## 🧩 Step 4: Manual CMake Build (If `make` Still Fails)

You can bypass `make`'s auto-discovery and configure PX4 manually:

```
cd ~/PX4-Autopilot mkdir build/px4_sitl_gz cd build/px4_sitl_gz cmake .. -DPX4_SIM_MODEL=gz_sim \ -DCMAKE_PREFIX_PATH=/opt/homebrew/lib/cmake \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -GNinja ninja
```

If that succeeds, you can launch PX4 SITL manually:

```
cd ~/PX4-Autopilot make px4_sitl gz_sim
```

---

## 🧠 Why This Happens on macOS

PX4's CMake scripts assume Linux-style paths (`/usr/lib/x86_64-linux-gnu/cmake`), so they don't automatically find the Homebrew install location (`/opt/homebrew/lib/cmake`) on Apple Silicon.

By setting `CMAKE_PREFIX_PATH`, you're manually pointing PX4's CMake system to the Homebrew-installed Gazebo Sim libraries.

---

## ✅ TL;DR – The Fix

1. Confirm Gazebo Sim exists:
2. `gz --version ls /opt/homebrew/lib/cmake/gz*`
3. Set environment variables:
4. `export CMAKE_PREFIX_PATH=/opt/homebrew/lib/cmake export GZ_VERSION=harmonic`
5. Rebuild:
6. `cd ~/PX4-Autopilot make clean make px4_sitl gz_sim`

Now PX4 should detect Gazebo Sim and compile the integration.

---

Would you like to paste the output of

```
ls /opt/homebrew/lib/cmake/gz*
```

and

```
gz --version
```

so I can confirm the exact CMake path to set for your specific Gazebo Sim installation?

**You said:**
gz --version doesn't show the version, but rather the help text

**ChatGPT said:**
Perfect — that's a really useful clue 👏

If `gz --version` prints the help text instead of showing something like `gz 8.1.0 (Harmonic)`, then it means:

➡️ The `gz` command is installed, but

➡️ It's an older command-line wrapper (or an incomplete install) that doesn't have the proper version flag configured.

Let's check exactly what you have and fix it so PX4 can see it.

---

## 🧩 Step 1: Check What `gz` Binary You're Running

Run:

```
which gz
```

You should see something like:

```
/opt/homebrew/bin/gz
```

Then check that it's actually the Gazebo Sim executable:

```
file $(which gz)
```

Expected output example:

```
/opt/homebrew/bin/gz: Mach-O 64-bit executable arm64
```

If it's located elsewhere (for example `/usr/local/bin/gz` or `/usr/bin/gz`), then you may have an old leftover binary from the old "Ignition" series or another tool with the same name.

---

## 🧩 Step 2: Check Installed Gazebo Sim Packages

Let's see which Gazebo components Homebrew installed:

```
brew list | grep gz-
```

If you see a list like this:

```
gz-harmonic
gz-math8
gz-msgs9
gz-transport14
gz-rendering8
gz-sim8
```

✅ then Gazebo Sim (Harmonic) is installed properly.

If you see nothing, install it cleanly with:

```
brew tap osrf/simulation brew install gz-harmonic
```

---

## 🧩 Step 3: Verify the Gazebo Sim CLI Works

Instead of `gz --version`, try:

```
gz sim --help
```

If you see usage info for `gz sim` (not an error), that means the new Gazebo CLI is working — it just doesn't show its version by default on macOS.

To double-check, you can also run:

```
gz info
```

or

```
gz plugin --help
```

Those commands confirm that Gazebo Sim's command-line utilities are accessible.

---

## 🧩 Step 4: Check for CMake Files

Finally, confirm PX4 can find the CMake packages:

```
ls /opt/homebrew/lib/cmake/gz*
```

You should see something like:

```
/opt/homebrew/lib/cmake/gz-sim8
/opt/homebrew/lib/cmake/gz-msgs9
/opt/homebrew/lib/cmake/gz-transport14
```

If those folders exist, then Gazebo Sim is installed correctly — we just need to help PX4 locate them (as we covered earlier).

---

## ✅ TL;DR

| What You Did | What It Means | Next Step |
|---|---|---|
| `gz --version` shows help | CLI exists, but version flag unsupported | Use `gz sim --help` to verify |
| `which gz` points to `/opt/homebrew/bin/gz` | Installed via Homebrew ✅ | Proceed to PX4 build |

| | | |
|---|---|---|
| `brew list | grep gz-` shows Gazebo libs | Gazebo Sim is installed ✅ |
| `ls /opt/homebrew/lib/cmake/gz*` shows folders | PX4 can find Gazebo CMake modules ✅ | Rebuild PX4 with `make px4_sitl gz_sim` |

---

If you'd like, paste the output of:

```
which gz brew list | grep gz- ls /opt/homebrew/lib/cmake/gz*
```

and I'll tell you exactly what to set for your `CMAKE_PREFIX_PATH` and how to get PX4 to recognize Gazebo Sim successfully.

**You said:**
running make px4_sitl gz sim gets this errer: ninja: error: unknown target 'gz_sim'

**ChatGPT said:**
Excellent — that's a *very clear and telling* error message 👏

When you see

```
ninja: error: unknown target 'gz_sim'
```

it means PX4's CMake configuration did not register any "gz_sim" build target.
This happens when Gazebo Sim (the new "gz" framework) isn't detected during CMake configuration, so PX4 never creates the `gz_sim` build rule.

Let's fix this carefully.

---

## 🧠 Why It Happens

Even though you installed Gazebo Sim (`gz-harmonic`), PX4's CMake build system can't automatically find its CMake packages on macOS — especially on Apple Silicon — unless you explicitly set the search path.

So we need to:

1. Confirm that Gazebo Sim is really installed and has its CMake files.
2. Point PX4's CMake to where those files are located.
3. Re-run PX4's configuration so it detects and registers the Gazebo integration.

---

## ✅ Step 1: Confirm Gazebo Sim CMake Packages Exist

Run:

```
ls /opt/homebrew/lib/cmake/gz*
```

You should see folders such as:

```
/opt/homebrew/lib/cmake/gz-sim8
/opt/homebrew/lib/cmake/gz-msgs9
/opt/homebrew/lib/cmake/gz-transport14
/opt/homebrew/lib/cmake/gz-math8
```

If those exist → Gazebo Sim is correctly installed.

If not → reinstall cleanly:

```
brew tap osrf/simulation brew reinstall gz-harmonic
```

---

## ✅ Step 2: Clear Any Old Build Artifacts

Let's start fresh to ensure CMake reconfigures correctly.

From your PX4 root directory:

```
cd ~/PX4-Autopilot make clean rm -rf build/px4_sitl*
```

---

## ✅ Step 3: Manually Point CMake to Gazebo Sim

Now create a clean build directory and run CMake manually so it knows where to look:

```
cd ~/PX4-Autopilot mkdir build/px4_sitl_gz cd build/px4_sitl_gz cmake ..
-GNinja \ -DPX4_SIM_MODEL=gz \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo
```

You should see lines like:

```
-- Found gz-sim8
-- Found gz-msgs9
-- Found gz-transport14
```

✅ That means PX4 has successfully detected Gazebo Sim and will now create the correct build targets.

Then compile:

```
ninja
```

---

## ✅ Step 4: Launch SITL with Gazebo Sim

Once the build completes:

```
cd ~/PX4-Autopilot make px4_sitl gz_sim
```

This should now work — PX4 will start the SITL backend and automatically open the Gazebo Sim GUI.

---

## ⚠️ If You Still Get “Unknown Target gz_sim”

Double-check that the build directory contains the Gazebo files:

```
ls ~/PX4-Autopilot/build/px4_sitl_gz/src/modules/simulation/
```

You should see Gazebo-related components such as:

```
gazebo_interface/
gazebo_motor_model/
gazebo_mavlink_interface/
```

If those don't exist, PX4's Gazebo module wasn't compiled — meaning it still couldn't find the CMake packages.

In that case, you may need to specify Gazebo's root install path directly:

```
cmake .. -GNinja \ -DPX4_SIM_MODEL=gz \
-DCMAKE_PREFIX_PATH="/opt/homebrew/opt/gz-sim8;/opt/homebrew/lib/cmake"
```

---

## ✅ TL;DR — The Fix

```
cd ~/PX4-Autopilot make clean rm -rf build/px4_sitl_gz mkdir
build/px4_sitl_gz cd build/px4_sitl_gz cmake .. -GNinja \
-DPX4_SIM_MODEL=gz \ -DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo ninja cd ~/PX4-Autopilot make px4_sitl
gz_sim
```

---

Would you mind pasting the output of this command:

```
ls /opt/homebrew/lib/cmake/gz*
```

so I can tell you exactly which Gazebo Sim version (e.g. `gz-sim8`, `gz-sim9`, etc.) your system is using and tailor the exact CMake configuration line for you?

**You said:**

The output of the command is this: /opt/homebrew/lib/cmake/gz-common5:
gz-common5-config-version.cmake gz-common5-targets-release.cmake
gz-common5-config.cmake gz-common5-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-all:
gz-common5-all-config-version.cmake gz-common5-all-targets.cmake
gz-common5-all-config.cmake

/opt/homebrew/lib/cmake/gz-common5-av:
gz-common5-av-config-version.cmake gz-common5-av-targets-release.cmake
gz-common5-av-config.cmake gz-common5-av-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-events:

gz-common5-events-config-version.cmake
gz-common5-events-targets-release.cmake
gz-common5-events-config.cmake gz-common5-events-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-geospatial:
gz-common5-geospatial-config-version.cmake
gz-common5-geospatial-config.cmake
gz-common5-geospatial-targets-release.cmake
gz-common5-geospatial-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-graphics:
gz-common5-graphics-config-version.cmake
gz-common5-graphics-config.cmake
gz-common5-graphics-targets-release.cmake
gz-common5-graphics-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-io:
gz-common5-io-config-version.cmake gz-common5-io-targets-release.cmake
gz-common5-io-config.cmake gz-common5-io-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-profiler:
gz-common5-profiler-config-version.cmake
gz-common5-profiler-config.cmake
gz-common5-profiler-targets-release.cmake
gz-common5-profiler-targets.cmake

/opt/homebrew/lib/cmake/gz-common5-testing:
gz-common5-testing-config-version.cmake
gz-common5-testing-config.cmake
gz-common5-testing-targets-release.cmake
gz-common5-testing-targets.cmake

/opt/homebrew/lib/cmake/gz-fuel_tools9:
gz-fuel_tools9-config-version.cmake gz-fuel_tools9-targets-release.cmake
gz-fuel_tools9-config.cmake gz-fuel_tools9-targets.cmake

/opt/homebrew/lib/cmake/gz-fuel_tools9-all:
gz-fuel_tools9-all-config-version.cmake gz-fuel_tools9-all-targets.cmake
gz-fuel_tools9-all-config.cmake

/opt/homebrew/lib/cmake/gz-gui8:
gz-gui8-config-version.cmake gz-gui8-targets-release.cmake
gz-gui8-config.cmake gz-gui8-targets.cmake

/opt/homebrew/lib/cmake/gz-gui8-all:
gz-gui8-all-config-version.cmake gz-gui8-all-targets.cmake
gz-gui8-all-config.cmake

/opt/homebrew/lib/cmake/gz-launch7:
gz-launch7-config-version.cmake gz-launch7-targets-release.cmake
gz-launch7-config.cmake gz-launch7-targets.cmake

/opt/homebrew/lib/cmake/gz-launch7-all:
gz-launch7-all-config-version.cmake gz-launch7-all-targets.cmake
gz-launch7-all-config.cmake

/opt/homebrew/lib/cmake/gz-math7:
gz-math7-config-version.cmake gz-math7-targets-release.cmake
gz-math7-config.cmake gz-math7-targets.cmake

/opt/homebrew/lib/cmake/gz-math7-all:
gz-math7-all-config-version.cmake gz-math7-all-targets.cmake
gz-math7-all-config.cmake

/opt/homebrew/lib/cmake/gz-math7-eigen3:
gz-math7-eigen3-config-version.cmake gz-math7-eigen3-targets.cmake
gz-math7-eigen3-config.cmake

/opt/homebrew/lib/cmake/gz-msgs10:
gz_msgs_factory.cmake gz-msgs10-config-version.cmake
gz_msgs_generate.cmake gz-msgs10-config.cmake
gz_msgs_protoc.cmake gz-msgs10-targets-release.cmake
gz_msgs_string_utils.cmake gz-msgs10-targets.cmake
gz-msgs-extras.cmake

/opt/homebrew/lib/cmake/gz-msgs10-all:
gz-msgs10-all-config-version.cmake gz-msgs10-all-targets.cmake
gz-msgs10-all-config.cmake

/opt/homebrew/lib/cmake/gz-physics7:

gz-physics7-config-version.cmake gz-physics7-targets-release.cmake
gz-physics7-config.cmake gz-physics7-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-all:
gz-physics7-all-config-version.cmake gz-physics7-all-targets.cmake
gz-physics7-all-config.cmake

/opt/homebrew/lib/cmake/gz-physics7-bullet:
gz-physics7-bullet-config-version.cmake gz-physics7-bullet-targets.cmake
gz-physics7-bullet-config.cmake

/opt/homebrew/lib/cmake/gz-physics7-bullet-featherstone:
gz-physics7-bullet-featherstone-config-version.cmake
gz-physics7-bullet-featherstone-config.cmake
gz-physics7-bullet-featherstone-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-bullet-featherstone-plugin:
gz-physics7-bullet-featherstone-plugin-config-version.cmake
gz-physics7-bullet-featherstone-plugin-config.cmake
gz-physics7-bullet-featherstone-plugin-targets-release.cmake
gz-physics7-bullet-featherstone-plugin-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-bullet-plugin:
gz-physics7-bullet-plugin-config-version.cmake
gz-physics7-bullet-plugin-config.cmake
gz-physics7-bullet-plugin-targets-release.cmake
gz-physics7-bullet-plugin-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-dartsim:
gz-physics7-dartsim-config-version.cmake
gz-physics7-dartsim-config.cmake
gz-physics7-dartsim-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-dartsim-plugin:
gz-physics7-dartsim-plugin-config-version.cmake
gz-physics7-dartsim-plugin-config.cmake
gz-physics7-dartsim-plugin-targets-release.cmake
gz-physics7-dartsim-plugin-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-heightmap:

gz-physics7-heightmap-config-version.cmake
gz-physics7-heightmap-config.cmake
gz-physics7-heightmap-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-mesh:
gz-physics7-mesh-config-version.cmake gz-physics7-mesh-targets.cmake
gz-physics7-mesh-config.cmake

/opt/homebrew/lib/cmake/gz-physics7-sdf:
gz-physics7-sdf-config-version.cmake gz-physics7-sdf-targets.cmake
gz-physics7-sdf-config.cmake

/opt/homebrew/lib/cmake/gz-physics7-tpe:
gz-physics7-tpe-config-version.cmake gz-physics7-tpe-targets.cmake
gz-physics7-tpe-config.cmake

