XE125 Pulsed Coherent Radar — Build Protocol

Sweeping Presence Detector with Live Web Radar Display

Author: J. Street
Date: June 2026
Difficulty: Intermediate
Time: 3–5 hours

---

1. Overview

This project builds a standalone sweeping radar unit using the Acconeer XE125 pulsed coherent radar sensor, an ESP32 microcontroller, and a 180° servo motor. The ESP32 hosts a web server serving a live radar display — accessible from any browser on the same WiFi network. No laptop or external computer is required during operation.

What it does:

- XE125 detects presence and distance of objects (0.2m–3.0m)
- Servo sweeps the radar 10°–170° back and forth
- ESP32 packages distance + angle into JSON and streams via WebSocket
- Browser displays a live polar radar map with a sweeping beam and red detection dots

Key concept — data flow:

```
XE125 (I2C) → ESP32 → WebSocket → Browser radar display
     ↑
  Servo angle
```

---

2. Hardware

Parts List

Component	Details
Acconeer XE125	Pulsed coherent radar evaluation board
ESP32 DevKit	Any standard 38-pin ESP32
Servo	SM-S2309S 180° servo
USB-C cable	Data capable (not charge-only)
Power	USB powerbank or 5V supply

Dupont wires	Female-female and male-female

Wiring

XE125 → ESP32 (I2C):

XE125 Pin (J2)	ESP32 Pin
VCC (3.3V)	3.3V
GND	GND
SDA	GPIO 21
SCL	GPIO 22

Servo SM-S2309S → ESP32:

Servo Wire	ESP32 Pin
Red	VIN (5V)
Black	GND
White (signal)	GPIO 13

> **Note:** Use the J2 header on the XE125 for I2C connections. The XE125 and servo share GND with the ESP32.

---

3. Software Prerequisites

Mac Setup

Install Python and pip if not already present. Then create a virtual environment:

```
python3 -m venv acconeer-env
source acconeer-env/bin/activate
pip install "acconeer-exptool[app]"
```

> **Zsh note:** Always quote the package name with brackets — zsh interprets `[app]` as a glob pattern otherwise.

Find the Serial Port

Plug the XE125 into your Mac via USB-C data cable and run:

```
ioreg -p IOUSB -l -w 0 | grep -E "idVendor|idProduct|USB Product"
```

Look for `"USB Product Name" = "Acconeer XE125"`. Then list serial ports:

```
ls /dev/cu.*
```

The XE125 appears as two ports via the CP2105 dual UART chip:

```
/dev/cu.usbserial-11200
/dev/cu.usbserial-11201
```

Use the first port (`11200`) for all subsequent commands.

> **Mac driver note:** Modern macOS includes CP210x support natively. No manual driver installation is needed if using a data-capable USB-C cable.

---

4. XE125 Firmware — Two Paths

The XE125 ships without application firmware. You must flash one of two firmware options depending on your use case.

Path A — Exploration Server (Python development)

Use this firmware to develop and test with Python on your computer.

Download: Log into developer.acconeer.com (https://developer.acconeer.com) and download the XM125 SDK. Extract and locate:

```
xm125/out/acc_exploration_server_a121.bin
```

Flash:

```
# Put XE125 in bootloader mode first:
# Hold BOOT → press and release RST → release BOOT

python -m acconeer.exptool.flash flash XM125 \
  --port /dev/cu.usbserial-11200 \
  --image /path/to/acc_exploration_server_a121.bin
```

Press RST once after flashing to boot into normal mode.

Test connection:

```
from acconeer.exptool import a121
client = a121.Client.open(serial_port="/dev/cu.usbserial-11200")
print("Connected!")
client.close()
```

Launch GUI:

```
python -m acconeer.exptool.app
```

Python Exploration (Optional)

Before building the ESP32 standalone system, you can explore raw sensor data directly in Python. This is useful for understanding what the sensor measures and tuning parameters. Requires Path A firmware (`acc_exploration_server_a121.bin`) flashed on the XE125.

