**DIY Marine Compass System Build Guide**

**ESP32/Pi Zero 2W + CMPS12 Replacement for Robertson AP1000 Autopilot  
Paul Collister, SV Sister Midnight, paul@collisters.com**

**Table of Contents**

1. [Project Overview](#project-overview)
2. [Hardware Selection](#hardware-selection)
3. [System Architecture](#system-architecture)
4. [Hardware Connections](#hardware-connections)
5. [Software Implementation](#software-implementation)
6. [Sin/Cos Output Methods](#sincos-output-methods)
7. [Development Environment](#development-environment)
8. [Web Interface](#web-interface)
9. [Installation Guide](#installation-guide)
10. [Troubleshooting](#troubleshooting)

**Project Overview**

**Background**

Replacing a faulty Robertson AP1000 autopilot compass (RFC35NS fluxgate) with a modern DIY solution using:

* **CMPS12** compass module (tilt-compensated, marine-grade)
* **Pi Zero 2W** microcontroller
* **Sin/Cos analog output** (0-5V) for Robertson autopilot compatibility
* **WiFi connectivity** for remote monitoring and NMEA data
* **Web interface** for calibration and diagnostics

**Original System Requirements**

* **Sin/Cos outputs**: Analog voltages representing heading
* **2.5V reference**: Center point for sin/cos signals
* **Signal behavior**:
  + 0° (N): Sin=2.5V, Cos=5.0V
  + 90° (E): Sin=5.0V, Cos=2.5V
  + 180° (S): Sin=2.5V, Cos=0V
  + 270° (W): Sin=0V, Cos=2.5V

**Hardware Selection**

**Recommended Platform: Pi Zero 2W**

**Why Pi Zero 2W:**

* ✅ **Full Linux** - native Python support
* ✅ **Built-in WiFi** - no additional hardware needed
* ✅ **GPIO for I2C** - direct CMPS12 connection
* ✅ **Low power** - suitable for marine environment
* ✅ **Compact** - easy to mount in waterproof enclosure
* ✅ **PWM outputs** for sin/cos generation

**Alternative considered:**

* ESP32: Good but more complex for WiFi + web interface in Python
* Pico WH: Limited to MicroPython
* Pi 5: Overkill, higher power consumption

**Compass Module: CMPS12**

**CMPS12 Advantages:**

* ✅ **Tilt compensation** - up to ±80° pitch/roll
* ✅ **Built-in calibration** - handles hard/soft iron compensation
* ✅ **Multiple interfaces** - I2C, Serial, PWM
* ✅ **Temperature compensation**
* ✅ **Good resolution** - 0.1° accuracy
* ✅ **Proven marine track record**
* ✅ **3D calibration capability**

**System Architecture**

┌─────────────────┐ I2C ┌──────────────┐

│ CMPS12 │◄─────────►│ Pi Zero 2W │

│ Compass │ │ │

└─────────────────┘ │ Python │

│ Software │

┌─────────────────┐ PWM+ │ │

│ Op-Amp │ Filter │ │

│ Sin/Cos Circuit │◄─────────►│ │

└─────────────────┘ └──────────────┘

│ │

│ Analog 0-5V │ WiFi

▼ │

┌─────────────────┐ │

│ Robertson │ │

│ AP1000 │ ▼

│ Autopilot │ ┌──────────────┐

└─────────────────┘ │ Your Mac │

│ Web App │

│ NMEA Client │

└──────────────┘

**Key Components**

1. **CMPS12 Compass Module** (~$50-70)
2. **Pi Zero 2W** (~$15)
3. **Op-amp circuit components** (~$5)
4. **Waterproof enclosure**
5. **Power supply** (12V → 5V/3.3V)