/opt/homebrew/lib/cmake/gz-physics7-tpe-plugin:
gz-physics7-tpe-plugin-config-version.cmake
gz-physics7-tpe-plugin-config.cmake
gz-physics7-tpe-plugin-targets-release.cmake
gz-physics7-tpe-plugin-targets.cmake

/opt/homebrew/lib/cmake/gz-physics7-tpelib:
gz-physics7-tpelib-config-version.cmake
gz-physics7-tpelib-config.cmake
gz-physics7-tpelib-targets-release.cmake
gz-physics7-tpelib-targets.cmake

/opt/homebrew/lib/cmake/gz-plugin2:
gz-plugin2-config-version.cmake gz-plugin2-targets-release.cmake
gz-plugin2-config.cmake gz-plugin2-targets.cmake

/opt/homebrew/lib/cmake/gz-plugin2-all:
gz-plugin2-all-config-version.cmake gz-plugin2-all-targets.cmake
gz-plugin2-all-config.cmake

/opt/homebrew/lib/cmake/gz-plugin2-loader:
gz-plugin2-loader-config-version.cmake gz-plugin2-loader-targets-release.cmake
gz-plugin2-loader-config.cmake gz-plugin2-loader-targets.cmake

/opt/homebrew/lib/cmake/gz-plugin2-register:
gz-plugin2-register-config-version.cmake
gz-plugin2-register-config.cmake
gz-plugin2-register-targets.cmake

/opt/homebrew/lib/cmake/gz-rendering8:
gz-rendering8-config-version.cmake gz-rendering8-targets-release.cmake
gz-rendering8-config.cmake gz-rendering8-targets.cmake

/opt/homebrew/lib/cmake/gz-rendering8-all:
gz-rendering8-all-config-version.cmake gz-rendering8-all-targets.cmake
gz-rendering8-all-config.cmake

/opt/homebrew/lib/cmake/gz-rendering8-ogre:
gz-rendering8-ogre-config-version.cmake
gz-rendering8-ogre-config.cmake
gz-rendering8-ogre-targets-release.cmake
gz-rendering8-ogre-targets.cmake

/opt/homebrew/lib/cmake/gz-rendering8-ogre2:
gz-rendering8-ogre2-config-version.cmake
gz-rendering8-ogre2-config.cmake
gz-rendering8-ogre2-targets-release.cmake
gz-rendering8-ogre2-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8:
gz-sensors8-config-version.cmake gz-sensors8-targets-release.cmake
gz-sensors8-config.cmake gz-sensors8-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-air_pressure:
gz-sensors8-air_pressure-config-version.cmake
gz-sensors8-air_pressure-config.cmake
gz-sensors8-air_pressure-targets-release.cmake
gz-sensors8-air_pressure-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-air_speed:
gz-sensors8-air_speed-config-version.cmake
gz-sensors8-air_speed-config.cmake
gz-sensors8-air_speed-targets-release.cmake
gz-sensors8-air_speed-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-all:
gz-sensors8-all-config-version.cmake  gz-sensors8-all-targets.cmake
gz-sensors8-all-config.cmake

/opt/homebrew/lib/cmake/gz-sensors8-altimeter:
gz-sensors8-altimeter-config-version.cmake
gz-sensors8-altimeter-config.cmake
gz-sensors8-altimeter-targets-release.cmake
gz-sensors8-altimeter-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-boundingbox_camera:
gz-sensors8-boundingbox_camera-config-version.cmake
gz-sensors8-boundingbox_camera-config.cmake
gz-sensors8-boundingbox_camera-targets-release.cmake
gz-sensors8-boundingbox_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-camera:
gz-sensors8-camera-config-version.cmake
gz-sensors8-camera-config.cmake
gz-sensors8-camera-targets-release.cmake
gz-sensors8-camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-depth_camera:
gz-sensors8-depth_camera-config-version.cmake
gz-sensors8-depth_camera-config.cmake
gz-sensors8-depth_camera-targets-release.cmake
gz-sensors8-depth_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-dvl:
gz-sensors8-dvl-config-version.cmake  gz-sensors8-dvl-targets-release.cmake
gz-sensors8-dvl-config.cmake  gz-sensors8-dvl-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-force_torque:
gz-sensors8-force_torque-config-version.cmake
gz-sensors8-force_torque-config.cmake
gz-sensors8-force_torque-targets-release.cmake
gz-sensors8-force_torque-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-gpu_lidar:

gz-sensors8-gpu_lidar-config-version.cmake
gz-sensors8-gpu_lidar-config.cmake
gz-sensors8-gpu_lidar-targets-release.cmake
gz-sensors8-gpu_lidar-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-imu:
gz-sensors8-imu-config-version.cmake  gz-sensors8-imu-targets-release.cmake
gz-sensors8-imu-config.cmake  gz-sensors8-imu-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-lidar:
gz-sensors8-lidar-config-version.cmake  gz-sensors8-lidar-targets-release.cmake
gz-sensors8-lidar-config.cmake  gz-sensors8-lidar-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-logical_camera:
gz-sensors8-logical_camera-config-version.cmake
gz-sensors8-logical_camera-config.cmake
gz-sensors8-logical_camera-targets-release.cmake
gz-sensors8-logical_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-magnetometer:
gz-sensors8-magnetometer-config-version.cmake
gz-sensors8-magnetometer-config.cmake
gz-sensors8-magnetometer-targets-release.cmake
gz-sensors8-magnetometer-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-navsat:
gz-sensors8-navsat-config-version.cmake
gz-sensors8-navsat-config.cmake
gz-sensors8-navsat-targets-release.cmake
gz-sensors8-navsat-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-rendering:
gz-sensors8-rendering-config-version.cmake
gz-sensors8-rendering-config.cmake
gz-sensors8-rendering-targets-release.cmake
gz-sensors8-rendering-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-rgbd_camera:
gz-sensors8-rgbd_camera-config-version.cmake
gz-sensors8-rgbd_camera-config.cmake

gz-sensors8-rgbd_camera-targets-release.cmake
gz-sensors8-rgbd_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-segmentation_camera:
gz-sensors8-segmentation_camera-config-version.cmake
gz-sensors8-segmentation_camera-config.cmake
gz-sensors8-segmentation_camera-targets-release.cmake
gz-sensors8-segmentation_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-thermal_camera:
gz-sensors8-thermal_camera-config-version.cmake
gz-sensors8-thermal_camera-config.cmake
gz-sensors8-thermal_camera-targets-release.cmake
gz-sensors8-thermal_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sensors8-wide_angle_camera:
gz-sensors8-wide_angle_camera-config-version.cmake
gz-sensors8-wide_angle_camera-config.cmake
gz-sensors8-wide_angle_camera-targets-release.cmake
gz-sensors8-wide_angle_camera-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8:
gz-sim8-config-version.cmake gz-sim8-targets-release.cmake
gz-sim8-config.cmake gz-sim8-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-ackermann-steering-system:
gz-sim8-ackermann-steering-system-config-version.cmake
gz-sim8-ackermann-steering-system-config.cmake
gz-sim8-ackermann-steering-system-targets-release.cmake
gz-sim8-ackermann-steering-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-acoustic-comms-system:
gz-sim8-acoustic-comms-system-config-version.cmake
gz-sim8-acoustic-comms-system-config.cmake
gz-sim8-acoustic-comms-system-targets-release.cmake
gz-sim8-acoustic-comms-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-advanced-lift-drag-system:
gz-sim8-advanced-lift-drag-system-config-version.cmake
gz-sim8-advanced-lift-drag-system-config.cmake

gz-sim8-advanced-lift-drag-system-targets-release.cmake
gz-sim8-advanced-lift-drag-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-air-pressure-system:
gz-sim8-air-pressure-system-config-version.cmake
gz-sim8-air-pressure-system-config.cmake
gz-sim8-air-pressure-system-targets-release.cmake
gz-sim8-air-pressure-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-air-speed-system:
gz-sim8-air-speed-system-config-version.cmake
gz-sim8-air-speed-system-config.cmake
gz-sim8-air-speed-system-targets-release.cmake
gz-sim8-air-speed-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-all:
gz-sim8-all-config-version.cmake  gz-sim8-all-targets.cmake
gz-sim8-all-config.cmake

/opt/homebrew/lib/cmake/gz-sim8-altimeter-system:
gz-sim8-altimeter-system-config-version.cmake
gz-sim8-altimeter-system-config.cmake
gz-sim8-altimeter-system-targets-release.cmake
gz-sim8-altimeter-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-apply-joint-force-system:
gz-sim8-apply-joint-force-system-config-version.cmake
gz-sim8-apply-joint-force-system-config.cmake
gz-sim8-apply-joint-force-system-targets-release.cmake
gz-sim8-apply-joint-force-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-apply-link-wrench-system:
gz-sim8-apply-link-wrench-system-config-version.cmake
gz-sim8-apply-link-wrench-system-config.cmake
gz-sim8-apply-link-wrench-system-targets-release.cmake
gz-sim8-apply-link-wrench-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-breadcrumbs-system:
gz-sim8-breadcrumbs-system-config-version.cmake
gz-sim8-breadcrumbs-system-config.cmake

gz-sim8-breadcrumbs-system-targets-release.cmake
gz-sim8-breadcrumbs-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-buoyancy-engine-system:
gz-sim8-buoyancy-engine-system-config-version.cmake
gz-sim8-buoyancy-engine-system-config.cmake
gz-sim8-buoyancy-engine-system-targets-release.cmake
gz-sim8-buoyancy-engine-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-buoyancy-system:
gz-sim8-buoyancy-system-config-version.cmake
gz-sim8-buoyancy-system-config.cmake
gz-sim8-buoyancy-system-targets-release.cmake
gz-sim8-buoyancy-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-camera-video-recorder-system:
gz-sim8-camera-video-recorder-system-config-version.cmake
gz-sim8-camera-video-recorder-system-config.cmake
gz-sim8-camera-video-recorder-system-targets-release.cmake
gz-sim8-camera-video-recorder-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-collada-world-exporter-system:
gz-sim8-collada-world-exporter-system-config-version.cmake
gz-sim8-collada-world-exporter-system-config.cmake
gz-sim8-collada-world-exporter-system-targets-release.cmake
gz-sim8-collada-world-exporter-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-comms-endpoint-system:
gz-sim8-comms-endpoint-system-config-version.cmake
gz-sim8-comms-endpoint-system-config.cmake
gz-sim8-comms-endpoint-system-targets-release.cmake
gz-sim8-comms-endpoint-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-contact-system:
gz-sim8-contact-system-config-version.cmake
gz-sim8-contact-system-config.cmake
gz-sim8-contact-system-targets-release.cmake
gz-sim8-contact-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-detachable-joint-system:

gz-sim8-detachable-joint-system-config-version.cmake
gz-sim8-detachable-joint-system-config.cmake
gz-sim8-detachable-joint-system-targets-release.cmake
gz-sim8-detachable-joint-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-diff-drive-system:
gz-sim8-diff-drive-system-config-version.cmake
gz-sim8-diff-drive-system-config.cmake
gz-sim8-diff-drive-system-targets-release.cmake
gz-sim8-diff-drive-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-dvl-system:
gz-sim8-dvl-system-config-version.cmake
gz-sim8-dvl-system-config.cmake
gz-sim8-dvl-system-targets-release.cmake
gz-sim8-dvl-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-elevator-system:
gz-sim8-elevator-system-config-version.cmake
gz-sim8-elevator-system-config.cmake
gz-sim8-elevator-system-targets-release.cmake
gz-sim8-elevator-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-environment-preload-system:
gz-sim8-environment-preload-system-config-version.cmake
gz-sim8-environment-preload-system-config.cmake
gz-sim8-environment-preload-system-targets-release.cmake
gz-sim8-environment-preload-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-environmental-sensor-system:
gz-sim8-environmental-sensor-system-config-version.cmake
gz-sim8-environmental-sensor-system-config.cmake
gz-sim8-environmental-sensor-system-targets-release.cmake
gz-sim8-environmental-sensor-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-follow-actor-system:
gz-sim8-follow-actor-system-config-version.cmake
gz-sim8-follow-actor-system-config.cmake
gz-sim8-follow-actor-system-targets-release.cmake
gz-sim8-follow-actor-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-forcetorque-system:
gz-sim8-forcetorque-system-config-version.cmake
gz-sim8-forcetorque-system-config.cmake
gz-sim8-forcetorque-system-targets-release.cmake
gz-sim8-forcetorque-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-gui:
gz-sim8-gui-config-version.cmake gz-sim8-gui-targets-release.cmake
gz-sim8-gui-config.cmake gz-sim8-gui-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-gz:
gz-sim8-gz-config-version.cmake gz-sim8-gz-targets-release.cmake
gz-sim8-gz-config.cmake gz-sim8-gz-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-hydrodynamics-system:
gz-sim8-hydrodynamics-system-config-version.cmake
gz-sim8-hydrodynamics-system-config.cmake
gz-sim8-hydrodynamics-system-targets-release.cmake
gz-sim8-hydrodynamics-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-imu-system:
gz-sim8-imu-system-config-version.cmake
gz-sim8-imu-system-config.cmake
gz-sim8-imu-system-targets-release.cmake
gz-sim8-imu-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-joint-controller-system:
gz-sim8-joint-controller-system-config-version.cmake
gz-sim8-joint-controller-system-config.cmake
gz-sim8-joint-controller-system-targets-release.cmake
gz-sim8-joint-controller-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-joint-position-controller-system:
gz-sim8-joint-position-controller-system-config-version.cmake
gz-sim8-joint-position-controller-system-config.cmake
gz-sim8-joint-position-controller-system-targets-release.cmake
gz-sim8-joint-position-controller-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-joint-state-publisher-system:

gz-sim8-joint-state-publisher-system-config-version.cmake
gz-sim8-joint-state-publisher-system-config.cmake
gz-sim8-joint-state-publisher-system-targets-release.cmake
gz-sim8-joint-state-publisher-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-joint-trajectory-controller-system:
gz-sim8-joint-trajectory-controller-system-config-version.cmake
gz-sim8-joint-trajectory-controller-system-config.cmake
gz-sim8-joint-trajectory-controller-system-targets-release.cmake
gz-sim8-joint-trajectory-controller-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-kinetic-energy-monitor-system:
gz-sim8-kinetic-energy-monitor-system-config-version.cmake
gz-sim8-kinetic-energy-monitor-system-config.cmake
gz-sim8-kinetic-energy-monitor-system-targets-release.cmake
gz-sim8-kinetic-energy-monitor-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-label-system:
gz-sim8-label-system-config-version.cmake
gz-sim8-label-system-config.cmake
gz-sim8-label-system-targets-release.cmake
gz-sim8-label-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-lens-flare-system:
gz-sim8-lens-flare-system-config-version.cmake
gz-sim8-lens-flare-system-config.cmake
gz-sim8-lens-flare-system-targets-release.cmake
gz-sim8-lens-flare-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-lift-drag-system:
gz-sim8-lift-drag-system-config-version.cmake
gz-sim8-lift-drag-system-config.cmake
gz-sim8-lift-drag-system-targets-release.cmake
gz-sim8-lift-drag-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-lighter_than_air_dynamics-system:
gz-sim8-lighter_than_air_dynamics-system-config-version.cmake
gz-sim8-lighter_than_air_dynamics-system-config.cmake
gz-sim8-lighter_than_air_dynamics-system-targets-release.cmake
gz-sim8-lighter_than_air_dynamics-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-linearbatteryplugin-system:
gz-sim8-linearbatteryplugin-system-config-version.cmake
gz-sim8-linearbatteryplugin-system-config.cmake
gz-sim8-linearbatteryplugin-system-targets-release.cmake
gz-sim8-linearbatteryplugin-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-log-system:
gz-sim8-log-system-config-version.cmake
gz-sim8-log-system-config.cmake
gz-sim8-log-system-targets-release.cmake
gz-sim8-log-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-log-video-recorder-system:
gz-sim8-log-video-recorder-system-config-version.cmake
gz-sim8-log-video-recorder-system-config.cmake
gz-sim8-log-video-recorder-system-targets-release.cmake
gz-sim8-log-video-recorder-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-logical-camera-system:
gz-sim8-logical-camera-system-config-version.cmake
gz-sim8-logical-camera-system-config.cmake
gz-sim8-logical-camera-system-targets-release.cmake
gz-sim8-logical-camera-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-logicalaudiosensorplugin-system:
gz-sim8-logicalaudiosensorplugin-system-config-version.cmake
gz-sim8-logicalaudiosensorplugin-system-config.cmake
gz-sim8-logicalaudiosensorplugin-system-targets-release.cmake
gz-sim8-logicalaudiosensorplugin-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-magnetometer-system:
gz-sim8-magnetometer-system-config-version.cmake
gz-sim8-magnetometer-system-config.cmake
gz-sim8-magnetometer-system-targets-release.cmake
gz-sim8-magnetometer-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-mecanum-drive-system:
gz-sim8-mecanum-drive-system-config-version.cmake
gz-sim8-mecanum-drive-system-config.cmake

gz-sim8-mecanum-drive-system-targets-release.cmake
gz-sim8-mecanum-drive-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-model-photo-shoot-system:
gz-sim8-model-photo-shoot-system-config-version.cmake
gz-sim8-model-photo-shoot-system-config.cmake
gz-sim8-model-photo-shoot-system-targets-release.cmake
gz-sim8-model-photo-shoot-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-multicopter-control-system:
gz-sim8-multicopter-control-system-config-version.cmake
gz-sim8-multicopter-control-system-config.cmake
gz-sim8-multicopter-control-system-targets-release.cmake
gz-sim8-multicopter-control-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-multicopter-motor-model-system:
gz-sim8-multicopter-motor-model-system-config-version.cmake
gz-sim8-multicopter-motor-model-system-config.cmake
gz-sim8-multicopter-motor-model-system-targets-release.cmake
gz-sim8-multicopter-motor-model-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-navsat-system:
gz-sim8-navsat-system-config-version.cmake
gz-sim8-navsat-system-config.cmake
gz-sim8-navsat-system-targets-release.cmake
gz-sim8-navsat-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-odometry-publisher-system:
gz-sim8-odometry-publisher-system-config-version.cmake
gz-sim8-odometry-publisher-system-config.cmake
gz-sim8-odometry-publisher-system-targets-release.cmake
gz-sim8-odometry-publisher-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-opticaltactileplugin-system:
gz-sim8-opticaltactileplugin-system-config-version.cmake
gz-sim8-opticaltactileplugin-system-config.cmake
gz-sim8-opticaltactileplugin-system-targets-release.cmake
gz-sim8-opticaltactileplugin-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-particle-emitter-system:

gz-sim8-particle-emitter-system-config-version.cmake
gz-sim8-particle-emitter-system-config.cmake
gz-sim8-particle-emitter-system-targets-release.cmake
gz-sim8-particle-emitter-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-perfect-comms-system:
gz-sim8-perfect-comms-system-config-version.cmake
gz-sim8-perfect-comms-system-config.cmake
gz-sim8-perfect-comms-system-targets-release.cmake
gz-sim8-perfect-comms-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-performer-detector-system:
gz-sim8-performer-detector-system-config-version.cmake
gz-sim8-performer-detector-system-config.cmake
gz-sim8-performer-detector-system-targets-release.cmake
gz-sim8-performer-detector-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-physics-system:
gz-sim8-physics-system-config-version.cmake
gz-sim8-physics-system-config.cmake
gz-sim8-physics-system-targets-release.cmake
gz-sim8-physics-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-pose-publisher-system:
gz-sim8-pose-publisher-system-config-version.cmake
gz-sim8-pose-publisher-system-config.cmake
gz-sim8-pose-publisher-system-targets-release.cmake
gz-sim8-pose-publisher-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-python-system-loader-system:
gz-sim8-python-system-loader-system-config-version.cmake
gz-sim8-python-system-loader-system-config.cmake
gz-sim8-python-system-loader-system-targets-release.cmake
gz-sim8-python-system-loader-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-rendering:
gz-sim8-rendering-config-version.cmake gz-sim8-rendering-targets-release.cmake
gz-sim8-rendering-config.cmake gz-sim8-rendering-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-rf-comms-system:

gz-sim8-rf-comms-system-config-version.cmake
gz-sim8-rf-comms-system-config.cmake
gz-sim8-rf-comms-system-targets-release.cmake
gz-sim8-rf-comms-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-scene-broadcaster-system:
gz-sim8-scene-broadcaster-system-config-version.cmake
gz-sim8-scene-broadcaster-system-config.cmake
gz-sim8-scene-broadcaster-system-targets-release.cmake
gz-sim8-scene-broadcaster-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-sensors-system:
gz-sim8-sensors-system-config-version.cmake
gz-sim8-sensors-system-config.cmake
gz-sim8-sensors-system-targets-release.cmake
gz-sim8-sensors-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-shader-param-system:
gz-sim8-shader-param-system-config-version.cmake
gz-sim8-shader-param-system-config.cmake
gz-sim8-shader-param-system-targets-release.cmake
gz-sim8-shader-param-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-spacecraft-thruster-model-system:
gz-sim8-spacecraft-thruster-model-system-config-version.cmake
gz-sim8-spacecraft-thruster-model-system-config.cmake
gz-sim8-spacecraft-thruster-model-system-targets-release.cmake
gz-sim8-spacecraft-thruster-model-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-thermal-sensor-system:
gz-sim8-thermal-sensor-system-config-version.cmake
gz-sim8-thermal-sensor-system-config.cmake
gz-sim8-thermal-sensor-system-targets-release.cmake
gz-sim8-thermal-sensor-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-thermal-system:
gz-sim8-thermal-system-config-version.cmake
gz-sim8-thermal-system-config.cmake
gz-sim8-thermal-system-targets-release.cmake
gz-sim8-thermal-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-thruster-system:
gz-sim8-thruster-system-config-version.cmake
gz-sim8-thruster-system-config.cmake
gz-sim8-thruster-system-targets-release.cmake
gz-sim8-thruster-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-touchplugin-system:
gz-sim8-touchplugin-system-config-version.cmake
gz-sim8-touchplugin-system-config.cmake
gz-sim8-touchplugin-system-targets-release.cmake
gz-sim8-touchplugin-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-track-controller-system:
gz-sim8-track-controller-system-config-version.cmake
gz-sim8-track-controller-system-config.cmake
gz-sim8-track-controller-system-targets-release.cmake
gz-sim8-track-controller-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-tracked-vehicle-system:
gz-sim8-tracked-vehicle-system-config-version.cmake
gz-sim8-tracked-vehicle-system-config.cmake
gz-sim8-tracked-vehicle-system-targets-release.cmake
gz-sim8-tracked-vehicle-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-trajectory-follower-system:
gz-sim8-trajectory-follower-system-config-version.cmake
gz-sim8-trajectory-follower-system-config.cmake
gz-sim8-trajectory-follower-system-targets-release.cmake
gz-sim8-trajectory-follower-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-triggered-publisher-system:
gz-sim8-triggered-publisher-system-config-version.cmake
gz-sim8-triggered-publisher-system-config.cmake
gz-sim8-triggered-publisher-system-targets-release.cmake
gz-sim8-triggered-publisher-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-user-commands-system:
gz-sim8-user-commands-system-config-version.cmake
gz-sim8-user-commands-system-config.cmake

gz-sim8-user-commands-system-targets-release.cmake
gz-sim8-user-commands-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-velocity-control-system:
gz-sim8-velocity-control-system-config-version.cmake
gz-sim8-velocity-control-system-config.cmake
gz-sim8-velocity-control-system-targets-release.cmake
gz-sim8-velocity-control-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-wheel-slip-system:
gz-sim8-wheel-slip-system-config-version.cmake
gz-sim8-wheel-slip-system-config.cmake
gz-sim8-wheel-slip-system-targets-release.cmake
gz-sim8-wheel-slip-system-targets.cmake

/opt/homebrew/lib/cmake/gz-sim8-wind-effects-system:
gz-sim8-wind-effects-system-config-version.cmake
gz-sim8-wind-effects-system-config.cmake
gz-sim8-wind-effects-system-targets-release.cmake
gz-sim8-wind-effects-system-targets.cmake

/opt/homebrew/lib/cmake/gz-tools2-all:
gz-tools2-all-config-version.cmake  gz-tools2-all-targets.cmake
gz-tools2-all-config.cmake

/opt/homebrew/lib/cmake/gz-transport13:
gz-transport13-config-version.cmake  gz-transport13-targets-release.cmake
gz-transport13-config.cmake  gz-transport13-targets.cmake

/opt/homebrew/lib/cmake/gz-transport13-all:
gz-transport13-all-config-version.cmake  gz-transport13-all-targets.cmake
gz-transport13-all-config.cmake

/opt/homebrew/lib/cmake/gz-transport13-log:
gz-transport13-log-config-version.cmake
gz-transport13-log-config.cmake
gz-transport13-log-targets-release.cmake
gz-transport13-log-targets.cmake

/opt/homebrew/lib/cmake/gz-transport13-parameters:

gz-transport13-parameters-config-version.cmake
gz-transport13-parameters-config.cmake
gz-transport13-parameters-targets-release.cmake
gz-transport13-parameters-targets.cmake

/opt/homebrew/lib/cmake/gz-utils2:
gz-utils2-config-version.cmake gz-utils2-targets-release.cmake
gz-utils2-config.cmake gz-utils2-targets.cmake

/opt/homebrew/lib/cmake/gz-utils2-all:
gz-utils2-all-config-version.cmake gz-utils2-all-targets.cmake
gz-utils2-all-config.cmake

/opt/homebrew/lib/cmake/gz-utils2-cli:
gz-utils2-cli-config-version.cmake gz-utils2-cli-targets.cmake
gz-utils2-cli-config.cmake

**ChatGPT said:**

Perfect — this output confirms that Gazebo Sim (Ignition Gazebo) is installed on your Mac, specifically Gazebo Harmonic (gz-sim8), which is the latest stable version of Gazebo under the new naming convention (`gz` instead of `ignition`).

So the simulator is installed, but PX4 isn't detecting it correctly because the PX4 build system still expects older environment variables or CMake hints, especially on macOS where Homebrew installs under `/opt/homebrew`.

Let's fix that so PX4 can find Gazebo Sim properly. 👇

---

## 🧭 Step 1: Verify `gz` command is in PATH

Run this:

```
which gz
```

If it doesn't return `/opt/homebrew/bin/gz`, then add it manually:

```
echo 'export PATH="/opt/homebrew/bin:$PATH"' >> ~/.zshrc source ~/.zshrc
```

Then confirm:

```
gz help
```

If you see the Gazebo CLI help message (like “Gazebo command line interface”), then your PATH is fine.

---

## 🧩 Step 2: Tell PX4 where Gazebo Sim is installed

Set the `CMAKE_PREFIX_PATH` environment variable so PX4’s CMake build system can find all Gazebo Sim dependencies:

```
export CMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake:$CMAKE_PREFIX_PATH"
```

You can also make it permanent by adding that line to your `~/.zshrc`.

---

## 🧰 Step 3: Clean and rebuild PX4 with Gazebo Sim enabled

From your PX4 source directory:

```
make clean DONT_RUN=1 make px4_sitl_default
```

Then try to build with Gazebo Sim explicitly:

```
make px4_sitl_default gz
```

or (depending on PX4 version):

```
make px4_sitl_default gazebo
```

⚠️ Important: `gz_sim` is not a valid target — PX4 uses `gz` as the target name.

---

## 🧪 Step 4: Test the simulator manually

Try launching Gazebo Sim directly to confirm it runs:

```
gz sim shapes.sdf
```

If you see a GUI window appear, Gazebo Sim is working fine.

---

If that command runs Gazebo but PX4 still can't find `gz`, it means PX4's CMake is missing the Gazebo Sim cmake config files in `/opt/homebrew/lib/cmake/gz-sim8`.

In that case, we can explicitly tell it by adding this line before running `make`:

```
export GZ_SIM_PATH="/opt/homebrew/lib/cmake/gz-sim8"
```

---

Would you like me to check which exact PX4 version or branch you're using (so I can give you the matching `make` target name)?

It varies slightly between PX4 v1.13, v1.14, and main.

**You said:**
Currently the make px4_sitl_default returns this error: Error: /Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default is not a directory

**ChatGPT said:**
Perfect — that's a very clear sign of what's happening.

Let's fix this systematically.

---

## 💡 The error

```
Error: /Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default is not a directory
```

means that PX4 has not yet built the SITL target, so the build directory is missing.

That's normal if it's your first build attempt.

---

## ✅ Step-by-step fix

1. Navigate to your PX4 repo:

```
cd ~/px4_ws/PX4-Autopilot
```

2. Clean any previous build leftovers (optional but safe):

4. `make clean rm -rf build`
5. Try building SITL without Gazebo first to ensure your PX4 environment is fine:
6. `make px4_sitl_default`
    - This should trigger a full compile (can take several minutes).
    - At the end, you should see something like:

```
-- Build files have been written to:
/Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default
[100%] Built target px4
```

    -
7. Then check that the build directory exists:
8. `ls build/px4_sitl_default`
   You should see several subfolders like `bin/`, `etc/`, etc.
9. Now, try launching with Gazebo Sim (Ignition Gazebo):
10. `make px4_sitl_default gz`
    ⚠️ Note: The correct target name is `gz`, not `gz_sim` for PX4 1.15+.
    If that fails with something like "unknown target `gz`", it means the Gazebo simulation plugin isn't detected — which we can fix next.

---

## 🧠 Quick verification

Before you try step 5, please run this and paste the output:

```
ls Tools/simulation/gz
```

That tells us if your PX4 repo actually contains the new Gazebo simulation module (it should if you're on PX4 v1.15 or newer).

---

Once you confirm that, I'll help you ensure PX4's CMake finds your Gazebo installation (you likely need to set `GZ_VERSION=8` or `GZ_SIM_RESOURCE_PATH`).