Install dependencies

```
pip install "acconeer-exptool[algo]"
pip install numpy
```

Basic presence readings

```
import numpy as np
from acconeer.exptool import a121
from acconeer.exptool.a121.algo.presence import Detector, DetectorConfig

# adjust port to match your system
PORT = "/dev/cu.usbserial-11200"

start_m = 0.25
end_m = 3.0
threshold = 2.0  # raise to reduce noise, lower to increase sensitivity

client = a121.Client.open(serial_port=PORT)

detector = Detector(
    client=client,
    sensor_id=1,
    detector_config=DetectorConfig(start_m=start_m, end_m=end_m)
)
detector.start()

for i in range(50):
    result = detector.get_next()

    # intra = fast motion (walking, waving)
    # inter = slow motion (breathing, subtle movement)
    print(f"presence: {result.presence_detected}  "
          f"distance: {result.presence_distance:.2f}m  "
          f"intra max: {result.intra_depthwise_scores.max():.2f}  "
          f"inter max: {result.inter_depthwise_scores.max():.2f}")

    # find all range points above threshold and print their distances
    num_points = len(result.intra_depthwise_scores)
    distances_m = np.linspace(start_m, end_m, num_points)
    detected = distances_m[result.intra_depthwise_scores > threshold]
    if len(detected) > 0:
        print(f"  detected at: {np.round(detected * 100, 1)} cm")

detector.stop()
client.close()
```

What the values mean

Value	Description
`presence_detected`	Boolean — something detected yes/no
`presence_distance`	Distance to strongest detection in meters

`intra_depthwise_scores`	Per-point score for fast motion (47 points across range)
`inter_depthwise_scores`	Per-point score for slow motion / breathing
`threshold`	Minimum score to count as a real detection — tune for your environment

Tuning the threshold

Run the script in an empty room and note the `intra max` values — typically 1.1–1.4. Then wave your hand and note the values — typically 3.0+. Set threshold between those two values. A value of `2.0` works well in most indoor environments.

Path B — I2C Presence Detector (ESP32 standalone)

Use this firmware for standalone ESP32 deployment. The XE125 runs detection onboard and outputs results via I2C — no host computer needed.

Download binary from SparkFun GitHub:

```
https://github.com/sparkfun/SparkFun_Qwiic_Pulsed_Radar_Sensor_XM125/tree/main/Firmware/A121 SDK for XM125/xm125/out/
```

File: `i2c_presence_detector.bin`

Flash same way as Path A:

```
python -m acconeer.exptool.flash flash XM125 \
  --port /dev/cu.usbserial-11200 \
  --image /path/to/i2c_presence_detector.bin
```

> **Note:** You can switch between Path A and Path B at any time by reflashing.

---

5. Arduino/ESP32 Setup

Required Libraries

Install via Arduino IDE Library Manager:

- **SparkFun Qwiic XM125** by SparkFun Electronics
- **ESP32Servo** by Kevin Harrington
- **AsyncTCP** by dvarrel
- **ESPAsyncWebServer** by lacamera

> **ESP32Servo note:** Do not use the standard `<Servo.h>` — it is not compatible with ESP32 timer hardware. Use `<ESP32Servo.h>` instead, which has an identical API.

Upload Method

If normal Arduino IDE upload fails, export as binary and flash with esptool:

```
esptool.py --port /dev/cu.usbserial-11200 \
  --baud 115200 write_flash 0x0 \
  YourSketch.ino.merged.bin
```