**Hardware Connections**

**Pi Zero 2W GPIO Pinout**

# I2C for CMPS12

GPIO 2 (SDA) → CMPS12 SDA

GPIO 3 (SCL) → CMPS12 SCL

3.3V → CMPS12 VCC

GND → CMPS12 GND

# PWM for Sin/Cos generation

GPIO 18 (PWM0) → Sin filter circuit → Robertson Sin input

GPIO 19 (PWM1) → Cos filter circuit → Robertson Cos input

5V → Op-amp supply

GND → Common ground

**CMPS12 Connection Details**

CMPS12 Pin Pi Zero 2W Pin Function

VCC 3.3V Power

GND GND Ground

SDA GPIO 2 I2C Data

SCL GPIO 3 I2C Clock

**Sin/Cos Output Methods**

**Method 1: PWM + Op-Amp (Recommended)**

**Why this method:**

* ✅ **No additional DAC chips needed**
* ✅ **Simpler software** (just PWM duty cycle)
* ✅ **Cheaper** (~$5 vs $15)
* ✅ **Good enough accuracy** for autopilot use
* ✅ **Easy to troubleshoot**

**PWM Filter Circuit (per channel)**

Pi PWM → 1kΩ resistor → 10µF capacitor → Op-amp → 0-5V output

└─ to ground

Op-amp configuration (non-inverting amplifier):

- Gain = 1.67 (to scale 3.3V PWM to 5V output)

- R1 = 10kΩ (feedback resistor)

- R2 = 15kΩ (gain setting resistor)

**Component List for PWM Method**

* 1x **LM358 dual op-amp** (~$1)
* 2x **1kΩ resistors** (PWM filter)
* 2x **10µF capacitors** (PWM filter)
* 2x **10kΩ resistors** (op-amp feedback)
* 2x **15kΩ resistors** (op-amp gain setting)
* 1x **Small breadboard** or stripboard
* **Jumper wires**
* **Total cost: ~$5**

**PWM Code Implementation**

import RPi.GPIO as GPIO

import math

import time

# Setup PWM channels

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.OUT) # Sin PWM

GPIO.setup(19, GPIO.OUT) # Cos PWM

pwm\_sin = GPIO.PWM(18, 1000) # 1kHz PWM

pwm\_cos = GPIO.PWM(19, 1000) # 1kHz PWM

pwm\_sin.start(0)

pwm\_cos.start(0)

def output\_sin\_cos(heading\_degrees):

"""Convert heading to sin/cos PWM signals"""

heading\_rad = math.radians(heading\_degrees)

# Calculate sin/cos values (-1 to +1)

sin\_val = math.sin(heading\_rad)

cos\_val = math.cos(heading\_rad)

# Convert to PWM duty cycle (10-90% range)

# 50% = 2.5V center, ±40% = ±2V swing

sin\_duty = 50 + 40 \* sin\_val

cos\_duty = 50 + 40 \* cos\_val

pwm\_sin.ChangeDutyCycle(sin\_duty)

pwm\_cos.ChangeDutyCycle(cos\_duty)

**Method 2: MCP4922 Dual DAC (Alternative)**

**If you prefer a dedicated DAC:**

* **MCP4922**: Dual 12-bit DAC with SPI interface
* **Cost**: ~$10-15 for breakout board
* **Advantage**: True analog output
* **Disadvantage**: More complex, more expensive

**Method 3: Dual MCP4725 (Not Recommended)**

**Why not recommended:**

* Need 2 separate modules
* More expensive than other methods
* Uses additional I2C addresses
* May need voltage scaling for 5V output

**Software Implementation**

**File Structure**

/home/pi/compass\_project/

├── compass\_server.py # Main application

├── web\_interface/

│ ├── app.py # Flask web server

│ ├── templates/

│ │ └── compass.html # Web dashboard

│ └── static/

│ ├── compass.js # Real-time updates

│ └── style.css

├── nmea\_server.py # NMEA TCP server

├── calibration.py # Compass calibration

├── requirements.txt # Python dependencies

└── config.json # Configuration settings

**Main Application Code**

# compass\_server.py

import asyncio

import websockets

import json

import smbus2

import math

import threading

from flask import Flask, render\_template

import socket

import RPi.GPIO as GPIO

import time

class CompassSystem:

def \_\_init\_\_(self):

self.i2c = smbus2.SMBus(1) # CMPS12 I2C connection

self.cmps12\_address = 0x60 # Default CMPS12 address

self.heading = 0

self.calibrated = False

self.setup\_pwm()

def setup\_pwm(self):