Would you like to confirm your PX4 version first (`git describe --tags`)?

**You said:**

Running the command make px4_sitl_default returns the error Error: /Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default is not a directory
make: *** [px4_sitl_default] Error 1

**ChatGPT said:**

Excellent — that's a *very clear signal* that PX4 didn't even start building.

Let's fix this cleanly.

---

## ⚠️ What the Error Means

```
Error: /Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default is not a directory
make: *** [px4_sitl_default] Error 1
```

This means:

- The PX4 `make` wrapper tried to access the build directory, but
- CMake was never invoked, so that directory doesn't exist.

This usually happens if:

1. PX4's build system wasn't initialized properly (no toolchain/config setup).
2. You're running the command from the wrong directory.
3. Or you're missing `cmake` / `ninja` / `python3` dependencies.

Let's fix that in order. 👇

---

## ✅ Step 1: Make sure you're in the PX4 root directory

Run this:

```
pwd
```

You should see something like:

```
/Users/user/px4_ws/PX4-Autopilot
```

If not, navigate there:

```
cd ~/px4_ws/PX4-Autopilot
```

---

## ✅ Step 2: Install PX4's build dependencies (macOS)

PX4 requires specific tools that Homebrew can install easily:

```
brew install cmake ninja python3 eigen pyqt python-packaging pip3 install jinja2 toml empy pyros-genmsg pyros-msgs
```

These are essential for the SITL build system.

---

## ✅ Step 3: Clean any partial builds

Let's remove any broken build cache:

```
make clean rm -rf build
```

---

## ✅ Step 4: Manually create the build directory

Just in case, we'll create the directory ourselves:

```
mkdir -p build/px4_sitl_default
```

Then run:

```
make px4_sitl_default
```

If this works, you'll start seeing CMake configure output like:

```
-- PX4 version: v1.15.0
-- Configuring done
-- Generating done
-- Build files have been written to:
/Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

If you see that — ✅ PX4 is building correctly.

---

## ✅ Step 5: (If Step 4 fails again)

Try running CMake manually to get a readable error message:

```
cd ~/px4_ws/PX4-Autopilot mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake .. -GNinja
```

If this fails, it'll show the *exact* missing dependency or path issue (for example, a missing toolchain or Python module).

---

## 🧭 Step 6: Verify PX4 repo is complete

Just to ensure your source tree is valid, run:

```
ls Tools/simulation/
```

You should see folders like `gz`, `gazebo-classic`, `jsbsim`, `jMAVSim`.

If not, your PX4 repo might be incomplete (e.g. missing submodules).

You can fix that with:

```
git submodule update --init --recursive
```

---

## ✅ Summary

| Step | Command | What It Does |
|---|---|---|
| 1 | `cd ~/px4_ws/PX4-Autopilot` | Ensure you're in PX4 root |
| 2 | `brew install cmake ninja python3 eigen pyqt python-packaging` | Installs build tools |
| 3 | `make clean && rm -rf build` | Clean stale build |
| 4 | `mkdir -p build/px4_sitl_default && make px4_sitl_default` | Force CMake to build |
| 5 | `cmake .. -GNinja` (inside build dir) | See exact CMake error if it fails |

| | | |
|---|---|---|
| 6 | `git submodule update --init --recursive` | Ensure full PX4 repo content |

---

Can you please run:

```
cd ~/px4_ws/PX4-Autopilot git describe --tags
```

and paste the output?

That will tell me which PX4 version you're on so I can confirm the correct build target (`px4_sitl_default`, `px4_sitl_clang`, or `px4_sitl_gz`).

**You said:**

I am currently getting this error message from Step 5: CMake Error: The source directory "/Users/user/px4_ws/PX4-Autopilot/build" does not appear to contain CMakeLists.txt.

**ChatGPT said:**

Perfect — that error actually tells us *exactly* what's wrong, and it's an easy fix 👍

Let's go through this carefully so you understand what's happening.

---

## ⚠️ The Problem

The error:

```
CMake Error: The source directory "/Users/user/px4_ws/PX4-Autopilot/build" does not appear to contain CMakeLists.txt.
```

means you ran:

```
cmake ..
```

from the wrong directory level — specifically, inside `build/` instead of `build/px4_sitl_default/`.

PX4's main `CMakeLists.txt` file is located in the root of the PX4 repo, i.e.:

```
/Users/user/px4_ws/PX4-Autopilot/CMakeLists.txt
```

But when you were inside `/Users/user/px4_ws/PX4-Autopilot/build`,

the `..` pointed to `/Users/user/px4_ws/PX4-Autopilot`'s parent directory, not the repo root — hence CMake couldn't find the file.

---

## ✅ The Correct Fix

Let's fix your working directory structure.

Run the following exact sequence:

```
cd ~/px4_ws/PX4-Autopilot rm -rf build mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../.. -GNinja
```

💡 Notice the two dots (`../..`) — that points CMake *two directories up* to the PX4 repo root (where `CMakeLists.txt` is located).

If everything's set up correctly, you'll now see:

```
-- PX4 version: v1.15.0
-- CMake 3.x.x
-- Build type: RelWithDebInfo
-- Configuring done
-- Generating done
-- Build files have been written to:
/Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

Then you can compile:

```
ninja
```

And once built, run SITL:

```
cd ~/px4_ws/PX4-Autopilot make px4_sitl_default gz
```

---

## 🧩 Quick Summary

| Problem | Why It Happened | Fix |
| --- | --- | --- |
| `CMake Error: source directory … does not contain CMakeLists.txt` | You ran CMake from one directory too high | Run `cmake ../.. -GNinja` inside `build/px4_sitl_default/` |

---

Would you like me to confirm whether your PX4 branch supports the `gz` simulator target (Ignition Gazebo)?

If you paste the output of:

```
git describe --tags
```

I can tell you the exact simulator build target for your version (since it changed between PX4 v1.13 → v1.15).

**You said:**
This error occurs: By not providing "FindOpenCV.cmake" in CMAKE_MODULE_PATH this project has
asked CMake to find a package configuration file provided by "OpenCV", but
CMake did not find one.
How would I remedy this?

**ChatGPT said:**
Perfect — that's a classic and very fixable PX4 build issue 🎯

The error means CMake couldn't find OpenCV, which PX4 uses for certain perception and computer vision modules (e.g. optical flow, camera simulation, etc.).

Let's walk through the proper way to fix this on macOS (with Homebrew) — step-by-step.

---

## 🧩1️⃣ Install OpenCV via Homebrew

PX4 requires OpenCV, so let's install it:

```
brew install opencv
```

That will install OpenCV headers, libraries, and CMake config files to:

```
/opt/homebrew/opt/opencv/
```

---

## 🧩2️⃣ Tell CMake Where OpenCV Is

Now we'll explicitly tell PX4's CMake system to look in that directory.

From your PX4 root:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

Then re-run CMake manually with OpenCV's path added:

```
cmake ../.. -GNinja \ -DCMAKE_PREFIX_PATH="/opt/homebrew/opt/opencv" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo
```

If your setup already includes Gazebo paths, you can include both:

```
cmake ../.. -GNinja \
-DCMAKE_PREFIX_PATH="/opt/homebrew/opt/opencv;/opt/homebrew/lib/cmake" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo
```

✅ You should now see:

```
-- Found OpenCV: /opt/homebrew/opt/opencv (found version "4.x.x")
```

Then build PX4 again:

```
ninja
```

---

## 🧩3️⃣ (Optional) Add OpenCV to PATH Permanently

To avoid having to specify this every time, add it to your shell config:

```
echo 'export
CMAKE_PREFIX_PATH="/opt/homebrew/opt/opencv:$CMAKE_PREFIX_PATH"' >>
~/.zshrc source ~/.zshrc
```

Then CMake will find OpenCV automatically in future builds.

---

## 🧩4️⃣(If You Still Get the Error)

You can also explicitly set the OpenCV directory using CMake's `OpenCV_DIR` variable:

```
cmake ../.. -GNinja \
-DOpenCV_DIR="/opt/homebrew/opt/opencv/share/opencv4" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo
```

This points directly to the folder containing `OpenCVConfig.cmake`.

You can verify it exists with:

```
ls /opt/homebrew/opt/opencv/share/opencv4/OpenCVConfig.cmake
```

---

## ✅ TL;DR – The Fix

```
brew install opencv cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default cmake
../.. -GNinja \
-DCMAKE_PREFIX_PATH="/opt/homebrew/opt/opencv;/opt/homebrew/lib/cmake" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo ninja
```

---

If you paste the *exact line* from your CMake error log that says something like

`Could NOT find OpenCV (missing: ... )`,

I can tell you exactly which variable PX4 is trying to use (`OpenCV_DIR` vs.

`CMAKE_PREFIX_PATH`) and tailor the exact command for your setup.

Would you like to do that?

**You said:**

This error displays: The link interface of target "gz-gui8::gz-gui8" contains:

Qt5::Core

but the target was not found.

**ChatGPT said:**

Perfect — that's an excellent catch, and this specific error tells us something very precise about what's missing 👇

---

## ⚠️ The error

```
The link interface of target "gz-gui8::gz-gui8" contains:
    Qt5::Core
  but the target was not found.
```

## 🔍 What it means

- PX4 is trying to link Gazebo Harmonic (gz-gui8) — which depends on Qt5 (Qt Core, Widgets, Quick, etc.).
- But CMake can't find a Qt5 installation on your Mac — or Gazebo was built against Qt5 while your system only has Qt6.
- On macOS, Homebrew defaults to installing Qt6, so you get this mismatch.

In short:

Gazebo Sim (gz-sim8 / gz-gui8) is looking for Qt5, but you only have Qt6 — or no Qt at all.

---

## ✅ Step-by-step fix

Let's fix this properly for your setup (macOS, Homebrew, PX4 SITL).

---

### 🧩1️⃣Confirm your Qt version

Run:

```
brew list | grep qt
```

If you see only `qt` or `qt6` — that's Qt6.

If you see `qt@5` — that's Qt5 (good).

---

## 🧩2️⃣Install Qt5 explicitly

If you don't already have it:

```
brew install qt@5
```

Then link it (Homebrew doesn't link Qt5 automatically):

```
brew link qt@5 --force
```

⚠️ Note: You might get a warning about "keg-only"; that's okay — it just means we're manually linking it.

---

## 🧩3️⃣Let CMake know where Qt5 is

Now we must tell CMake and Gazebo where Qt5 lives.

On Apple Silicon, Qt5 is typically here:

```
/opt/homebrew/opt/qt@5
```

So, in your PX4 build directory, re-run CMake like this:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default cmake ../.. -GNinja \
-DCMAKE_PREFIX_PATH="/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv;/opt/
homebrew/lib/cmake" \ -DQt5_DIR="/opt/homebrew/opt/qt@5/lib/cmake/Qt5" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo
```

---

## 🧩4️⃣Verify detection

You should now see lines like:

```
-- Found Qt5Core
-- Found Qt5Widgets
-- Found gz-gui8
```

✅ That means Qt5 and Gazebo GUI are successfully configured.

Then rebuild:

```
ninja
```

and run:

```
make px4_sitl_default gz
```

---

## 🧩5️⃣ If You Still See Qt Errors

If you still get `Qt5::Core` not found, check this path exists:

```
ls /opt/homebrew/opt/qt@5/lib/cmake/Qt5/Qt5Config.cmake
```

If it does, then try setting `CMAKE_PREFIX_PATH` globally in your shell:

```
echo 'export
CMAKE_PREFIX_PATH="/opt/homebrew/opt/qt@5/lib/cmake:$CMAKE_PREFIX_PATH"'
>> ~/.zshrc source ~/.zshrc
```

---

## ✅ TL;DR — The Fix in One Go

```
brew install qt@5 brew link qt@5 --force cd
~/px4_ws/PX4-Autopilot/build/px4_sitl_default cmake ../.. -GNinja \
-DCMAKE_PREFIX_PATH="/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv;/opt/
homebrew/lib/cmake" \ -DQt5_DIR="/opt/homebrew/opt/qt@5/lib/cmake/Qt5" \
-DCMAKE_BUILD_TYPE=RelWithDebInfo ninja make px4_sitl_default gz
```

---

Would you mind running this and pasting the output?

```
brew list | grep qt
```

That'll tell me whether we should keep using `qt@5` or switch Gazebo to Qt6 (both are possible, but PX4 works best with Qt5).

**You said:**
This error displays: fatal error: "Protobuf C++ gencode is built with an incompatible version of"
16 | #error "Protobuf C++ gencode is built with an incompatible version of"
| ^

**ChatGPT said:**
Excellent — that's a very *precise and diagnostic* error 👏

You've now hit the Protobuf version mismatch problem, which is one of the most common build blockers on macOS when combining PX4 + Gazebo (gz-sim).

Let's go through what's happening and how to fix it safely.

---

## ⚠️ What's Going On

The key part of your error:

```
#error "Protobuf C++ gencode is built with an incompatible version of"
```

means:

You have multiple versions of Protobuf on your system — and Gazebo (gz-sim8) or PX4 was built with one version, while another version is currently being used to compile or link.

This mismatch triggers because:

- Homebrew recently upgraded `protobuf` from v21 → v25 (and v26+),
- but your installed Gazebo Harmonic libraries (`gz-msgs9`, `gz-transport14`) were built against an older protobuf ABI.

So PX4's build links headers from one version and libraries from another → 💥 binary incompatibility.

---

## ✅ The Fix (macOS, Homebrew-based)

There are two possible ways to solve this, depending on how far you want to go.

---

### 🧩1️⃣ Quick Fix: Force Rebuild Gazebo Against Current Protobuf

This is usually all you need.

Run:

```
brew reinstall protobuf brew reinstall gz-harmonic
```

That ensures both Gazebo and Protobuf are built against the same version.

Then clean PX4's build cache:

```
cd ~/px4_ws/PX4-Autopilot make clean rm -rf build
```

Then reconfigure PX4:

```
mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../..
-GNinja \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/qt@5;/opt/h
omebrew/opt/opencv" \ -DCMAKE_BUILD_TYPE=RelWithDebInfo ninja
```

✅ PX4 will now link to the updated Gazebo libs built with the matching Protobuf version.

---

### 🧩2️⃣ If Reinstall Doesn't Fix It

Sometimes Homebrew keeps old cached protobuf headers under `/usr/local/include` or `/opt/homebrew/include`.

Let's check for duplicates.

Run:

```
ls /opt/homebrew/include | grep protobuf
```

If you see both `google/` and something like `protobuf-3.21` folders, there's a mismatch.

To fix, remove the old ones:

```
brew cleanup protobuf
```

Then confirm Gazebo is linking the correct version:

```
otool -L /opt/homebrew/lib/libgz-msgs9.dylib | grep proto
```

You should see:

```
/opt/homebrew/opt/protobuf/lib/libprotobuf.**.dylib
```

matching your installed protobuf.

---

## 🧩3️⃣ Optional: Verify Protobuf version used by Gazebo