---

6. Full Arduino Sketch

```
#include "SparkFun_Qwiic_XM125_Arduino_Library.h"
#include <Arduino.h>
#include <Wire.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <ESP32Servo.h>

// ---------------------------------------------------------
// WiFi credentials
// ---------------------------------------------------------
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// ---------------------------------------------------------
// Radar
// ---------------------------------------------------------
SparkFunXM125Presence radarSensor;
uint8_t i2cAddress = SFE_XM125_I2C_ADDRESS;
uint32_t distance = 0;
int32_t presValError = 0;

#define MY_XM125_RANGE_START 200   // mm
#define MY_XM125_RANGE_END   3000  // mm

// ---------------------------------------------------------
// Servo
// ---------------------------------------------------------
Servo radarServo;
#define SERVO_PIN 13
#define SERVO_MIN 10   // soft limit – avoids mechanical strain
```

```
#define SERVO_MAX 170  // soft limit — avoids mechanical strain
int servoAngle = 10;
int servoStep = 5;     // degrees per step

// ------------------------------------------------------
// Web server + WebSocket
// ------------------------------------------------------
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

// ------------------------------------------------------
// HTML radar interface (stored in flash memory)
// ------------------------------------------------------
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>XE125 Radar</title>
  <style>
    * { margin: 0; padding: 0; box-sizing: border-box; }
    body { background: #000; display: flex; flex-direction: column; align
-items: center; justify-content: center; min-height: 100vh; font-family:
monospace; }
    #dist-display { font-size: 24px; color: #f87171; margin: 8px 0 4px; m
in-height: 30px; }
    #status { font-size: 12px; color: #4ade80; margin-bottom: 8px; }
  </style>
</head>
<body>
<div id="dist-display">— cm</div>
<div id="status">connecting...</div>
<canvas id="c" width="500" height="500"></canvas>
<script>
const canvas = document.getElementById('c');
const ctx = canvas.getContext('2d');
const W=500, H=500, CX=250, CY=250, R=220, MAX=300;
let detectedList = [];

// currentAngle tracks servo position and drives beam direction
// starts at 90 so beam points up (12 o'clock) on load
let currentAngle = 90;

function draw() {
  ctx.clearRect(0,0,W,H);
  ctx.fillStyle='#000';
  ctx.fillRect(0,0,W,H);
```

```
  // range rings
  for(let i=1;i<=4;i++){
    const r=(R/4)*i;
    ctx.strokeStyle='#0f3a0f';
    ctx.lineWidth=0.5;
    ctx.setLineDash([4,4]);
    ctx.beginPath(); ctx.arc(CX,CY,r,0,Math.PI*2); ctx.stroke();
    ctx.setLineDash([]);
    ctx.fillStyle='#1a5a1a';
    ctx.font='11px monospace';
    ctx.textAlign='center';
    ctx.fillText(Math.round((MAX/4)*i)+'cm', CX+r+24, CY-4);
  }

  // outer ring
  ctx.strokeStyle='#0a2a0a';
  ctx.lineWidth=1;
  ctx.beginPath(); ctx.arc(CX,CY,R,0,Math.PI*2); ctx.stroke();

  // degree tick marks — every 5 degrees, major every 30
  for(let deg=0;deg<360;deg+=5){
    const rad=(deg-90)*Math.PI/180;
    const isMajor=deg%30===0;
    ctx.strokeStyle=isMajor?'#1a6a1a':'#0f3a0f';
    ctx.lineWidth=isMajor?1:0.5;
    ctx.beginPath();
    ctx.moveTo(CX+Math.cos(rad)*(isMajor?R-10:R-5), CY+Math.sin(rad)*(isM
ajor?R-10:R-5));
    ctx.lineTo(CX+Math.cos(rad)*R, CY+Math.sin(rad)*R);
    ctx.stroke();
    if(isMajor){
      ctx.fillStyle='#2a8a2a';
      ctx.font='10px monospace';
      ctx.textAlign='center';
      ctx.textBaseline='middle';
      ctx.fillText(deg+'°', CX+Math.cos(rad)*(R+16), CY+Math.sin(rad)*(R+
16));
    }
  }

  // radial grid lines every 30 degrees
  for(let deg=0;deg<360;deg+=30){
    const rad=(deg-90)*Math.PI/180;
    ctx.strokeStyle='#0a2a0a';
    ctx.lineWidth=0.5;
    ctx.setLineDash([2,6]);
    ctx.beginPath(); ctx.moveTo(CX,CY); ctx.lineTo(CX+Math.cos(rad)*R, CY
+Math.sin(rad)*R);
```

```
    ctx.stroke();
    ctx.setLineDash([]);
  }

  // beam angle driven by servo position
  // mapping: 0deg=left, 90deg=up (12 o'clock), 180deg=right
  const beam = (currentAngle - 90) * Math.PI / 180;

  // sweeping beam line
  ctx.strokeStyle='#00ff4433';
  ctx.lineWidth=2;
  ctx.beginPath(); ctx.moveTo(CX,CY); ctx.lineTo(CX+Math.cos(beam)*R, CY+
Math.sin(beam)*R);
  ctx.stroke();

  // detection dots — position uses beam angle so dots follow the servo