"""Initialize PWM outputs for sin/cos"""

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.OUT) # Sin PWM

GPIO.setup(19, GPIO.OUT) # Cos PWM

self.pwm\_sin = GPIO.PWM(18, 1000) # 1kHz PWM

self.pwm\_cos = GPIO.PWM(19, 1000)

self.pwm\_sin.start(50) # Start at center (2.5V)

self.pwm\_cos.start(90) # Start at max (5V) for 0° heading

def read\_cmps12\_heading(self):

"""Read heading from CMPS12 via I2C"""

try:

# Read 16-bit heading register (0x02-0x03)

self.i2c.write\_byte(self.cmps12\_address, 0x02)

time.sleep(0.001) # Small delay

high\_byte = self.i2c.read\_byte(self.cmps12\_address)

low\_byte = self.i2c.read\_byte(self.cmps12\_address)

# Combine bytes and convert to degrees

heading\_raw = (high\_byte << 8) | low\_byte

self.heading = heading\_raw / 10.0 # CMPS12 returns degrees \* 10

return self.heading

except Exception as e:

print(f"Error reading CMPS12: {e}")

return None

def output\_sin\_cos(self, heading\_degrees):

"""Generate sin/cos PWM outputs for Robertson autopilot"""

heading\_rad = math.radians(heading\_degrees)

sin\_val = math.sin(heading\_rad)

cos\_val = math.cos(heading\_rad)

# Convert to PWM duty cycle (10-90% range)

sin\_duty = 50 + 40 \* sin\_val

cos\_duty = 50 + 40 \* cos\_val

# Clamp values to safe range

sin\_duty = max(10, min(90, sin\_duty))

cos\_duty = max(10, min(90, cos\_duty))

self.pwm\_sin.ChangeDutyCycle(sin\_duty)

self.pwm\_cos.ChangeDutyCycle(cos\_duty)

def generate\_nmea\_hdt(self):

"""Generate NMEA HDT sentence"""

# Calculate checksum

sentence = f"IIHDT,{self.heading:.1f},T"

checksum = 0

for char in sentence:

checksum ^= ord(char)

return f"${sentence}\*{checksum:02X}\r\n"

async def compass\_loop(self):

"""Main compass reading and output loop"""

while True:

heading = self.read\_cmps12\_heading()

if heading is not None:

self.output\_sin\_cos(heading)

await asyncio.sleep(0.05) # 20Hz update rate

async def websocket\_handler(self, websocket, path):

"""Handle WebSocket connections for real-time data"""

try:

while True:

data = {

'heading': self.heading,

'nmea': self.generate\_nmea\_hdt().strip(),

'calibrated': self.calibrated,

'timestamp': time.time()

}

await websocket.send(json.dumps(data))

await asyncio.sleep(0.2) # 5Hz web update

except websockets.exceptions.ConnectionClosed:

pass

def start\_nmea\_server(self):

"""TCP server for NMEA data (port 10110)"""

def nmea\_server():

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

server\_socket.bind(('', 10110))

server\_socket.listen(5)

print("NMEA server listening on port 10110")

while True:

try:

client\_socket, address = server\_socket.accept()

print(f"NMEA client connected: {address}")

while True:

nmea\_data = self.generate\_nmea\_hdt()

client\_socket.send(nmea\_data.encode())

time.sleep(1) # 1Hz NMEA rate

except Exception as e:

print(f"NMEA server error: {e}")

if 'client\_socket' in locals():

client\_socket.close()

thread = threading.Thread(target=nmea\_server, daemon=True)

thread.start()