```
pkg-config --modversion protobuf
```

and

```
strings /opt/homebrew/lib/libgz-msgs9.dylib | grep "libprotobuf"
```

Both should show the same version number (e.g. 25.x.x).

---

## 🧩4️⃣ Clean PX4's generated Protobuf files (optional)

PX4 generates Protobuf message bindings during build. To remove stale ones:

```
cd ~/px4_ws/PX4-Autopilot find build -name "*.pb.cc" -delete find build -name "*.pb.h" -delete
```

Then re-run:

```
make px4_sitl_default
```

---

## ✅ TL;DR — The Fix

```
brew reinstall protobuf brew reinstall gz-harmonic brew cleanup protobuf
cd ~/px4_ws/PX4-Autopilot make clean rm -rf build mkdir -p
build/px4_sitl_default cd build/px4_sitl_default cmake ../.. -GNinja \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/qt@5;/opt/h
omebrew/opt/opencv" \ -DCMAKE_BUILD_TYPE=RelWithDebInfo ninja
```

---

If this still triggers the same “incompatible version of protobuf” error,

please run:

```
brew info protobuf brew info gz-harmonic
```

and paste the version numbers — I can tell you *exactly* which Gazebo + protobuf combination is currently compatible with your PX4 branch (since PX4 main expects Gazebo Harmonic + protobuf 25.x).

**You said:**
This error displays: Error: The maximum number of open files on this system has been reached.

**ChatGPT said:**
💡 Excellent — that error is *very common* when building large C++ projects like PX4 on macOS. It’s not a code problem — it’s a system resource limit problem.

Let’s walk through how to fix it cleanly.

---

## ⚠️ The error

```
Error: The maximum number of open files on this system has been reached.
```

means your macOS user process hit the OS-imposed limit for open file descriptors — essentially, the number of files/sockets CMake or Ninja can open simultaneously during the PX4 build.

PX4's build system opens thousands of small C++ files, so the default macOS limit (256) is far too low.

---

## ✅ Step 1: Check Your Current Limit

Run:

```
ulimit -n
```

You'll probably see something like:

```
256
```

That's too low for PX4 — you'll want at least 4096, ideally 8192 or higher.

---

## ✅ Step 2: Temporarily Increase It for This Session

You can bump the limit temporarily like this:

```
ulimit -n 8192
```

Then re-run your PX4 build:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default ninja
```

If it builds successfully, you know this was the issue.

---

## ✅ Step 3: Make It Permanent (macOS)

To make the change stick across reboots and terminals, do the following.

### 🔹 Option A — For zsh (your default shell)

Edit your shell startup file:

```
open -a TextEdit ~/.zshrc
```

Add this line:

```
ulimit -n 8192
```

Save and close, then reload:

```
source ~/.zshrc
```

---

### 🔹 Option B — System-wide Launch Limits

If you still get errors even after step 2, macOS's launchd may be capping your limit.

You can override it with a `launchd` plist.

Run:

```
sudo mkdir -p /Library/LaunchDaemons sudo tee
/Library/LaunchDaemons/limit.maxfiles.plist > /dev/null <<EOF <?xml
version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD
PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist
version="1.0"> <dict> <key>Label</key><string>limit.maxfiles</string>
<key>ProgramArguments</key> <array> <string>launchctl</string>
<string>limit</string> <string>maxfiles</string> <string>8192</string>
<string>65536</string> </array> <key>RunAtLoad</key><true/> </dict>
</plist> EOF sudo launchctl load -w
/Library/LaunchDaemons/limit.maxfiles.plist
```

Then reboot your Mac.

After reboot, verify:

```
ulimit -n
```

should show 8192 or higher.

---

## ✅ Step 4: Resume Your Build

Once the limit is raised:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default ninja
```

It should now proceed without the “maximum number of open files” error.

---

## 🧠 Quick TL;DR

| Step | Command | Purpose |
|---|---|---|
| Check current limit | `ulimit -n` | See your current file handle limit |
| Temporarily fix | `ulimit -n 8192` | Raise limit for this shell |
| Make permanent | Add `ulimit -n 8192` to `~/.zshrc` | Persist for every session |
| System-wide (optional) | `limit.maxfiles.plist` | Override global limit |
| Retry build | `ninja` | Should now succeed ✅ |

---

Would you like me to show the quick one-liner to verify both your soft and hard limits (so we can ensure launchd isn’t still constraining it)?

**You said:**
This error displays: fatal error: "C++ versions less than C++17 are not supported."

**ChatGPT said:**
Perfect — that’s actually a very clear, final build configuration issue, and it’s easy to fix. 👋

Let’s go over what it means and exactly how to fix it for your PX4 + Gazebo build on macOS.

---

## ⚠️ The error

```
fatal error: "C++ versions less than C++17 are not supported."
```

## 🔍 What it means

PX4 (and Gazebo Sim) require C++17 or newer,

but your compiler or CMake configuration is currently using an older C++ standard

(likely C++14).

This usually happens when:

- CMake defaults to Apple's Clang without the right `CMAKE_CXX_STANDARD` setting, or
- your environment variables or toolchain files override the C++ standard.

---

## ✅ Step 1: Check Your Compiler

Run:

```
c++ --version
```

You should see something like:

```
Apple clang version 15.0.0 (clang-1500.0.40.1)
```

That's fine — Clang fully supports C++17 and newer.

So the issue is just your CMake configuration, not the compiler itself.

---

## ✅ Step 2: Tell CMake to Use C++17

Let's explicitly tell CMake which C++ version to use.

From your PX4 build directory:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default rm -rf * cmake ../..
-GNinja \ -DCMAKE_CXX_STANDARD=17 \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
```

```
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/opencv;/opt/homebrew/opt/qt@5"
```

Then build again:

```
ninja
```

✅ This enforces C++17 globally and overrides any older default.

---

## ✅ Step 3: (Optional) If PX4 or Gazebo Uses C++20

Some branches of Gazebo Harmonic (gz-sim8) or PX4 main might require C++20.

If you still see the same error after step 2, try:

```
cmake ../.. -GNinja \ -DCMAKE_CXX_STANDARD=20 \
-DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/opencv;/opt/homebrew/opt/qt@5"
```

---

## ✅ Step 4: Check That It's Taking Effect

You can verify what standard PX4 is using by checking the CMake cache:

```
grep CMAKE_CXX_STANDARD CMakeCache.txt
```

It should say:

```
CMAKE_CXX_STANDARD:STRING=17
```

or

```
CMAKE_CXX_STANDARD:STRING=20
```

---

## ✅ Step 5: Make It Permanent (Optional)

If you don't want to keep retyping it, you can export it globally:

```
echo 'export CXXFLAGS="-std=c++17"' >> ~/.zshrc source ~/.zshrc
```

That ensures all C++ builds default to at least C++17.

---

## ✅ TL;DR — The Fix

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p
build/px4_sitl_default cd build/px4_sitl_default cmake ../.. -GNinja \
-DCMAKE_CXX_STANDARD=17 \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/opencv;/opt
/homebrew/opt/qt@5" ninja
```

---

If this still fails, please paste the exact line from your build output that mentions `"C++ versions less than C++17 are not supported"`.

That line will show *which library or dependency* (PX4 core, Gazebo, or Protobuf) is complaining — and I can tell you whether you need to bump to C++20 specifically.

**You said:**
When running Step 2, it asks this: zsh: sure you want to delete all 33 files in /Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default [yn]?
Is it okay to delete these?

**ChatGPT said:**
✅ Yes — it's completely safe (and recommended) to delete those files.

Here's why 👇

---

## 🧩 What's Happening

When you ran:

```
rm -rf *
```

inside your build directory (`~/px4_ws/PX4-Autopilot/build/px4_sitl_default`), macOS's zsh shell is simply asking for confirmation because you're deleting a batch of files manually.

Those 33 files are temporary build artifacts — things like:

- compiled object files (`.o`),
- generated CMake cache/config files, and
- intermediate Ninja build data.

They are not part of your PX4 source code, and CMake will recreate them automatically on the next build.

---

## 🛠️ Why You're Deleting Them

You're cleaning out that directory so that when you re-run CMake,

it starts with a fresh configuration that explicitly sets the C++ standard (C++17 or newer).

This prevents older cached settings (e.g. C++14) from being reused.

---

## ✅ What to Do

Just type:

```
y
```

and press Enter.

Then continue with the commands:

```
cmake ../.. -GNinja \ -DCMAKE_CXX_STANDARD=17 \
-DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/opencv;/opt/homebrew/opt/qt@5" ninja
```

---

## 🧠 Quick Rule of Thumb

It's always safe to delete anything inside a `build/` directory in PX4 or most CMake projects —

never delete files from the root PX4-Autopilot/ folder itself.

---

Would you like me to show you a short "safe clean" command that avoids these zsh prompts in the future?

**You said:**
This error displays: fatal error: "Protobuf C++ gencode is built with an incompatible version of"

**ChatGPT said:**
Excellent — this is the Protobuf ABI mismatch error again, and it's *the* classic build breaker on macOS when mixing PX4, Gazebo, and Homebrew packages.

Let's go through this cleanly and fix it step-by-step.

---

## ⚠️ What's Happening

The key part of your error:

```
#error "Protobuf C++ gencode is built with an incompatible version of"
```

means that one component was compiled using one version of Protobuf, while another (like Gazebo or PX4) is trying to link against a different version.

This can happen when:

- Homebrew upgrades `protobuf` (v21 → v25 → v26, etc.),
- Gazebo or PX4 still links to the older version,
- or multiple versions exist in `/opt/homebrew` and `/usr/local`.

---

## ✅ Step 1: Check Your Installed Protobuf

Run:

```
brew info protobuf
```

You'll see something like:

```
protobuf: stable 25.3 (bottled)
...
/opt/homebrew/Cellar/protobuf/25.3
```

Write down that version (e.g., 25.3).

---

## ✅ Step 2: Rebuild Everything That Uses Protobuf

You need to make sure that both Gazebo and PX4 are built against *this same* Protobuf version.

Run these commands:

```
brew reinstall protobuf brew reinstall gz-harmonic brew reinstall gz-msgs9
brew reinstall gz-transport14 brew cleanup protobuf
```

This ensures all Gazebo libraries are rebuilt using your current `protobuf`.

---

## ✅ Step 3: Clean PX4 Build Cache

PX4 may still have old protobuf-generated files.

Let's delete all build artifacts:

```
cd ~/px4_ws/PX4-Autopilot make clean rm -rf build
```

This will not delete your PX4 source — just build output.

---

## ✅ Step 4: Reconfigure PX4 with Matching CMake Paths

Now re-run the CMake configuration and explicitly tell it where to find your Homebrew libraries:

```
mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../..
-GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -DCMAKE_CXX_STANDARD=17 \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/o
pt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then build again:

```
ninja
```

---

## ✅ Step 5: Verify the Linked Protobuf Version

To double-check that Gazebo and PX4 are using the same Protobuf, run:

```
otool -L /opt/homebrew/lib/libgz-msgs9.dylib | grep proto
```

and:

```
pkg-config --modversion protobuf
```

They should both show the same version, e.g. `25.3`.

If they differ (say Gazebo links to an older one like `21.x`), then Homebrew didn't rebuild the Gazebo libraries properly — we can fix that by manually unlinking old versions.

---

## ✅ Step 6 (if still broken): Remove Old Protobufs

List all protobuf directories:

```
ls /opt/homebrew/Cellar/protobuf
```

If you see multiple versions (e.g., `21.12`, `25.3`, `26.1`), remove the old ones:

```
brew uninstall protobuf@21 brew uninstall protobuf@23 brew cleanup
protobuf
```

Then rebuild Gazebo again:

```
brew reinstall gz-harmonic
```

---

## ✅ TL;DR Fix Summary

```
brew reinstall protobuf gz-harmonic gz-msgs9 gz-transport14 brew cleanup protobuf cd ~/px4_ws/PX4-Autopilot make clean rm -rf build mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -DCMAKE_CXX_STANDARD=17 \ -DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv" ninja
```

---

If you'd like, you can paste the output of:

```
brew info protobuf brew info gz-harmonic
```

and I can confirm whether those versions are *binary-compatible* with PX4's SITL build right now.

**You said:**
The following errors display:
fatal error: 'absl/base/optimization.h' file not found
fatal error: implicit conversion increases floating-point precision: 'const float' to 'const double' [-Wdouble-promotion]

**ChatGPT said:**
Excellent — you're deep into the PX4/Gazebo build process now 🚀

These two errors have separate causes, but both are straightforward to fix on macOS with Homebrew.

Let's handle them one at a time. 👇

---

## ⚠️1️⃣ Error: `'absl/base/optimization.h' file not found`

## 🔍 Meaning

This means PX4 or one of its dependencies (often `gz-msgs` or `protobuf`) is trying to include the Abseil C++ library (`absl::`), but it isn't installed — or CMake can't find its include path.

Abseil is a dependency of modern Protobuf (v25+).

---

## ✅ Fix

Install Abseil via Homebrew and make sure it's in your CMake path:

```
brew install abseil
```

Then clean your PX4 build cache:

```
cd ~/px4_ws/PX4-Autopilot make clean rm -rf build
```

Then reconfigure and build again, explicitly adding Abseil's include path:

```
mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../..
-GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -DCMAKE_CXX_STANDARD=17 \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/o
pt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
ninja
```

✅ This should fix the missing `absl` headers error.

---

## ⚠️2️⃣Error:

```
implicit conversion increases floating-point precision: 'const float' to
'const double' [-Wdouble-promotion]
```

## 🔍 Meaning

This isn't a fatal build error *unless* your compiler treats warnings as errors (`-Werror`).

PX4's code mixes `float` and `double` types in certain math expressions — totally safe, but Clang warns about it.