  for(const cm of detectedList){
    if(cm <= MAX){
      const dr=(cm/MAX)*R;
      const dx=CX+Math.cos(beam)*dr;
      const dy=CY+Math.sin(beam)*dr;
      ctx.fillStyle='#ff000044';
      ctx.beginPath(); ctx.arc(dx,dy,14,0,Math.PI*2); ctx.fill();
      ctx.fillStyle='#ff4444';
      ctx.beginPath(); ctx.arc(dx,dy,6,0,Math.PI*2); ctx.fill();
      ctx.fillStyle='#ff8888';
      ctx.beginPath(); ctx.arc(dx,dy,3,0,Math.PI*2); ctx.fill();
    }
  }

  // sensor dot at center
  ctx.fillStyle='#00ff4488';
  ctx.beginPath(); ctx.arc(CX,CY,10,0,Math.PI*2); ctx.fill();
  ctx.fillStyle='#00ff44';
  ctx.beginPath(); ctx.arc(CX,CY,6,0,Math.PI*2); ctx.fill();
  ctx.fillStyle='#1a5a1a';
  ctx.font='10px monospace';
  ctx.textAlign='left'; ctx.textBaseline='top';
  ctx.fillText('XE125', CX+14, CY-8);

  requestAnimationFrame(draw);
}
draw();

// WebSocket with auto-reconnect and heartbeat watchdog
let ws;
let lastMessage = Date.now();
```

```
// heartbeat check — if no message received for 5 seconds, force reconnec
t
setInterval(() => {
  if(Date.now() - lastMessage > 5000) {
    ws.close();
  }
}, 5000);

function connectWS() {
  ws = new WebSocket(`ws://${location.host}/ws`);
  ws.onopen = () => document.getElementById('status').textContent = 'conn
ected ✓';
  ws.onclose = () => {
    document.getElementById('status').textContent = 'disconnected — retry
ing...';
    detectedList = [];
    setTimeout(connectWS, 2000);  // retry after 2 seconds
  };
  ws.onmessage = (event) => {
    lastMessage = Date.now();  // reset heartbeat timer on every message
    const data = JSON.parse(event.data);
    if(data.angle !== undefined) currentAngle = data.angle;
    if(data.presence && data.values.length > 0){
      detectedList = data.values;
      document.getElementById('dist-display').textContent = data.values.m
ap(v => v.toFixed(1)+'cm').join('  |  ');
    } else {
      detectedList = [];
      document.getElementById('dist-display').textContent = '— cm';
    }
  };
}
connectWS();
</script>
</body>
</html>
)rawliteral";

// WebSocket event handler — required by AsyncWebSocket
void onWsEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
               AwsEventType type, void *arg, uint8_t *data, size_t len) {
}

void setup() {
  Serial.begin(115200);
  Serial.println("XE125 Radar Starting...");

  // I2C and radar init
```

```
  Wire.begin();
  bool success = radarSensor.begin(i2cAddress, Wire);
  if (success == false) {
    Serial.println("Radar not found - check wiring. Halting.");
    while (1);
  }
  int32_t setupError = radarSensor.detectorStart(MY_XM125_RANGE_START, MY
_XM125_RANGE_END);
  if (setupError != 0) {
    Serial.print("Radar setup error: ");
    Serial.println(setupError);
  }
  Serial.println("Radar ready");

  // Servo init - move to start position
  radarServo.attach(SERVO_PIN);
  radarServo.write(SERVO_MIN);
  delay(500);
  Serial.println("Servo ready");

  // WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Ready - open browser at: http://");
  Serial.println(WiFi.localIP());

  // WebSocket and web server
  ws.onEvent(onWsEvent);
  server.addHandler(&ws);
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });
  server.begin();
}

void loop() {
  // move servo one step and reverse at soft limits
  servoAngle += servoStep;
  if(servoAngle >= SERVO_MAX || servoAngle <= SERVO_MIN) servoStep = -ser
voStep;
  radarServo.write(servoAngle);
  delay(50);  // settle time before reading
```

```
  // read radar
  radarSensor.busyWait();
  presValError = radarSensor.getDistanceValuemm(distance);

  // package distance + angle into JSON and send to all connected browsers

  String json;
  if (presValError == 0 && distance > 0) {
    float cm = distance * 0.1;
    json = "{\"presence\":true,\"values\":[" + String(cm, 1) + "],\"angle
\":" + String(servoAngle) + "}";
    Serial.print("Presence YES - ");
    Serial.print(cm);
    Serial.print("cm @ ");
    Serial.print(servoAngle);
    Serial.println("deg");
  } else {
    json = "{\"presence\":false,\"values\":[],\"angle\":" + String(servoA
ngle) + "}";
  }

  ws.textAll(json);
  ws.cleanupClients();

  // keepalive ping every 5 seconds — prevents browser WebSocket idle tim
eout
  static unsigned long lastPing = 0;
  if(millis() - lastPing > 5000) {
    ws.textAll("{\"ping\":true}");
    lastPing = millis();
  }
}
```