# Main execution

if \_\_name\_\_ == "\_\_main\_\_":

compass = CompassSystem()

# Start NMEA server

compass.start\_nmea\_server()

# Start WebSocket server

start\_server = websockets.serve(

compass.websocket\_handler,

"0.0.0.0",

8765

)

# Start compass loop

loop = asyncio.get\_event\_loop()

loop.run\_until\_complete(asyncio.gather(

start\_server,

compass.compass\_loop()

))

loop.run\_forever()

**Python Dependencies**

# requirements.txt

flask==2.3.3

websockets==11.0.3

smbus2==0.4.2

RPi.GPIO==0.7.1

asyncio==3.4.3

**Installation Commands**

# Update Pi Zero 2W

sudo apt update && sudo apt upgrade -y

# Install Python dependencies

pip install -r requirements.txt

# Enable I2C interface

sudo raspi-config

# Interface Options → I2C → Enable

# Install project

git clone <your-repo> /home/pi/compass\_project

cd /home/pi/compass\_project

chmod +x compass\_server.py

**Development Environment**

**Remote Development from Mac**

**SSH Setup**

# Connect to Pi Zero 2W

ssh pi@zero2w.local

# Enable SSH key authentication

ssh-copy-id pi@zero2w.local

# Edit code remotely with VS Code

code --remote ssh-remote+zero2w.local /home/pi/compass\_project

**File Transfer**

# Copy files to Pi

scp compass\_server.py pi@zero2w.local:/home/pi/compass\_project/

# Sync entire directory

rsync -avz ./compass\_project/ pi@zero2w.local:/home/pi/compass\_project/

**Testing and Debugging**

# Test I2C connection

sudo i2cdetect -y 1

# Should show CMPS12 at address 0x60

# Test CMPS12 reading

python3 -c "

import smbus2

bus = smbus2.SMBus(1)

bus.write\_byte(0x60, 0x02)

import time; time.sleep(0.001)

h = bus.read\_byte(0x60) << 8 | bus.read\_byte(0x60)

print(f'Heading: {h/10.0}°')

"

# Monitor PWM output with multimeter

# GPIO 18 and 19 should show varying voltage 1-4V

**Web Interface**

**Flask Web Server**

# web\_interface/app.py

from flask import Flask, render\_template, jsonify

import json

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('compass.html')

@app.route('/api/compass')

def api\_compass():

# Return current compass data

return jsonify({

'heading': compass.heading,

'calibrated': compass.calibrated

})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000, debug=True)

**HTML Template**

<!-- web\_interface/templates/compass.html -->

<!DOCTYPE html>

<html>

<head>

<title>Marine Compass System</title>

<style>

.compass-container {

width: 300px;

height: 300px;

border: 2px solid #333;

border-radius: 50%;

position: relative;

margin: 50px auto;

background: radial-gradient(circle, #87CEEB, #4682B4);

}

.compass-needle {

width: 4px;

height: 120px;

background: red;

position: absolute;

top: 30px;

left: 50%;

transform-origin: bottom center;

transition: transform 0.3s ease;

}

.heading-display {

text-align: center;

font-size: 24px;

font-weight: bold;

margin: 20px;

}

.nmea-output {

background: #000;

color: #0f0;

font-family: monospace;

padding: 10px;

margin: 20px;

height: 100px;

overflow-y: scroll;

}

</style>

</head>

<body>

<h1>Marine Compass System</h1>

<div class="compass-container">

<div class="compass-needle" id="needle"></div>

<!-- Add compass rose markings -->

<div style="position: absolute; top: 5px; left: 50%; transform: translateX(-50%);">N</div>

<div style="position: absolute; top: 50%; right: 5px; transform: translateY(-50%);">E</div>

<div style="position: absolute; bottom: 5px; left: 50%; transform: translateX(-50%);">S</div>

<div style="position: absolute; top: 50%; left: 5px; transform: translateY(-50%);">W</div>