---

## ✅ Fix Option A — Turn Off That Warning

If it stops the build, you can safely tell CMake to ignore `-Wdouble-promotion` warnings:

Add this flag during CMake configuration:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv" \
-DCMAKE_CXX_FLAGS="-Wno-double-promotion"
```

Then rebuild:

```
ninja
```

✅ This will silence the harmless warning and let the build continue.

---

## ✅ Fix Option B — Allow Warnings Without Stopping the Build

Some PX4 builds treat warnings as errors by default.

You can disable that behavior globally:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then:

```
ninja
```

---

## ✅ TL;DR Summary

| Error | Fix |
| --- | --- |
| `'absl/base/optimization.h' file not found` | `brew install abseil` and add `/opt/homebrew/opt/abseil` to `CMAKE_PREFIX_PATH` |
| `implicit conversion increases floating-point precision` | Add `-Wno-double-promotion` or `-DPX4_WARNINGS_AS_ERRORS=OFF` |

---

## Example Combined Command (safe, all fixes)

```
cd ~/px4_ws/PX4-Autopilot make clean rm -rf build mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_FLAGS="-Wno-double-promotion" \ -DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv" ninja
```

---

If the `absl` error persists *after installing it*, please share the output of:

```
brew --prefix abseil
```

and

```
ls /opt/homebrew/include/absl/base/
```

— that will confirm whether CMake is seeing the correct Abseil headers.

**You said:**

This error displays: ninja: error: loading 'build.ninja': No such file or directory

**ChatGPT said:**

Perfect — that's actually a simple, non-fatal build setup issue 👍

This just means that CMake never finished generating the Ninja build files, so Ninja doesn't know what to build yet.

Let's fix it step-by-step 👇

---

## ⚠️ The error

```
ninja: error: loading 'build.ninja': No such file or directory
```

### 🔍 What it means

Ninja expects a `build.ninja` file inside your build folder — this file is created by CMake during the configuration step.

If it's missing, then:

- either CMake failed (silently or with an error),
- or you ran `ninja` in the wrong directory,
- or the folder was cleaned before running CMake.

---

## ✅ Step 1: Make sure you're in the correct directory

Run:

```
pwd
```

It should end with:

```
.../PX4-Autopilot/build/px4_sitl_default
```

If not, move there:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

---

## ✅ Step 2: Re-run the CMake configuration

We need to regenerate the build system before running `ninja`.

Run this exact command again (this is safe and necessary):

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \
-DCMAKE_CXX_FLAGS="-Wno-double-promotion" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

You should see output like:

```
-- The C compiler identification is AppleClang 15.0.0
-- The CXX compiler identification is AppleClang 15.0.0
-- Configuring done
-- Generating done
-- Build files have been written to:
/Users/you/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

That last line confirms the `build.ninja` file has been created ✅

---

## ✅ Step 3: Now build

Once CMake finishes successfully, run:

```
ninja
```

Now it should work — Ninja will use the new build system generated by CMake.

---

## ⚠️ Step 4: If CMake Fails Again

If CMake prints any errors, stop there — don't run Ninja.

Instead, copy/paste the first 10–15 lines of the CMake output, and I'll tell you exactly which dependency or path is missing.

Common issues:

- Missing Abseil or Protobuf headers
- Wrong CMake prefix paths
- CMake version < 3.22 (should be ≥ 3.25)

You can check your CMake version with:

```
cmake --version
```

If it's too old:

```
brew upgrade cmake
```

---

## ✅ TL;DR

| Problem | Fix |
|---|---|
| `ninja: error: loading 'build.ninja'` | Run `cmake ../.. -GNinja ...` inside your `build/px4_sitl_default` directory first |
| CMake errors | Fix those before running Ninja |
| Wrong directory | Make sure you're in `PX4-Autopilot/build/px4_sitl_default` |

---

If you want, paste your last few lines of CMake output — I'll check whether the configuration completed correctly (and confirm that `build.ninja` should exist).

**You said:**
This warning displays: CMake Warning at src/modules/simulation/gz_plugins/gstreamer/CMakeLists.txt:42 (message):
  GStreamer and Gstreamer App are required to build this plugin
Is this necessary?

**ChatGPT said:**
💯 Excellent question — and you caught a *non-critical but common* build message for PX4's Gazebo integration.

Let's unpack it clearly so you know whether to worry about it 👇

---

## ⚠️ The warning

```
CMake Warning at
src/modules/simulation/gz_plugins/gstreamer/CMakeLists.txt:42 (message):
  GStreamer and Gstreamer App are required to build this plugin
```

---

## 🧩 What it means

This is not an error, just a notice from CMake.

PX4 includes an optional Gazebo plugin for video streaming (the “GStreamer camera plugin”), which lets you:

- Stream simulated camera output (e.g. FPV or RGB cameras) over UDP or RTSP.
- Use real-time video feeds in SITL.

However, if GStreamer isn’t installed on your system, PX4 simply skips building that plugin — and the rest of PX4 and Gazebo still compile and run normally.

---

## ✅ TL;DR — Is it necessary?

| Situation | Required? | Notes |
|---|---|---|
| You just want to run PX4 SITL (Simulated drone in Gazebo) | ❌ Not needed | Safe to ignore this warning. |
| You want to stream video from simulated cameras (e.g. to QGroundControl or ROS) | ✅ Required | You’ll need GStreamer and its dev libraries. |

---

## ⚙️ Optional — If You Want to Enable It

If you *do* want the streaming plugin (e.g. for camera simulation), you can install GStreamer on macOS:

```
brew install gst-plugins-base gst-plugins-good gst-libav gstreamer
```

Then clean and rebuild PX4:

```
cd ~/px4_ws/PX4-Autopilot make clean rm -rf build/px4_sitl_default mkdir -p build/px4_sitl_default cd build/px4_sitl_default cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv;/opt/homebrew/opt/gstreamer" ninja
```

That will enable the plugin automatically once CMake detects GStreamer.

---

## ✅ In summary

- The warning is harmless.
- PX4 SITL will still run perfectly fine without GStreamer.
- Only install it if you need video streaming from simulated cameras.

---

Would you like me to show you how to verify later (after build) whether the camera streaming plugin was compiled or skipped? It's a simple one-line check.

**You said:**
Even after running the above code for build.ninja, this error still displays:

**ChatGPT said:**
Got it 👍 — so even after running the full CMake configuration command, you're still seeing

```
ninja: error: loading 'build.ninja': No such file or directory
```

That means CMake didn't actually generate the build system (it failed silently or stopped early).

Let's go step-by-step to figure out *why* and fix it cleanly 👇

---

## 🧩 Step 1: Verify You're in the Correct Folder

Please confirm you're inside your PX4 build directory:

```
pwd
```

Expected output should end with:

```
/Users/<yourname>/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

If not, fix it:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

---

## 🧩 Step 2: Check What Happened During CMake

Re-run only the CMake command and look closely at the last few lines of output:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \
-DCMAKE_CXX_FLAGS="-Wno-double-promotion" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Now — check the last 10–20 lines of the output.

If you don't see:

```
-- Configuring done
-- Generating done
-- Build files have been written to: ...
```

then CMake failed to configure the build.

That's why `build.ninja` doesn't exist yet.

---

## 🧩 Step 3: Confirm CMake Version

PX4 requires CMake ≥ 3.22.

Run:

```
cmake --version
```

If it's older (e.g. 3.19, 3.21), update it:

```
brew upgrade cmake
```

Then retry the CMake command above.

---

## 🧩 Step 4: Check for Hidden Error Messages

Sometimes CMake's error messages scroll past quickly.

Let's capture them to a log file so we can inspect:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \
-DCMAKE_CXX_FLAGS="-Wno-double-promotion" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv" \
> cmake_log.txt 2>&1
```

Then view the last few lines:

```
tail -n 30 cmake_log.txt
```

Please copy those last lines here — that's where the real reason lies (e.g. "could not find Protobuf", "no suitable compiler", etc.).

---

## 🧩 Step 5: Check Directory Contents

Let's confirm whether CMake generated anything at all:

```
ls
```

If you don't see files like `CMakeCache.txt` or `CMakeFiles/`,

then CMake never wrote output — confirming the configuration failed.

---

## ✅ What To Expect Once It's Fixed

A successful CMake step ends with:

```
-- Configuring done
-- Generating done
-- Build files have been written to:
/Users/<you>/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

And your directory will contain:

```
CMakeCache.txt
CMakeFiles/
build.ninja
rules.ninja
```

After that, `ninja` will run fine.

---

So — please run:

```
tail -n 30 cmake_log.txt
```

after re-running the CMake command,

and paste the output here — I'll pinpoint *exactly* which dependency or flag is blocking

generation of `build.ninja`.

**You said:**

-- Searching for <gz-sensors8> component [rgbd_camera]
-- Looking for gz-sensors8-rgbd_camera -- found version 8.2.2
-- Searching for dependencies of gz-sensors8-rgbd_camera
-- Looking for gz-sensors8 -- found version 8.2.2

-- Searching for <gz-sensors8> component [thermal_camera]
-- Looking for gz-sensors8-thermal_camera -- found version 8.2.2
-- Searching for dependencies of gz-sensors8-thermal_camera
-- Looking for gz-sensors8 -- found version 8.2.2
-- Searching for <gz-sensors8> component [wide_angle_camera]
-- Looking for gz-sensors8-wide_angle_camera -- found version 8.2.2
-- Searching for dependencies of gz-sensors8-wide_angle_camera
-- Looking for gz-rendering8 -- found version 8.2.2
-- Looking for gz-math7 -- found version 7.5.2
-- Looking for gz-utils2 -- found version 2.2.1
-- Looking for gz-sensors8 -- found version 8.2.2
-- Looking for gz-plugin2 -- found version 2.0.4
-- Found OpenCV: /opt/homebrew/Cellar/opencv/4.12.0_12 (found version "4.12.0")
-- Found OpenCV: /opt/homebrew/Cellar/opencv/4.12.0_12 (found version "4.12.0")
found components: core imgproc
-- Checking for module 'gstreamer-1.0'
--   Package 'gstreamer-1.0' not found
-- Checking for module 'gstreamer-app-1.0'
--   Package 'gstreamer-app-1.0' not found
CMake Warning at src/modules/simulation/gz_plugins/gstreamer/CMakeLists.txt:42 (message):
  GStreamer and Gstreamer App are required to build this plugin.


-- ROMFS: ROMFS/px4fmu_common
-- Configuring done (2.0s)
-- Generating done (0.3s)
-- Build files have been written to: /Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default

**ChatGPT said:**

✅ Perfect — that's actually *great news*!

This output means your CMake configuration succeeded fully 🎉

Let's break it down quickly 👇

---

## 🔍 What happened

Your log shows all dependencies were found successfully:

- ✅ Gazebo (gz-sensors8, gz-rendering8, gz-utils2, etc.) → all detected correctly
- ✅ OpenCV 4.12.0 → found
- ⚠️ GStreamer → *not found* (optional)
- ✅ CMake finished configuring and generating the build system

And most importantly, these two lines at the bottom:

```
-- Configuring done (2.0s)
-- Generating done (0.3s)
-- Build files have been written to:
/Users/user/px4_ws/PX4-Autopilot/build/px4_sitl_default
```

✅ That's exactly what you want.

It means CMake successfully created the file Ninja needs — `build.ninja`.

---

## 🧩 What's Next

Now you can safely run the actual build:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default ninja
```

That will start compiling PX4 SITL with all Gazebo integration.

---

## 🧠 Note about the GStreamer Warning

You can completely ignore:

```
GStreamer and Gstreamer App are required to build this plugin
```

unless you specifically want camera video streaming from Gazebo.

It does *not* affect flight simulation.

---

## ✅ TL;DR

Everything is working.

Your build folder is correctly configured.

Next step is simply:

```
ninja
```

---

Would you like me to show what a *successful* end of the `ninja` build looks like (so you know what to expect before running SITL)?

**You said:**
This error displays: CMake Error at external/klt_feature_tracker/CMakeLists.txt:1 (cmake_minimum_required):
  Compatibility with CMake < 3.5 has been removed from CMake.

**ChatGPT said:**
Excellent catch — and good news: this is an easy fix 🎯

That specific error:

```
CMake Error at external/klt_feature_tracker/CMakeLists.txt:1
(cmake_minimum_required):
  Compatibility with CMake < 3.5 has been removed from CMake.
```

means that one of PX4's third-party dependencies (`klt_feature_tracker`) is using an old CMake syntax that's no longer valid in your current CMake version (CMake ≥ 3.30).

---

## 🧠 Why this happens

PX4 bundles several external modules under `external/`, and some (like `klt_feature_tracker`) were written when CMake 3.5 was current.

Recent CMake versions (especially 3.30+, which comes with Homebrew on macOS ARM) removed compatibility shims for older commands, causing this exact error.

---

## 🛠️ 3 Working Solutions

You have a few ways to fix or bypass it, depending on your goals:

---

### ✅ Option 1: Patch the CMakeLists.txt (recommended)

Edit the first line of this file:

```
~/px4_ws/PX4-Autopilot/external/klt_feature_tracker/CMakeLists.txt
```

and replace:

```
cmake_minimum_required(VERSION 2.8.3)
```

with:

```
cmake_minimum_required(VERSION 3.16)
```

(Or any version between 3.16 and 3.25 — 3.16 is a safe baseline.)

Then re-run:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default cmake ../.. -GNinja ninja
```

That should fix the issue.

---

### ⚙️ Option 2: Remove the legacy module (if you don't need KLT)

If you're just building PX4 SITL for simulation, the KLT feature tracker is *not required*.

You can safely remove or disable it:

```
rm -rf ~/px4_ws/PX4-Autopilot/external/klt_feature_tracker
```

Then rerun CMake and Ninja as above.

PX4 will skip that module automatically.

---

### 🧰 Option 3: Pin an older CMake version

If you prefer not to edit PX4 source files, you can install an older CMake (like 3.25) that still allows older syntax:

```
brew unlink cmake brew install cmake@3.25 brew link cmake@3.25
```

Then confirm:

```
cmake --version
```

and rebuild.

This avoids touching PX4's files but is less ideal if you're doing modern development.

---

## ✅ Recommended Path

🔧 Edit the CMakeLists.txt (Option 1).
It's clean, forward-compatible, and won't affect other parts of PX4.

---

Would you like me to show exactly what the top of that file should look like after the fix (with context lines included)?

**You said:**
After running option 2, this error displays: fatal error: 'absl/base/optimization.h' file not found

**ChatGPT said:**
Good progress — you've moved past the old `klt_feature_tracker` issue 🎯

Now the new error:

```
fatal error: 'absl/base/optimization.h' file not found
```

means that PX4 (or one of its Gazebo dependencies) is trying to compile against Abseil, but your compiler can't locate Abseil's header files.

---

## 🧩 Why this happens

Gazebo (Ignition/GZ) and Protobuf both depend on Abseil, a C++ utility library from Google.

On macOS with Homebrew, Abseil is installed here:

```
/opt/homebrew/opt/abseil/include
```

But PX4's CMake configuration doesn't always automatically detect it — especially if you've manually set `CMAKE_PREFIX_PATH`.

So you need to tell CMake explicitly where Abseil lives.

---

## ✅ Fix

### Step 1: Install (or reinstall) Abseil

Run:

```
brew install abseil
```

If it's already installed:

```
brew reinstall abseil
```

---

### Step 2: Clean the build folder

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p build/px4_sitl_default cd build/px4_sitl_default
```

---

### Step 3: Re-run CMake, explicitly adding Abseil include paths

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DPX4_WARNINGS_AS_ERRORS=OFF \
-DCMAKE_CXX_FLAGS="-Wno-double-promotion
-I/opt/homebrew/opt/abseil/include" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/o
pt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then build:

```
ninja
```

---

## 🧠 If It Still Fails

Run:

```
ls /opt/homebrew/opt/abseil/include/absl/base/
```

If you don't see `optimization.h` there, it means your Abseil install is broken — Homebrew might have renamed the folder (`absl/base/internal/` for newer versions).

In that case, re-link it manually:

```
brew link abseil --force
```

Then retry CMake.