7. How It Works — Data Flow

```
┌──────────────────────────────────────────────┐
|                   ESP32                      |
|                                              |
|  XE125 ──I2C──► getDistanceValuemm()         |
|  Servo ─────────► servoAngle (10°–170°)      |
|                      |                       |
|              JSON packet assembled           |
|    {"presence":true,"values":[142.3],"angle":45} |
|                      |                       |
|              ws.textAll(json) ─────► Browser |
|                                        |     |
|              Web server ─────────► index.html |
└──────────────────────────────────────────────┘

Browser:
  JSON received → currentAngle updated → beam redrawn
                → detectedList updated → red dot placed
```

8. Operation

1. Power the ESP32 via USB or powerbank
2. Wait for Serial monitor to print the IP address
3. Connect your phone or computer to the same WiFi network
4. Open browser and navigate to the printed IP address
5. Radar screen loads and begins displaying live data

Servo sweep: 10°–170° in 5° steps, reversing at each limit
Detection range: 20cm–300cm
Update rate: ~100ms per step

9. Troubleshooting

Problem	Likely Cause	Fix
XE125 not detected on Mac	Charge-only USB-C cable	Use data cable (e.g. iPad cable)
Serial port not showing	Wrong firmware or no driver	Check `ioreg` output for Acconeer XE125
I2C device not found	Wrong firmware on XE125	Flash `i2c_presence_detector.bin`
Servo not moving		Use `<ESP32Servo.h>` not

	Wrong library	`<Servo.h>`
Servo stuck at 179°	`servoAngle` starts at 0, below SERVO_MIN	Initialize `servoAngle = 10`
Browser shows disconnected	WebSocket idle timeout	Keepalive ping + auto-reconnect handles this
Upload fails in Arduino IDE	ESP32 USB driver issue	Export binary, flash with `esptool.py`

---

10. Next Steps

- **Two-unit system:** Field sensor unit (XE125 + servo + ESP32) communicates wirelessly via ESP-NOW to a handheld display unit (ESP32 + TFT screen) — no WiFi network required
- **TFT display:** Replace browser interface with onboard TFT screen for fully standalone operation
- **Persistent sweep map:** Retain detection dots across full sweep to build a complete 180° map
- **Multi-sensor fusion:** Add thermal camera for heat detection, combine with radar for robust classification