</div>

<div class="heading-display" id="heading">---°</div>

<div class="status" id="status">Connecting...</div>

<div class="nmea-output" id="nmea"></div>

<script>

const ws = new WebSocket('ws://' + window.location.hostname + ':8765');

const needle = document.getElementById('needle');

const headingDisplay = document.getElementById('heading');

const statusDisplay = document.getElementById('status');

const nmeaOutput = document.getElementById('nmea');

ws.onopen = function() {

statusDisplay.textContent = 'Connected';

statusDisplay.style.color = 'green';

};

ws.onmessage = function(event) {

const data = JSON.parse(event.data);

// Update compass needle

needle.style.transform = `rotate(${data.heading}deg)`;

// Update heading display

headingDisplay.textContent = `${data.heading.toFixed(1)}°`;

// Update NMEA output

nmeaOutput.innerHTML += data.nmea + '<br>';

nmeaOutput.scrollTop = nmeaOutput.scrollHeight;

// Limit NMEA output lines

const lines = nmeaOutput.innerHTML.split('<br>');

if (lines.length > 50) {

nmeaOutput.innerHTML = lines.slice(-50).join('<br>');

}

};

ws.onclose = function() {

statusDisplay.textContent = 'Disconnected';

statusDisplay.style.color = 'red';

};

ws.onerror = function() {

statusDisplay.textContent = 'Connection Error';

statusDisplay.style.color = 'red';

};

</script>

</body>

</html>

**Installation Guide**

**Step 1: Pi Zero 2W Setup**

# Flash Raspberry Pi OS Lite to SD card

# Enable SSH and WiFi in config

# First boot setup

sudo raspi-config

# → Interface Options → I2C → Enable

# → Interface Options → SSH → Enable

# → Finish and reboot

# Update system

sudo apt update && sudo apt upgrade -y

# Install required packages

sudo apt install python3-pip python3-venv git i2c-tools -y

**Step 2: Hardware Assembly**

1. **Mount CMPS12** in waterproof enclosure
2. **Connect I2C wires** (SDA, SCL, VCC, GND)
3. **Build PWM filter circuit** on breadboard/PCB
4. **Connect power supply** (12V boat → 5V/3.3V regulation)
5. **Test all connections** with multimeter

**Step 3: Software Installation**

# Create project directory

mkdir /home/pi/compass\_project

cd /home/pi/compass\_project

# Create virtual environment

python3 -m venv venv

source venv/bin/activate

# Install dependencies

pip install flask websockets smbus2 RPi.GPIO

# Copy project files

# (Copy all the Python files from above)

# Test CMPS12 connection

sudo i2cdetect -y 1

# Should show device at 0x60

**Step 4: Calibration**

# Run calibration routine

python3 calibration.py

# Follow prompts for:

# 1. Horizontal rotation (360° on level surface)

# 2. 3D calibration (figure-8 in all orientations)

# 3. Save calibration to CMPS12 EEPROM

**Step 5: Service Setup**

# Create systemd service

sudo nano /etc/systemd/system/compass.service

# Service file content:

[Unit]

Description=Marine Compass System

After=network.target

[Service]

Type=simple

User=pi

WorkingDirectory=/home/pi/compass\_project

ExecStart=/home/pi/compass\_project/venv/bin/python compass\_server.py

Restart=always

RestartSec=3

[Install]

WantedBy=multi-user.target

# Enable service

sudo systemctl enable compass.service

sudo systemctl start compass.service

# Check status

sudo systemctl status compass.service

**Troubleshooting**

**Common Issues**

**1. CMPS12 Not Detected**

# Check I2C connection

sudo i2cdetect -y 1

# Verify wiring:

# Pi Pin 3 (GPIO 2) → CMPS12 SDA

# Pi Pin 5 (GPIO 3) → CMPS12 SCL

# Pi Pin 1 (3.3V) → CMPS12 VCC

# Pi Pin 6 (GND) → CMPS12 GND

# Check CMPS12 power LED

# Should be solid when powered correctly

**2. Fixed 225° Reading**

* **Compass failure**: CMPS12 not responding
* **Calibration needed**: Run 3D calibration routine
* **Magnetic interference**: Move away from metal objects
* **Power issues**: Check 3.3V supply voltage