---

## ✅ TL;DR

| Step | Command |
|---|---|
| Ensure Abseil is installed | `brew reinstall abseil` |
| Clean build | `rm -rf build/px4_sitl_default` |
| Reconfigure with Abseil include path | add `-I/opt/homebrew/opt/abseil/include` |

| | |
|---|---|
| Rebuild | `ninja` |

---

Would you like me to show you exactly how to verify whether PX4's CMake successfully detects Abseil (so we can confirm before rebuilding)?

**You said:**
These two errors occur:
fatal error: "C++ versions less than C++17 are not supported."
fatal error: implicit conversion increases floating-point precision: 'const float' to 'const double' [-Wdouble-promotion]

**ChatGPT said:**
Excellent — you're almost there 🚀

Those two errors tell us two important things about your build environment and flags.

Let's fix both cleanly and permanently.

---

## 🧩 The Two Errors Explained

1️⃣

```
fatal error: "C++ versions less than C++17 are not supported."
```

Meaning:

One of the dependencies (very likely Gazebo or Abseil) was compiled with the expectation of C++17, but your PX4 build is currently defaulting to C++14 or older.

Even though we passed `-DCMAKE_CXX_STANDARD=17`, sometimes the flag is ignored if a cached build configuration or an older compiler setting remains active.

---

2️⃣

```
fatal error: implicit conversion increases floating-point precision:
'const float' to 'const double' [-Wdouble-promotion]
```

Meaning:

That’s just a warning (but PX4 treats warnings as errors by default).

We can safely disable this behavior.

---

## ✅ Fix (Step-by-Step)

Let’s rebuild with clean, explicit flags.

---

### 🧹 Step 1: Clean the build

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p
build/px4_sitl_default cd build/px4_sitl_default
```

---

### ⚙️ Step 2: Explicitly set compiler and C++17 standard

Run this CMake command exactly as-is:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DCMAKE_CXX_EXTENSIONS=OFF \
-DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_FLAGS="-std=c++17
-Wno-double-promotion" \ -DCMAKE_EXE_LINKER_FLAGS="-std=c++17" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/o
pt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

---

### 🔨 Step 3: Build

```
ninja
```

This explicitly forces the compiler to use C++17 and disables treating warnings as errors (including `-Wdouble-promotion`).

---

## 🧠 Why this works

- `-std=c++17` overrides the compiler default.
- `-DPX4_WARNINGS_AS_ERRORS=OFF` allows builds to continue even if warnings appear.
- `-Wno-double-promotion` silences the float/double conversion warning.

PX4 and Gazebo's modern components *require* C++17 for full compatibility with Abseil and Protobuf v23+.

---

## ✅ Quick Verification (Optional)

After running CMake, you should see this line near the top of the output:

```
-- The CXX compiler identification is AppleClang ...
-- Check for working CXX compiler: ...
-- Detecting CXX compile features - done
-- Performing Test CMAKE_CXX_STANDARD - 17
```

That confirms your build is now using C++17.

---

If you still see either of those errors after doing this, please paste the first 10 lines of `CMake` output (it shows the compiler and C++ standard detection) — we'll confirm whether your compiler or CMake cache is overriding the setting.

**You said:**
This error displays: fatal error: 'absl/base/optimization.h' file not found

**ChatGPT said:**
Perfect — that message is very specific and tells us exactly what's wrong and how to fix it.

The error

```
fatal error: 'absl/base/optimization.h' file not found
```

means your compiler can't locate Abseil's header files — a dependency required by Protobuf and Gazebo.

Let's fix this cleanly 👇

---

## 🧩 Why this happens

Even though you've likely installed Abseil (`brew install abseil`),

CMake isn't finding it automatically.

On macOS with Homebrew, the include directory is usually:

```
/opt/homebrew/opt/abseil/include
```

PX4 (and Gazebo) sometimes need an explicit hint for where to find it.

---

## ✅ Step-by-step Fix

### 1️⃣ Ensure Abseil is installed properly

Run:

```
brew reinstall abseil
```

Then confirm that the header exists:

```
ls /opt/homebrew/opt/abseil/include/absl/base/optimization.h
```

✅ If the file exists — good.

❌ If it says "No such file or directory," the Homebrew package might have changed structure (rare, but fixable).

---

### 2️⃣ Clean your build folder

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p
build/px4_sitl_default cd build/px4_sitl_default
```

---

### 3️⃣ Explicitly point CMake to Abseil

Run this exact command:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DCMAKE_CXX_STANDARD=17 \ -DCMAKE_CXX_EXTENSIONS=OFF \
-DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_FLAGS="-std=c++17
-Wno-double-promotion -I/opt/homebrew/opt/abseil/include" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/abseil;/opt
/homebrew/opt/protobuf;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then:

```
ninja
```

---

## 🧠 If It Still Fails

### 🔍 Option A: Check Abseil headers

Run:

```
find /opt/homebrew -name optimization.h
```

If it shows a different path (for example,

`/opt/homebrew/include/absl/base/optimization.h`),

update your CMake command accordingly:

```
-DCMAKE_CXX_FLAGS="-I/opt/homebrew/include"
```

---

### 🔍 Option B: Re-link Abseil (sometimes needed)

```
brew unlink abseil && brew link abseil --force
```

Then rerun the CMake command above.

---

## ✅ TL;DR

| Step | Command |
|---|---|
| Reinstall Abseil | `brew reinstall abseil` |
| Clean build | `rm -rf build/px4_sitl_default` |
| Reconfigure | include `-I/opt/homebrew/opt/abseil/include` in CMake flags |
| Build | `ninja` |

---

If you want, paste the output of:

```
ls /opt/homebrew/opt/abseil/include/absl/base/
```

and I'll tell you the exact include flag your compiler needs (since paths can differ slightly across Homebrew versions).

**You said:**
This error displays: fatal error: "C++ versions less than C++17 are not supported."

**ChatGPT said:**
That's a really good sign — it means you're extremely close to a successful PX4 + Gazebo build 🎯

This particular error:

```
fatal error: "C++ versions less than C++17 are not supported."
```

tells us the compiler is still defaulting to an older C++ standard, even though we've asked for C++17.

On macOS (especially with Apple Clang), CMake sometimes ignores `-DCMAKE_CXX_STANDARD=17` when cached values or old CMakeLists reset it.

Let’s fix that cleanly and permanently 👇

---

## ✅ Step-by-Step Fix

### 🧹1️⃣Clean the Build Completely

You need to ensure CMake doesn’t reuse an older configuration:

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p
build/px4_sitl_default cd build/px4_sitl_default
```

---

### ⚙️2️⃣Re-Run CMake with Explicit Compiler Flags

Run this exact command:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_STANDARD=17 \
-DCMAKE_CXX_EXTENSIONS=OFF \ -DCMAKE_CXX_FLAGS="-std=c++17
-Wno-double-promotion -I/opt/homebrew/opt/abseil/include" \
-DCMAKE_EXE_LINKER_FLAGS="-std=c++17" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/abseil;/opt
/homebrew/opt/protobuf;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then:

```
ninja
```

---

### 🔍3️⃣Confirm the Detected Standard

When you run `cmake`, you should see a line like:

```
-- Detecting CXX compile features - done
-- Performing Test CMAKE_CXX_STANDARD - 17
```

If it instead shows 14 or lower, something is overriding the flag.

---

## 🧰4️⃣ Optional: Force the Compiler Used

Sometimes CMake picks an older AppleClang.

You can explicitly set it:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_STANDARD=17 \
-DCMAKE_CXX_EXTENSIONS=OFF \ -DCMAKE_C_COMPILER=/usr/bin/clang \
-DCMAKE_CXX_COMPILER=/usr/bin/clang++ \ -DCMAKE_CXX_FLAGS="-std=c++17
-Wno-double-promotion -I/opt/homebrew/opt/abseil/include" \
-DCMAKE_EXE_LINKER_FLAGS="-std=c++17" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/abseil;/opt
/homebrew/opt/protobuf;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

---

## 🧠 Why this Works

- `-std=c++17` forces the compiler to use the correct language level even if PX4's CMake logic resets it.
- `-DCMAKE_CXX_EXTENSIONS=OFF` prevents CMake from silently downgrading to `gnu++14`.
- Rebuilding from a clean directory removes cached flags from old builds.

---

## ✅ After Success

When the configuration and build succeed, you should be able to run:

```
make px4_sitl_default
```

or

```
make px4_sitl gz
```

(depending on your Gazebo setup).

---

If it *still* shows the same C++ version error, please paste the first 20 lines of the CMake output — that's where it reveals which compiler and standard were actually detected, so I can pinpoint the override.

**You said:**
This error displays: make: *** No rule to make target `px4_sitl'.  Stop.

**ChatGPT said:**
Excellent — that's a different issue now, and it tells us something fundamental:

PX4 itself hasn't been fully built/configured for SITL yet, so the target `px4_sitl` doesn't exist in your build system.

Let's fix this cleanly and get PX4 SITL running with Gazebo.

---

## 🧩 Why You See

```
make: *** No rule to make target `px4_sitl'.  Stop.
```

That message means:

- You're in the wrong directory, or
- CMake didn't finish configuring PX4's simulation (SITL) build targets.

When you use `make px4_sitl`, that only works from the root PX4 source directory, not inside the `build/` folder.

---

## ✅ Step-by-Step Fix

### 🧭1️⃣Confirm Your Location

Run:

```
pwd
```

If it ends with:

```
.../PX4-Autopilot/build/px4_sitl_default
```

```
```

then you're too deep in the folder hierarchy.

PX4's `make` targets (like `px4_sitl`) can only be invoked from the top-level PX4-Autopilot directory.

So go back up:

```
cd ~/px4_ws/PX4-Autopilot
```

---

## ⚙️2️⃣ Build PX4 SITL (Correct Command)

Now run:

```
make px4_sitl_default
```

That tells PX4's top-level make system to:

- Generate the build directory,
- Configure it with CMake, and
- Compile the SITL binary.

If you want Gazebo integration (new Gazebo Sim):

```
make px4_sitl gz_sim
```

---

## 🧠3️⃣ If It Still Fails with "No Rule to Make Target"

If you see the same error again *even from the root directory*, check whether the `Makefile` exists:

```
ls ~/px4_ws/PX4-Autopilot/Makefile
```

If it's missing, you might be in the wrong directory or have an incomplete clone.

Re-clone PX4 cleanly:

```
cd ~/px4_ws git clone https://github.com/PX4/PX4-Autopilot.git cd PX4-Autopilot
```

Then try:

```
make px4_sitl_default
```

---

## 🧰4️⃣ Check Available Build Targets (Optional)

To see what SITL targets exist:

```
make help | grep sitl
```

You should see something like:

```
px4_sitl
px4_sitl_default
px4_sitl_default-clang
```

If you only see those and no `gz_sim`, it means Gazebo Sim support wasn't enabled — which is okay; `px4_sitl_default` alone runs SITL with JMAVSim or headless Gazebo.

---

## ✅ TL;DR

| Step | Command |
|---|---|
| Move to PX4 root | `cd ~/px4_ws/PX4-Autopilot` |
| Build PX4 SITL | `make px4_sitl_default` |
| (Optional) Gazebo Sim | `make px4_sitl gz_sim` |

---

If you run `make help | grep sitl` again and paste the output,

I can confirm exactly which simulator targets are active in your setup.

**You said:**
These errors display:

fatal error: use of undeclared identifier 'typeof'; did you mean 'typeid'?
fatal error: "C++ versions less than C++17 are not supported."

**ChatGPT said:**

Excellent — this combo of errors tells us something *very specific* about your toolchain setup. Let's go step-by-step and fix it cleanly.

---

## 🧩 The two errors

```
fatal error: use of undeclared identifier 'typeof'; did you mean 'typeid'?
fatal error: "C++ versions less than C++17 are not supported."
```

### What's really going on

- `typeof` is a GCC/GNU extension, not part of standard C++.
  Apple's Clang (the default compiler on macOS) doesn't support it unless it's in GNU extension mode (`-std=gnu++17` instead of `-std=c++17`).
- The C++17 error means your compiler is still being invoked with an older language standard (e.g. C++14).

So, the current configuration is:

using Clang in strict C++14 mode → rejects `typeof` and fails the C++17 requirement.

---

## ✅ Fix it permanently

### 1️⃣ Clean the build

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p
build/px4_sitl_default cd build/px4_sitl_default
```

### 2️⃣ Re-run CMake in GNU++17 mode

This lets Clang accept `typeof` while still satisfying PX4's C++17 requirement.

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_STANDARD=17 \
```

```
-DCMAKE_CXX_EXTENSIONS=ON \ -DCMAKE_CXX_FLAGS="-std=gnu++17
-Wno-double-promotion -I/opt/homebrew/opt/abseil/include" \
-DCMAKE_EXE_LINKER_FLAGS="-std=gnu++17" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/abseil;/opt
/homebrew/opt/protobuf;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then build:

```
ninja
```

---

## 3️⃣ Verify during configuration

In the CMake output, look for:

```
-- Performing Test CMAKE_CXX_STANDARD - 17
-- C++ compiler is AppleClang ...
```

and ensure it doesn't downgrade to C++14.

---

## 4️⃣ Optional sanity checks

- Check your compiler version:
- `clang++ --version`
  Should show something like `Apple clang version 16.x` or newer.
- If you have GCC installed via Homebrew and want to use it instead (for stricter GNU compatibility):
- `brew install gcc cmake ../.. -GNinja -DCMAKE_C_COMPILER=gcc-14 -DCMAKE_CXX_COMPILER=g++-14 ...`
  (Replace `14` with whatever version Homebrew installed.)

---

## ✅ TL;DR

| Problem | Fix |
|---|---|
| `typeof` not recognized | Enable GNU extensions → `-std=gnu++17` |

| | |
|---|---|
| C++17 not enabled | Force compiler flags and disable cached configs |
| Using AppleClang | Works fine with `gnu++17` |

---

Once you re-run CMake with `-std=gnu++17`, both errors should disappear.

Would you like me to show you how to check which file or dependency is introducing the `typeof` so we can confirm it's being handled correctly?

**You said:**
This error displays: fatal error: "C++ versions less than C++17 are not supported."

**ChatGPT said:**
Perfect — that error means your build environment is still compiling with an older C++ standard (e.g. C++14 or even C++11), despite you trying to force C++17.

Let's go step-by-step and fix this for sure on macOS.

You're extremely close — this is usually the *last roadblock* before PX4 SITL builds successfully.

---

## 🧩 Why this happens

PX4 uses CMake to generate its build rules.

When you see:

```
fatal error: "C++ versions less than C++17 are not supported."
```

it means:

- Your compiler (likely Apple Clang) is fine — it supports C++17.
- But your CMake configuration is either:
  - using cached flags from a previous (older) run, or
  - a dependency's CMakeLists.txt is overriding `CMAKE_CXX_STANDARD`.

So even though you passed `-DCMAKE_CXX_STANDARD=17`, the active build target is still being compiled as C++14 or less.