**3. Sin/Cos Output Problems**

# Test PWM signals with multimeter

# GPIO 18: Should vary 1-4V as heading changes

# GPIO 19: Should vary 1-4V as heading changes

# Check op-amp circuit:

# Input: 0-3.3V PWM (filtered)

# Output: 0-5V analog

# Verify Robertson autopilot input impedance

# High impedance inputs work better

**4. Web Interface Not Loading**

# Check Flask server

curl http://localhost:5000

# Check WebSocket server

netstat -an | grep 8765

# Check firewall

sudo ufw status

# Check WiFi connection

iwconfig wlan0

**5. NMEA Server Issues**

# Test NMEA TCP server

telnet localhost 10110

# Should receive HDT sentences:

# $IIHDT,123.4,T\*XX

# Check port binding

netstat -an | grep 10110

**Marine Environment Considerations**

**Waterproofing**

* **IP67 rated enclosure** minimum
* **Cable glands** for all wire entries
* **Desiccant packs** inside enclosure
* **Conformal coating** on PCB

**Power Supply**

* **12V to 5V converter** for Pi Zero 2W
* **Clean power** - add filtering capacitors
* **Overcurrent protection** - fuses on all inputs
* **Reverse polarity protection** - diode on 12V input

**Mounting**

* **Rigid mounting** - prevent vibration
* **Low in boat** - reduce motion amplification
* **Away from metal** - engine, fuel tanks, batteries
* **Compass rose alignment** - note magnetic declination

**EMI Protection**

* **Shielded cables** for long runs
* **Ferrite cores** on power/signal cables
* **Twisted pair wiring** for differential signals
* **Star ground** - single point grounding

**Performance Optimization**

**Marine Motion Filtering**

def filter\_heading(new\_heading, prev\_heading, alpha=0.8):

"""Exponential filter for heading stability"""

# Handle 359° → 0° wraparound

diff = new\_heading - prev\_heading

if diff > 180:

diff -= 360

elif diff < -180:

diff += 360

filtered = prev\_heading + alpha \* diff

return filtered % 360

def reject\_outliers(heading, heading\_history, max\_rate=10):

"""Reject impossible heading changes"""

if len(heading\_history) > 0:

rate = abs(heading - heading\_history[-1])

if rate > max\_rate: # degrees per update

return heading\_history[-1] # Use previous value

return heading

**Calibration Best Practices**

1. **Away from metal** - dock, engine off
2. **All systems on** - electronics, engine, lights
3. **Multiple orientations** - level, tilted, inverted
4. **Save to EEPROM** - permanent storage in CMPS12
5. **Verify accuracy** - compare to known heading

**Conclusion**

This DIY compass system provides a modern, cost-effective replacement for marine autopilot compass sensors. The combination of CMPS12 accuracy, Pi Zero 2W flexibility, and simple PWM output generation creates a robust solution suitable for demanding marine environments.

**Key benefits:**

* **Cost effective**: ~$100 vs $500+ commercial units
* **Highly accurate**: 0.1° resolution with tilt compensation
* **Remote monitoring**: WiFi web interface and NMEA output
* **Maintainable**: Open source Python code
* **Upgradeable**: Easy firmware updates via SSH

**Total project cost**: ~$75-100 **Build time**: 1-2 weekends **Skill level**: Intermediate (basic electronics + Python)

The system is designed for reliability in harsh marine conditions while providing modern connectivity features not available in traditional marine compass systems.

*Last updated: January 2025* *Hardware: Pi Zero 2W + CMPS12* *Software: Python 3.9+ on Raspberry Pi OS*