---

## ✅ Fix (Step-by-step)

### 🧹1️⃣Clean everything to remove cached flags

```
cd ~/px4_ws/PX4-Autopilot rm -rf build/px4_sitl_default mkdir -p build/px4_sitl_default cd build/px4_sitl_default
```

---

### ⚙️2️⃣Force C++17 explicitly and override everything

Run this exact command:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \ -DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_CXX_STANDARD=17 \ -DCMAKE_CXX_EXTENSIONS=OFF \ -DCMAKE_CXX_FLAGS="-std=c++17 -Wno-double-promotion -I/opt/homebrew/opt/abseil/include" \ -DCMAKE_EXE_LINKER_FLAGS="-std=c++17" \ -DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/opt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then:

```
ninja
```

---

### 🧠3️⃣Check that CMake actually applies it

In your CMake output, you must see something like:

```
-- The CXX compiler identification is AppleClang ...
-- Detecting CXX compile features - done
-- Performing Test CMAKE_CXX_STANDARD - 17
```

If it shows 14 or less, it means something inside PX4 or a dependency is overriding it.

---

## 🧰4️⃣(Optional but more forceful): Set the compiler explicitly

Apple Clang sometimes uses older flags when invoked via aliases.

Force it like this:

```
cmake ../.. -GNinja \ -DCMAKE_BUILD_TYPE=RelWithDebInfo \
-DPX4_WARNINGS_AS_ERRORS=OFF \ -DCMAKE_C_COMPILER=/usr/bin/clang \
-DCMAKE_CXX_COMPILER=/usr/bin/clang++ \ -DCMAKE_CXX_STANDARD=17 \
-DCMAKE_CXX_EXTENSIONS=OFF \ -DCMAKE_CXX_FLAGS="-std=c++17
-Wno-double-promotion -I/opt/homebrew/opt/abseil/include" \
-DCMAKE_EXE_LINKER_FLAGS="-std=c++17" \
-DCMAKE_PREFIX_PATH="/opt/homebrew/lib/cmake;/opt/homebrew/opt/protobuf;/o
pt/homebrew/opt/abseil;/opt/homebrew/opt/qt@5;/opt/homebrew/opt/opencv"
```

Then rebuild:

```
ninja
```

---

## 🧾5️⃣Verify your compiler version

Check that you're not using an outdated system Clang:

```
clang++ --version
```

Expected output (roughly):

```
Apple clang version 16.0.0 (clang-1600.0.25.1)
Target: arm64-apple-darwin23.x.x
```

If it shows something much older (like Apple Clang 12 or 13), update via:

```
xcode-select --install
```

---

## ✅ TL;DR Summary

| Problem | Root Cause | Fix |
| --- | --- | --- |

| | | |
|---|---|---|
| “C++ versions less than C++17 are not supported.” | CMake using cached/overridden flags | Clean build + explicit `-std=c++17` |
| Apple Clang strict mode | Missing `-std=c++17` flag passed through | Add `-DCMAKE_CXX_FLAGS="-std=c++17"` |
| Stale build config | `build.ninja` using old settings | Delete `build/px4_sitl_default` and re-run |

---

If it still gives that error after this exact command, please copy-paste the first 20–25 lines of your CMake output — that’s where it reveals what compiler and flags it’s actually using, and I’ll pinpoint the override.

**You said:**
This error displays: fatal error: use of undeclared identifier 'typeof'; did you mean 'typeid'?

**ChatGPT said:**
Ah — now we’ve hit a classic macOS / Apple Clang issue. That error:

```
fatal error: use of undeclared identifier 'typeof'; did you mean 'typeid'?
```

means that the code is using the GNU/C compiler extension `typeof`, which Apple Clang does not support. On Clang, the equivalent is `__typeof__`. PX4 and Gazebo sometimes pull in older dependencies that rely on `typeof`.

---

## 🧩 Why this happens

- Some PX4 or Gazebo plugins (especially older Gazebo Classic plugins) still use `typeof` instead of `__typeof__`.
- Apple Clang is stricter than GNU Clang and won’t recognize `typeof`.

- This usually happens after switching compilers or updating C++ versions, because headers are included differently.

---

## ✅ How to fix

### 1️⃣ Check which file triggers it

From your error message, it should show the file path, e.g.:

```
some_header.hpp:123: error: use of undeclared identifier 'typeof'
```

- Note the file — it's often a Gazebo plugin or PX4 module.

---

### 2️⃣ Modify the source (temporary fix)

Edit the file and replace `typeof` with `__typeof__`:

```
// Original typeof(var) x; // Fix __typeof__(var) x;
```

This is safe because `__typeof__` is the Clang equivalent.

---

### 3️⃣ Alternative: Force GNU extensions

You can try telling Clang to allow GNU extensions:

```
export CXXFLAGS="-std=c++17 -stdlib=libc++
-Wno-error=deprecated-declarations -Wno-double-promotion -fgnu89-inline"
```

Then rebuild:

```
cd ~/px4_ws/PX4-Autopilot/build/px4_sitl_default ninja
```

- The `-fgnu89-inline` or `-fgnu` flags may allow older code using `typeof` to compile.

---

### 4️⃣ Clean build

If you've tried building multiple times with different flags:

```
rm -rf ~/px4_ws/PX4-Autopilot/build/px4_sitl_default mkdir -p
~/px4_ws/PX4-Autopilot/build/px4_sitl_default cd
~/px4_ws/PX4-Autopilot/build/px4_sitl_default cmake ../.. -GNinja
-DCMAKE_BUILD_TYPE=RelWithDebInfo -DCMAKE_CXX_STANDARD=17 ninja
```

Ensures there are no cached compiler flags causing conflicts.

---

⚠️ Note: PX4 SITL on macOS with Gazebo is notoriously sensitive to Homebrew versions, Clang versions, and Gazebo Classic vs. Ignition Gazebo. Many developers have to patch a few `typeof` occurrences in the source when using Apple Clang.

---

I can write a one-liner patch that replaces all `typeof` → `__typeof__` in PX4 + Gazebo headers so you won't have to hunt them manually.

Do you want me to do that?

**You said:**
Yes, that would be great

**ChatGPT said:**
Perfect! We can use a `sed` command to automatically patch all the files that use `typeof` in your PX4 and Gazebo source directories. Here's a safe way to do it:

---

### 1️⃣ Backup first (just in case)

```
cd ~/px4_ws/PX4-Autopilot cp -r src src_backup
```

---

### 2️⃣ Replace `typeof` with `__typeof__`

```
find . -type f \( -name "*.h" -o -name "*.hpp" -o -name "*.cpp" \) -exec
sed -i '' 's/\btypeof\b/__typeof__/g' {} +
```

Explanation:

- `find . -type f \( -name "*.h" -o -name "*.hpp" -o -name "*.cpp" \)` → looks for all header and C++ source files.
- `sed -i '' 's/\btypeof\b/__typeof__/g'` → replaces the word `typeof` with `__typeof__` in-place on macOS.
- `\b` ensures it only matches the word `typeof` and not part of another word.

---

## 3️⃣ Clean and rebuild

```
rm -rf ~/px4_ws/PX4-Autopilot/build/px4_sitl_default mkdir -p
~/px4_ws/PX4-Autopilot/build/px4_sitl_default cd
~/px4_ws/PX4-Autopilot/build/px4_sitl_default cmake ../.. -GNinja
-DCMAKE_BUILD_TYPE=RelWithDebInfo -DCMAKE_CXX_STANDARD=17 ninja
```

---

This should remove the `typeof` errors and allow Apple Clang to compile.

I can also give you a more complete patch command that fixes `typeof` only in Gazebo and PX4 plugins that typically trigger this to avoid unnecessary replacements.

Do you want me to do that more targeted patch?

**You said:**
This error displays: sed: RE error: illegal byte sequence

**ChatGPT said:**
Ah! That `sed: illegal byte sequence` error on macOS usually happens when `sed` encounters a file with non-UTF-8 characters. macOS `sed` is strict about encoding. We can fix this in a couple of ways. The cleanest approach is to force the locale to UTF-8 while running `sed`.

Here's a safer, macOS-compatible method:

---

## 1️⃣ Set the locale to UTF-8 temporarily

```
export LC_ALL=C.UTF-8
```

---

## 2️⃣ Use `LC_CTYPE=C` inline with `find` and `sed` to fix the replacement

```
find . -type f \( -name "*.h" -o -name "*.hpp" -o -name "*.cpp" \) -exec sh -c 'LC_CTYPE=C sed -i "" "s/\btypeof\b/__typeof__/g" "$0"' {} \;
```

Explanation:

- `LC_CTYPE=C` tells `sed` to ignore encoding issues.
- `-i ""` is required for macOS in-place editing.
- `"$0"` passes each file from `find` to `sed`.

---

## 3️⃣ Verify

```
grep -r "typeof" .
```

This should show no results if all `typeof` instances were replaced.

---

⚠️ Tip: If you still get errors, it might be safer to target only the specific directories that trigger `typeof` errors, e.g., PX4 modules or Gazebo plugins, instead of the entire source tree.

I can write that targeted command next, which avoids touching unrelated files and prevents encoding issues. Do you want me to do that?

**You said:**
The error repeats after running Step 2

**ChatGPT said:**
Ah, that explains a lot — the `typeof` issue keeps coming back because it's inside external dependencies (Gazebo, PX4 modules) that are re-downloaded or re-generated during build. That's why patching your local files alone isn't enough. We need a persistent fix that applies before each build. Here's the strategy:

---

## 1️⃣ Target the external sources that actually use `typeof`

For PX4 + Gazebo builds on macOS, the common culprits are:

- `src/modules/simulation/gz_plugins/` (PX4 Gazebo plugins)
- `Tools/sitl_gazebo/` (PX4 SITL Gazebo interface)
- Certain Gazebo source files in `gz-*` packages if you built from source

---

## 2️⃣ Apply the patch before building

You can create a small script to run each time before `make px4_sitl_default`:

```
#!/bin/bash # fix-typeof.sh set -e # PX4 Gazebo plugins find
~/px4_ws/PX4-Autopilot/src/modules/simulation/gz_plugins/ \ -type f \(
-name "*.h" -o -name "*.hpp" -o -name "*.cpp" \) \ -exec sh -c 'LC_CTYPE=C
sed -i "" "s/\btypeof\b/__typeof__/g" "$0"' {} \; # SITL Gazebo interface
find ~/px4_ws/PX4-Autopilot/Tools/sitl_gazebo/ \ -type f \( -name "*.h" -o
-name "*.hpp" -o -name "*.cpp" \) \ -exec sh -c 'LC_CTYPE=C sed -i ""
"s/\btypeof\b/__typeof__/g" "$0"' {} \; echo "typeof → __typeof__ patch
applied"
```

Make it executable:

```
chmod +x fix-typeof.sh
```

Then run this script before each build:

```
./fix-typeof.sh make px4_sitl_default
```

---

## ✅ Why this works

- The script only patches files that are actually compiled during SITL builds.
- Running it before `make` ensures regenerated or downloaded files (like Gazebo plugins) are patched each time.
- Using `LC_CTYPE=C` avoids the `illegal byte sequence` error on macOS.

---

If you want, I can give you a full step-by-step workflow for macOS that ensures PX4 SITL with Gazebo builds without hitting typeof, C++17, or other recurring errors. This avoids repeating manual patches.

Do you want me to do that?

**You said:**
Yes, that would be great

**ChatGPT said:**
Perfect! Let's outline a macOS-ready PX4 SITL + Gazebo build workflow that avoids the recurring `typeof`, C++17, Protobuf, and other common errors. I'll break it down into clear, sequential steps.

---

## Step 0 — Prerequisites

Make sure you have:

1. Homebrew installed (you already have it).
2. Xcode Command Line Tools:

```
xcode-select --install
```

3. Python 3 (Homebrew):

```
brew install python
```

---

## Step 1 — Install PX4 dependencies via Homebrew

```
brew tap osrf/simulation brew install cmake pkg-config qt5 protobuf eigen pcre tinyxml libyaml gstreamer gstreamer gst-plugins-base gst-plugins-good wget wget ninja brew install bullet ogre gz-sim8 gz-rendering8 gz-sensors8 gz-physics7
```

⚠️ Notes:

- Use `gz-sim8` for Ignition Gazebo 8 (recommended for PX4).

- Install GStreamer only if you plan to use camera plugins (`gst-plugins-base` + `gst-plugins-good`).

---

## Step 2 — Clone PX4 Autopilot

```
mkdir -p ~/px4_ws cd ~/px4_ws git clone
https://github.com/PX4/PX4-Autopilot.git --recursive cd PX4-Autopilot
```

---

## Step 3 — Ensure correct C++ version

Macs come with clang. PX4 now requires C++17:

```
export CXX=clang++ export CC=clang export CXXFLAGS="-std=c++17" export
CFLAGS="-std=c17"
```

Check:

```
clang++ --version
```

PX4 requires Xcode 14+ on recent macOS versions for full C++17 support.

---

## Step 4 — Patch `typeof` in PX4 Gazebo plugins

Create a script `fix-typeof.sh`:

```
#!/bin/bash set -e # Patch PX4 Gazebo plugin files find
~/px4_ws/PX4-Autopilot/src/modules/simulation/gz_plugins/ \ -type f \(
-name "*.h" -o -name "*.hpp" -o -name "*.cpp" \) \ -exec sh -c 'LC_CTYPE=C
sed -i "" "s/\btypeof\b/__typeof__/g" "$0"' {} \; # Patch SITL Gazebo
interface find ~/px4_ws/PX4-Autopilot/Tools/sitl_gazebo/ \ -type f \(
-name "*.h" -o -name "*.hpp" -o -name "*.cpp" \) \ -exec sh -c 'LC_CTYPE=C
sed -i "" "s/\btypeof\b/__typeof__/g" "$0"' {} \; echo "typeof →
__typeof__ patch applied"
```

Make it executable:

```
chmod +x fix-typeof.sh
```

Important: Run this before each PX4 SITL build, especially after `git pull`.

---

## Step 5 — Build PX4 SITL with Gazebo

1. Clear previous builds (if any):

```
rm -rf build/px4_sitl_default
```

2. Apply the patch:

```
./fix-typeof.sh
```

3. Build PX4 SITL:

```
make px4_sitl_default gazebo
```

- This uses your installed Gazebo (`gz-sim8`).
- If you installed GStreamer, camera plugins will build; otherwise, you'll see warnings — that's safe if you don't need camera simulation.

---

## Step 6 — Run SITL

Example with Iris quadrotor:

```
make px4_sitl_default gazebo_iris
```

---

## Step 7 — Optional: QGroundControl

Install QGroundControl (flight GUI):

```
brew install --cask qgroundcontrol
```

Launch it and connect to PX4 SITL for telemetry & control.

---

## ✅ Why this workflow avoids recurring errors

- `typeof` → `__typeof__` patch handles macOS Clang incompatibility.
- C++17 enforced with `CXXFLAGS`.
- Protobuf, Gazebo, GStreamer installed via Homebrew to ensure compatible versions.

- Build folder cleared before rebuilds avoids `ninja: no build.ninja` errors.

---

I can also make you a single script that installs all dependencies, patches files, and builds PX4 SITL — so you can do everything with one command on macOS.